Reproducibility and Reusability in Deep Reinforcement Learning

Peter Henderson

Master of Science

School of Computer Science
McGill University
Montreal, Quebec, Canada

December 2017

A thesis submitted to McGill University in partial
fulfillment of the requirements of the degree of
Master of Science

©Peter Henderson, 2017

Abstract

Reinforcement learning (RL) has been shown to be an effective mechanism for learning complex tasks via interaction with the environment. Recent advances in combining deep neural networks with RL have resulted in powerful tools that outperform previous state-of-the-art methods for many domains including: robotics, video games, and board games. However, due to the interactive nature of these algorithms, as well as both intrinsic and extrinsic stochasticity, learning performance can be highly variant and difficult to reproduce. Furthermore, reusing information between tasks using these techniques can be problematic since they may overfit to a single task or environment. In this thesis, we investigate both *reproducibility* and *reusability* in deep RL. We begin by demonstrating the difficulty in reproducing a subset of deep RL algorithms: policy gradient methods for continuous control. We propose guidelines for rigorous experimental methodology and several statistical methods to help prevent misleading results. Next, we provide open-source reproducible environments for multitask RL. We evaluate simple sequential learning on sets of these tasks to show their effectiveness as benchmarks for multitask learning. Finally, we leverage these benchmark environments to investigate the notion of reusability. We focus on one-shot transfer learning in inverse RL. That is, given expert demonstrations from a mixture of environments with different dynamics is it possible to learn to properly complete a task in a previously unseen environment with different dynamics. To do this, we extend the options framework with the notion of reward options and develop a method for learning join reward-policy options in the context of generative adversarial inverse RL. This method is able to reuse information from a mixture of different environments to successfully learn a task in its current environment and significantly outperforms inverse RL without options.

Abrégé

L'apprentissage par renforcement (AR) s'est avéré être un mécanisme efficace pour apprendre à résoudre des tâches complexes par interaction avec l'environnement. Les progrès récents dans la combinaison de réseaux neuronaux profonds avec l'AR ont abouti à des outils puissants qui surpassent les précédentes méthodes de pointe dans des domaines tels que: la robotique, les jeux vidéo et les jeux de société. Cependant, en raison de la nature interactive de ces algorithmes, ainsi que de la stochasticité intrinsèque et extrinsèque, leurs performances d'apprentissage peuvent être très variables et difficiles à reproduire. En outre, la réutilisation d'informations entre plusieurs tâches à l'aide de ces techniques peut être problématique, car elles peuvent être adaptées à une tâche ou un environnement unique. Dans cette thèse, nous étudions à la fois la *reproductibilité* et la *réutilisabilité* dans l'AR profond. Nous commençons par démontrer la difficulté à reproduire un sous-ensemble d'algorithmes d'AR profonds: les méthodes de gradient de politique pour le contrôle continu. Nous proposons plusieurs méthodes basées sur l'analyse statistique pour aider à éviter les résultats trompeurs et donner des lignes directrices pour une méthodologie expérimentale plus rigoureuse. Ensuite, nous fournissons des environnements de référence reproductibles open-source pour l'apprentissage multi-tâche et évaluons l'apprentissage séquentiel simple sur des ensembles de tâches en tant que référence. Enfin, nous tirons parti de ces environnements de référence pour étudier la notion de réutilisabilité. Nous nous concentrons sur le transfert d'apprentissage en un coup dans l'AR inverse. C'est-à-dire, donné des démonstrations d'experts venant d'environnements variés avec des dynamiques différentes, il est possible d'apprendre à accomplir correctement une tâche dans un environnement inconnu avec des dynamiques différentes. Pour ce faire, nous poussons le cadre des options avec la

notion d'options de récompense et développons une méthode d'apprentissage des options de récompense-politique dans le contexte de l'AR adversariale génératrice. Cette méthode est capable de réutiliser des informations provenant d'un mélange d'environnements variés pour réussir l'apprentissage d'une tâche dans son environnement actuel et surclasse significativement le cas de l'approximateur unique.

Contributions

The main contributions of this thesis are tools and methods for reproducible and reusable deep reinforcement learning algorithmic and experimental methods. In particular, this thesis introduces the following:

- Experimental investigation of factors affecting reproducibility in deep RL methods (Chapter 3)
- Proposal of proper experimental procedures, evaluation methodologies, and future lines of investigations for reproducibility in RL (Chapter 3)
- Proposal and public release of reproducible benchmark environments for multitask learning in RL (Chapter 4)
- Extension of the options framework to reward options (Chapter 5)
- Creation of OptionGAN, an optionated version of Generative Adversarial Inverse Reinforcement Learning, for use of extended options framework in inverse RL (Chapter 5)
- Empirical results demonstrating this extension's effectiveness in continuous control tasks and one-shot transfer learning in inverse RL (Chapter 5)

The contributions described here have been published in several works during the course of the development of this thesis (Islam et al., 2017; Henderson et al., 2018b;

Henderson et al., 2017; Henderson et al., 2018a). As such, the co-authors of the related publications contributed in varying degrees to the relevant chapters of this thesis.

- The work in (Islam et al., 2017; Henderson et al., 2018b) correlates to Chapter 3. The co-authors on (Islam et al., 2017) were Riashat Islam, Maziar Gomrokchi, and Doina Precup – though this work is primarily not used here, the subsequent work (Henderson et al., 2018b) is based on this workshop paper. The co-authors of (Henderson et al., 2018b) were Riashat Islam, Philip Bachman, Joelle Pineau, David Meger, and Doina Precup.

- The work in (Henderson et al., 2017) correlates to Chapter 4. The co-authors of this work are Wei-Di Chang, Florian Shkurti, Johanna Hansen, David Meger, and Gregory Dudek.

- The work in (Henderson et al., 2018a) correlates to Chapter 5. The co-authors of this work are Wei-Di Chang, Pierre-Luc Bacon, David Meger, Joelle Pineau, and Doina Precup.

The author of this thesis was the primary author[1] on all related works and the co-authors acknowledge the use of these works in this thesis.

[1]The author of this thesis shares first authorship with Riashat Islam in (Henderson et al., 2018b; Islam et al., 2017).

Acknowledgements

This thesis could not have been completed without the support, advice, and generosity of many people.

Both of my supervisors, David Meger and Joelle Pineau, always go above and beyond in their support, constantly finding time and ways to help when I thought it would not be possible. They are role models for both their students and what supervisors should be, and I am extremely grateful for this. David Meger has been an amazing supervisor in encouraging me to pursue interesting problems, always being around for helpful and enlightening discussions, and understanding of an unusual path to the completion of this thesis. Equally, Joelle Pineau is a fantastic supervisor who is always there for her students no matter what, is a constant source of wise advice and steady support, and has helped me become a better researcher than I thought possible.

I would also like to thank all my co-authors on publications stemming from the ideas in this thesis. The discussions, encouragement, and help from all co-authors made these impactful ideas possible.

A special thank you to my family for inspiration and support throughout this experience. In particular, my mother, Julia, is a source of inspiration for overcoming adversity against all odds and I am grateful to Jieru, who has always been understanding, reassuring, and always finds time to read through last minute drafts for any mistakes.

Finally, I would like to thank the National Science and Engineering Research Council of Canada for the financial support which has made it possible for me to pursue this research.

Contents

List of Figures

List of Tables

Acronyms

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
ACKTR	Actor-Critic using Kronecker-Factored Trust Region
DDPG	Deep Deterministic Policy Gradients
DQN	Deep Q-Network (Learning)
ELU	Exponential Linear Unit
GAIL	Generative Adversarial Imitation Learning
GAN	Generative Adversarial Network
IRL	Inverse Reinforcement Learning
IRLGAN	Generative Adversarial Inverse Reinforcement Learning
KFAC	Kronecker Factorization
KL DIVERGENCE	Kullback-Leibler divergence
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multilayer Perceptron
MOE	Mixture-of-Experts
OPTIONGAN	Optionated Generative Adversarial Networks
PG	Policy Gradients
PPO	Proximal Policy Optimization
RBF	Radial Basis Function
RELU	Rectified Linear Unit
RL	Reinforcement Learning
TRPO	Trust Region Policy Optimization

Part I

Foundations

1

Introduction

Reinforcement learning (RL) provides a methodology for solving sequential decision-making problems through interaction with the environment. This technique has proven to be an effective method for accomplishing complex tasks such as: controlling continuous systems in robotics (Lillicrap et al., 2015), playing Go (Silver et al., 2016), Atari (Mnih et al., 2013), and competitive video games (Vinyals et al., 2017; Silva and Chaimowicz, 2017). To solve such complex scenarios, classic RL methods such as Q-Learning (Watkins, 1989) and Policy Gradient (PG) methods (Sutton et al., 2000) are combined with neural network function approximators to formulate deep RL techniques (as we will describe in Chapter 2). To illustrate the points in this section, let us use the example of a robot manipulator tasked with picking up a toy block. In such a case, RL optimization will try moving the manipulator in different ways until it successfully picks up the block (exploration). Then, the methodology will refine this policy to pick up the block consistently (exploitation). However, such methodologies can have various problems relating to *reproducibility* and *reusability*.

As the authors of (Cacioppo et al., 2015) state, "Reproducibility refers to the ability of a researcher to duplicate the results of a prior study using the same materials as were used by the original investigator. (...) Reproducibility is a minimum necessary condition for a finding to be believable and informative." We can use this as the basic definition of *reproducbility*, expanding upon it in Section 2.3. RL methods may experience highly variant learning due to their need for interaction with an environment, sequential objective func-

tions, and non-differentiable rewards. For example, returning to the robotic manipulator task, if the manipulator does not take the same exploratory actions in each learning trial, the time it takes to find the block my vary significantly. In one trial, it may accidentally come across the block quickly. In subsequent trial, it may explore the wrong region of space for a long time before finding the block. Furthermore, due to the large continuous space of actions and states in this task, the predictions as to which states and actions are valuable may vary highly between steps in a single learning trial. Several works investigate ways to address variance in RL methods, including using advantage estimation (Schulman et al., 2015, 2017, 2016; Mnih et al., 2016), n-step backups (Sutton and Barto, 1998; Mnih et al., 2016), and multiple function approximators (Van Hasselt et al., 2016; Bellemare et al., 2017).

However, just as important as addressing the variance in RL methods is addressing proper experimental techniques under highly variant conditions such that proper *reproducible* comparisons can be made between novel algorithms and baselines. In such conditions, where small perturbations to random initializations or changes in the environment can affect learning significantly, it is important to ensure proper experimental methodology to differentiate real improvements from favourable selection of conditions. To this end, this thesis investigates the factors affecting reproducibility in RL algorithms and suggests possible proper experimental methodologies based in statistical analysis when evaluating RL algorithms. To narrow the scope of our experiments, we focus on policy gradient methods for benchmark continuous control tasks.

Stemming from the problem of variance and reproducibility is the notion of reusability. We define *reusability* as the capability of reusing knowledge from one setting in another – an expanded definition is provided in Section 2.2. That is, can information be learned and reused for accomplishing different tasks (i.e., multitask, transfer, or lifelong learning as defined in Section 2.2). As demand drives systems to generalize to various domains and problems, the study of multitask, transfer and lifelong learning has become an increasingly important pursuit. In discrete domains, performance on the Atari game suite (Bellemare et al., 2013) has emerged as the *de facto* benchmark for assessing multitask learning. However, in continuous domains there is a lack of agreement on standard multitask evaluation environments which makes it difficult to compare different approaches fairly. This thesis

describes a benchmark set of tasks that we have developed in an extendable framework based on OpenAI Gym (Brockman et al., 2016). We run a simple baseline using Trust Region Policy Optimization (Schulman et al., 2015) and release the framework publicly to be expanded and used for the systematic comparison of multitask, transfer, and lifelong learning in continuous domains.

RL methods rely on a reward function to provide a signal that can be optimized. However, in most cases, currently, this reward function is hand-crafted. In our example robot manipulation task, somehow the environment must provide a reward to the RL agent such that it can know that it has successfully picked up the block. Currently, a human might hand-craft a reward function such that, for example, a reward of $+1$ is provided when a sensor in the block feels an upward velocity. While RL methods provide a useful way to solve tasks with a predefined objective, manually specifying a good reward function can be difficult, especially for intricate tasks. Inverse reinforcement learning (IRL) refers to the problem of learning this reward function from expert demonstrations. In the context of our example, the robotic manipulator could learn a reward signal from a human-controlled demonstration of the manipulator moving in the proper way to complete the task. We use our created multitask learning benchmark environments as a platform for proposing a novel method for one-shot transfer learning in IRL – that is, reusing information from many different experts in differing environments and conditions. Returning to our robotic manipulator, this notion of one-shot transfer learning would augment the IRL task such that the demonstrations may be from several different tasks. For example, the manipulator could be shown picking up objects on many different kinds of surfaces or in settings with obstacles impeding its progress. The manipulator must learn to infer the underlying structure of the task from these many demonstrations to successfully learn to complete a task in its current setting.

IRL offers a useful paradigm to learn the underlying reward function directly from expert demonstrations. Yet in reality, the corpus of demonstrations may contain trajectories arising from a diverse set of underlying reward functions rather than a single one. Thus, in IRL, it is useful to consider such a decomposition. The options framework in RL is specifically designed to decompose policies in a similar light. We therefore extend the options framework and propose a method to simultaneously recover reward options in addition to

policy options. We leverage adversarial methods to learn joint reward-policy options using only observed expert states. We show that this approach works well in both simple and complex continuous control tasks and shows significant performance increases in one-shot transfer learning. Thus, we present a novel method for reusability of demonstrations in different contexts through an extension of the options framework.

Overall, the goals of this thesis are two-fold: (1) to highlight and provide tools and methodologies for reproducible RL to drive the field forward and (2) to provide new methods for reusability by extending the options framework for decomposable reward functions and IRL.

2

Background

There are a number of concepts throughout this work which may require technical background. We introduce several core concepts here, revisiting and expanding them in each section as relevant.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is primarily a method for solving sequential decision-making problems. The material in this section provides a minimal overview of RL methodology required to understand the subsequent material. Sutton and Barto (1998) provide a thorough overview of RL, and the curious reader can further detailed conceptual explanations there.

Generally, RL provides a means for optimizing an entire sequence of decision-making events to achieve the overall best optimum across the predicted trajectory of decisions. Given a sequential decision-making task, RL algorithms leverage interaction with a task environment as a means to explore and exploit possible decisions, rather than requiring an explicitly pre-gathered dataset.

The central decison-making unit in an RL optimization problem is the RL *agent*. Given its current state, the agent can take an action within its *environment*. This provides the agent with some *reward*. This reward can be used to determine the effectiveness of its actions and optimize its decision-making *policy*. Its state information will also be updated such that it

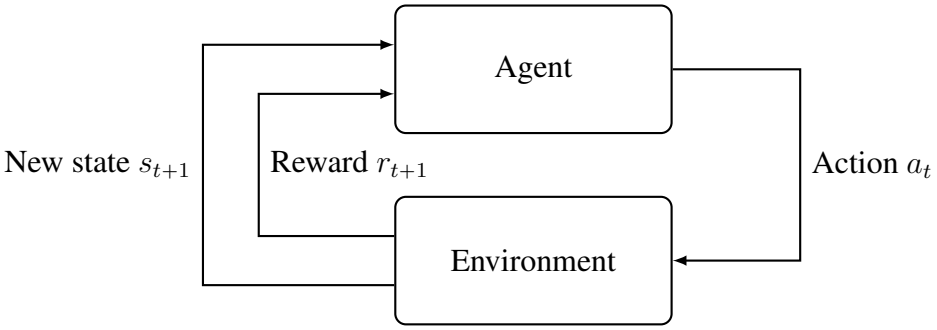

Figure 2.1: The core RL process. Reproduced with permission from: https://gist.github.com/pierrelux/6501790

can take a new action according to its updated state and policy. This process can be seen in Figure 2.1.

2.1.1 The Sequential Decision-Making Framework

We generally define RL in the context of the *Markov Decision Process* (MDP) (Bellman, 1957) – the environment that an RL agent can act in. MDPs consist of states S, actions A, a transition distribution $P : S \times A \rightarrow (S \rightarrow \mathbb{R})$, and a reward function $r : S \rightarrow \mathbb{R}$. We formulate our methods in the space of continuous control tasks ($A \in \mathbb{R}, S \in \mathbb{R}$) using measure-theoretic assumptions as described in Bacon et al. (2017). Thus we define a parameterized *policy* as the probability distribution over actions conditioned on states $\pi_\theta : S \times A \rightarrow [0, 1]$. The distribution of actions in a stochastic policy can be modeled by a stochastic policy $\pi_\theta(a|s)$. For continuous action spaces such a policy can be modeled by a Gaussian $\pi_\theta \sim \mathcal{N}(\mu, \sigma^2)$ where θ are the policy parameters. An agent can take an action a in its environment (the MDP) such that it transitions to a new state with some probability. Here, we focus on model-free methods where the probability of correctly transitioning to a new state is not explicitly known or defined.

2.1.2 Value Functions

The goal of an RL agent is to maximize the cumulative reward (also known as the return) across the trajectory of a given episode. We define an episode as the sequential MDP where there is a finite number of timesteps T that can be taken and a starting state at timestep t_0

which can be returned to. To maximize the cumulative reward of an episodic task, there must be a notion of how good a state is. This is known as the *value function* $V(s_t)$. Similarly, there must be a notion of how good an action is to take in a certain state. This is known as an *action-value function* $Q(s_t, a_t)$. Generally, the *advantage function* in this context can be defined as the advantage of an action over all other actions $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$. The value function of a policy (π) can be defined as $V_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t r_{t+1} | s_0 = s]$ and the action-value is $Q_\pi(s, a) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t r_{t+1} | s_0 = s, a_0 = a]$, where $\gamma \in [0, 1)$ is the discount factor. The discount factor is used to emphasize shorter term goals during optimization (discounting the effect of rewards many steps away). This can be leveraged for planning at different time horizons (François-Lavet et al., 2015). However, as is noted by Thomas (2014) and Schulman et al. (2016), the discount factor reduces variance at the cost of some bias. However, this is acceptable, as performance is usually better with the biased discount factor (as is specifically noted in the Discussion section of Bacon et al. (2017)).

2.1.3 Learning Value Functions

Sutton and Barto (1998) describe a myriad of methods for learning the aforementioned value functions. However, in this subsection we focus on the singular case of *temporal difference* (TD) learning, which will aid in future discussions. In the case of deep RL – as we focus on in this thesis – the value functions ($Q^\pi(s_t, a_t)$ and $V^\pi(s_t)$) can be modeled via neural networks. We will sometimes refer to these as *function approximators*. To explain and define TD learning, let us start with the specific example of deep Q-learning (Mnih et al., 2015). A neural network approximator for the value function can be referred to as a Q-network. To derive an update rule for this neural network based on an RL agents step tuple (s_t, a_t, r_t, s_{t+1}), we can start be defining the optimal action-value function $Q^*(s, a)$ as the maximum cumulative return achievable by any policy. Using the notation of (Mnih et al., 2015), the so-called *Bellman equation* states that the optimal action-value function must obey the following criterion:

$$Q^*(s, a) = \mathbb{E}_{s_{t+1} \in \mathcal{E}} \left[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) | s_t, a_t \right]. \tag{2.1}$$

Using this Bellman equation, an iterative update can be derived for the function approximator for the optimal value function. Given a function approximator for the optimal Q-value function ($Q(s, a; \theta)$), this update comes in the form:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s_t, a_t; \theta_i))^2\right] \tag{2.2}$$

where $y_i = \mathbb{E}_{s_{t+1}\in\mathcal{E}}\left[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{i-1})|s_t, a_t\right]$. Here, i is the update iteration and $\rho(s, a)$ is the behaviour distribution (i.e., the state-action pairs possible for a trajectory given the current policy for this $Q(s, a; \theta_i)$). In the Q-learning case, actions to collect samples for this update are taken by sampling an ϵ-greedy. That is, with probability $1-\epsilon$ samples are collected according to the greedy strategy, $a_t = \max_a Q(s, a; \theta)$, and with probability ϵ the action is sampled at random. It can be noted that the error in Equation 2.2 is the TD error for the action-value function, hence Q-learning is a form of TD learning. Importantly, the action selection strategy described here in Q-learning requires a discrete action space so as to enumerate the action with the maximum Q-value. However, as will be subsequently described, other methods can make use of a continuous action space by the action selection method in the form of a policy.

2.1.4 Policy Gradient Methods

Throughout this work, we focus on Policy Gradient (PG) (Sutton et al., 2000) methods. This allows for the modeling of both continuous and discrete action spaces. Rather than using the value function to take actions, these methods demonstrate a way to explicitly optimize a parameterized policy π_θ through stochastic gradient ascent. In the discounted setting, PG methods optimize $\rho(\theta, s_0) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r(s_t)|s_0\right]$. To derive a gradient update rule, the PG theorem states: $\frac{\delta\rho(\theta,s_0)}{\delta\theta} = \sum_s \mu_{\pi_\theta}(s|s_0) \sum_a \frac{\delta\pi_\theta(a|s)}{\delta\theta} Q_{\pi_\theta}(s, a)$, where $\mu_{\pi_\theta}(s|s_0) = \sum_{t=0}^{T} \gamma^t P(s_t = s|s_0)$ (Sutton et al., 2000).

REINFORCE and the Likelihood Ratio

Given some Monte Carlo rollouts τ of a sequence decision making process consisting of tuples (s_t, a_t, r_t, s_{t+1}), the gradient with respect to the policy parameters can be estimated using the log likelihood ratio (Aleksandrov et al., 1968; Glynn, 1987). This became better

known as REINFORCE (Williams, 1992). This policy gradient algorithm uses an update for the policy parameters based on the log likelihood such that $\Delta\theta_t = \alpha\nabla_\theta \log \pi_\theta(a_t|s_t)v_t$. In this case v_t is the Monte Carlo return for timestep t and is an unbiased sample of $Q^\pi(s_t, a_t)$.

Trust Region and Proximal Policy Optimization

In Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) and Proximal Policy Optimization (PPO) (Schulman et al., 2017) this Monte Carlo REINFORCE update is constrained and transformed into the advantage estimation view such that the above becomes a constrained optimization problem

$$\max_\theta \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t(s_t, a_t) \right], \tag{2.3}$$

subject to $\mathbb{E}_t \left[\mathrm{KL} \left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t) \right] \right] \leqslant \delta$ where $A_t(s_t, a_t)$ is the generalized advantage function (GAE) according to Schulman et al. (2016). This is defined as

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{T} (\gamma\lambda)^l \delta_{t+l}, \tag{2.4}$$

where $\delta_{t+l} = -V(s_t) + r_t + \gamma r_{t+1} + .. + \gamma^k V(s_{t+k})$. Advantage estimation is used to reduce the variance of updates. Schulman et al. (2016) provide an analysis of possible methods for expressing a policy gradient update, where $Q(s, a)$ can be replaced with slightly different expressions. Which can add some bias in exchange for variance reduction. The authors show a correlation between the different expressions and derive a generalized formula as seen in Equation 2.4. It can be noted that the λ and γ are hyperparameters which allow bias-variance tradeoff to be tuned to some extent.

In TRPO, this is solved as a constrained conjugate gradient descent problem, while in PPO the constraint is transformed into a penalty term or clipping objective. In both of these methods, the value function is found via Monte Carlo rollouts (that is, trying out the policy in the environment for the life of the trajectory).

Actor-Critic Methods

While TRPO and PPO use constrained REINFORCE-style updates where the advantage function is estimated via Monte Carlo sampling, another option is to use the *actor-critic* methodology. In this context, a policy (the actor) is learned by using another function approximator for the value function (the critic) in the policy update rather than using full Monte Carlo rollouts. Then, the value function (critic) is updated according to the TD error, just as described in Section 2.1.3. Compared to the solely value-based method of Q-learning with an ϵ-greedy strategy, the actor-critic methodology allows for learning the value function and a complementary action selection strategy in both continuous and discrete action spaces.

To elucidate how actor-critic methods work, we will focus on and describe advantage actor-critic (A2C). This is the synchronous version of asynchronous advantage actor-critic described in (Mnih et al., 2016). In the asynchronous version, multiple agents execute in an environment at once using the same policy and gradient updates are calculated in each thread asynchronously and then aggregated together. To simplify the explanation of the A3C updates, we instead present the synchronous version where Monte Carlo samples are collected with a single policy and the gradients are computed in a synchronous manner with these samples. The synchronous version, A2C, is seen in works such as (Schulman et al., 2017; Wu et al., 2017) as a baseline comparison.

The general idea of A2C is to take N steps in the environment where N can be defined by the researcher as a hyperparameter. These N-steps are collected into a batch. The advantage function is then calculated based on a critic value function $V_\theta(s_t)$ such that for a given sample the advantage is defined as:

$$A_\theta(s_t, a_t) = \sum_{i=0}^{N-1} \gamma^i r_{t+i} \gamma^k V_\theta(s_{t+k}) - V_\theta(s_t). \tag{2.5}$$

Using this advantage function, the same policy gradient update as in REINFORCE can be used – that is, the gradient of $\log \pi_{\theta'}(a_t|s_t)(A_\theta(s_t, a_t))$ with respect to the policy parameters θ'. Subsequently, the value function parameters are updated according to the TD-error, just as in Q-learning. That is V_θ is updated with the gradient of

$$\left(\sum_{i=0}^{N-1} \gamma^i r_{t+i} \gamma^k V_\theta(s_{t+k}) - V_\theta(s_t)\right)^2 \tag{2.6}$$

with respect to the value function parameters θ.

There is actually some notable similarity between the two methods of advantage estimation in PPO and A2C. A key difference between them is the reliance on the critic approximation in the case of A2C. For A2C, N is typically small – for example, in the OpenAI baselines implementation, $N = 5$ by default (Hesse et al., 2017). Thus the majority of the gradient signal comes from the critic. PPO, on the other hand, typically collects full Monte Carlo rollouts of a complete episode, thus emphasizing the use of the value function approximator as a baseline rather than as the main gradient signal. Hence, actor-critic methods are a blend of policy gradients and value function learning which can be used to provide more sample efficient methodologies.

Actor Critic Kronecker-factorized Trust Region Policy Optimization

In the case of Actor Critic using Kronecker-Factored Trust Region (ACKTR) (Wu et al., 2017), the same methodology of A3C (Mnih et al., 2016) is followed. However, instead of the unconstrained REINFORCE-style policy updates of A3C, ACKTR adds trust region optimization and distributed Kronecker factorization (KFAC) for improved sample efficiency and scalability. The use of natural gradients with constraints mirrors the trust regions of TRPO. However, TRPO uses Hessian-free optimization via conjugate gradient descent, whereas ACKTR uses KFAC updates which are less computationally expensive. Wu et al. (2017) claim that this allows for the sample efficiency of actor-critic to be combined with the stability of trust region methods, while still being computationally inexpensive (as compared to conjugate gradient descent methods).

Deep Deterministic Policy Gradients

Deep deterministic policy gradients (DDPG) (Lillicrap et al., 2015) is another actor-critic method which uses Q-learning and a deterministic version of the policy gradient theorem to learn a *deterministic* policy (that is, $a = \pi(s)$) in a continuous action space. In this

case, the action-value function (the critic) is learned via TD learning. That is, a parameterized $Q^{\pi}_{\theta}(s,a)$ is learned by optimizing the negative TD error (for example, in Q-learning, $E_{TD} = Q^{\pi}_{\theta}(s_t,a_t) - (r_t + \gamma \max Q^{\pi}_{\theta}(s_{t+1},a_{t+1}))$. Once the action-value function is learned, the parameterized policy is optimized according to the parameterized critic. In the case of DDPG, this is done via a deterministic version of the policy gradient theorem. In this case, the gradient of the action-value function approximator with respect to a given batch of N sampled state-action pairs is used to update the policy via the chain rule. Thus the sampled policy gradient ($\nabla_{\theta} J$) takes the form:

$$\nabla_{\theta} J \approx \frac{1}{N} \sum_{i} \nabla_{a} Q(s,a|\theta^{Q})|_{s=s_i, a=\pi(s_i)} \nabla_{\theta^{\pi}} \pi(s|\theta^{\pi})|_{s_i}. \tag{2.7}$$

Often, this allows for the agent to avoid Monte Carlo rollouts, and instead optimize the actor and critic with each step in the environment (i.e., online learning). However, this comes at the cost of increased variance as seen in (Henderson et al., 2018b).

2.1.5 The Options Framework

In interactive tasks, different courses of action may be needed at different parts of the state space or at different time scales. For example, let us examine a robotics task, where a robot must pick up dishes from a sink and place them in a drying rack. In such a complex task, four broad courses of action can be defined: moving to the sink, grasping a dish, moving it to the drying rack, and placing it properly. The concept of temporal abstraction in RL allows for the division of such courses at different time scales. The options framework (Sutton et al., 2000; Precup, 2000) provides a useful paradigm for learning and planning such courses of action using this notion of temporal abstraction. As aforementioned, these components are particularly useful for reusability as different temporally abstracted options can be reused in different tasks – for example, as in Hawasly and Ramamoorthy (2013).

In RL, an option ($\omega \in \Omega$) can be defined by a triplet ($I_{\omega}, \pi_{\omega}, \beta_{\omega}$). In this definition, π_{ω} is called an intra-policy option, $I_{\omega} \subseteq S$ is an initiation set, and $\beta_{\omega} : S \rightarrow [0,1]$ is a termination function (i.e., the probability that an option ends at a given state) (Sutton et al., 1999). Furthermore, π_{Ω} is the policy-over-options. That is, π_{Ω} determines which option

π_ω an agent picks to use until the termination function β_ω indicates that a new option should be chosen. Many works have investigated different methods for learning options, including (Bacon et al., 2017; Stolle and Precup, 2002; Sutton et al., 1999; Precup, 2000). As we base our work in reusability within the context of the option-critic framework (Bacon et al., 2017), we will focus on introducing this as an example method for learning options in RL.

Algorithm 1: The Option-Critic framework for Q-Learning. Reproduced with permission from (Bacon et al., 2017).

$s \leftarrow s_0$ Choose ω according to an ϵ-soft policy-over-options ($\pi_\Omega(s_t)$) **repeat**
  Choose a_t according to $\pi_{\omega,\theta}(a_t|s_t)$;
  Take action a_t in state s_t, observe s_{t+1}, r_t;
  $\delta \leftarrow r - Q_U(s_t, \omega, a_t)$
  if s_{t+1} *is non-terminal* **then**
    $\delta \leftarrow \delta + \gamma(1 - \beta_{\omega,\vartheta}(s_{t+1}))Q_\Omega(s_{t+1}, \omega) + \gamma\beta_{\omega,\vartheta}(s_{t+1}) \max_{\bar{\omega}} Q_\Omega(s_{t+1}, \bar{\omega})$
  end
  $Q_U(s_t, \omega, a_t) \leftarrow Q_U(s_t, \omega, a_t) + \alpha\delta$
  $\theta \leftarrow \theta + \alpha_\theta \frac{\delta \log \pi_{\omega,\theta}(a_t|s_t)}{\delta\theta}(Q_\Omega(s_{t+1}), \omega) - V_\Omega(s_{t+1}))$
  $\vartheta \leftarrow \vartheta - \alpha_\vartheta \frac{\delta\beta_{\omega,\vartheta}(s_{t+1})}{\delta\vartheta}(Q_\Omega(s_{t+1}), \omega) - V_\Omega(s_{t+1}))$
  if $\beta_{\omega,\vartheta}$ *terminates in* s_{t+1} **then**
    choose new ω according to ϵ-soft policy-over-options $s_t \leftarrow s_{t+1}$
  end
until s_{t+1} *is terminal*;

The execution model in the option-critic framework is called the *call-and-return* option execution model. That is, an option ω is picked by the policy-over-options π_Ω. The agent then follows the intra-option policy π_ω until termination is indicated by the termination function β_ω. This procedure is then repeated until the episode is terminated. To learn the optimal parameterized termination function $\beta_{\omega,\vartheta}$ and intra-option policies $\pi_{\omega,\theta}$ (parameterized by ϑ and θ respectively), an update rule can be derived. We summarize this derivation quickly here. We can start with the option-value function: $Q_\Omega(s, \omega) = \sum_a \pi_{\omega,\theta}(a|s)Q_U(s, \omega, a)$. Here, $Q_U : S \times \Omega \times A \rightarrow \mathbb{R}$ is the value of an action with a state option pair: $Q_U(s_t, \omega, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)U(\omega, s_{t+1})$.

Here, $U : \Omega \times S \rightarrow \mathbb{R}$ is the option-value function upon entering a state s_{t+1}, or "upon arrival" (Bacon et al., 2017; Sutton et al., 1999). This is given by: $U(\omega, s_{t+1}) =$

$(1 - \beta_{\omega,\vartheta}(s_{t+1}))Q_{\Omega(s_{t+1},\omega)} + \beta_{\omega,\vartheta}(s_{t+1})V_{\Omega}(s_{t+1})$. Using this key set of equations, the gradients with respect to ϑ, θ can be calculated with respect to the option-value function (i.e., the critic). These update rules are summarized in Algorithm 1, as replicated from (Bacon et al., 2017). As can be seen, these gradient based updates are very similar to the actor-critic framework previously mentioned. In this case, though a set of policies (options) are learned in one step instead of a singular policy. Option-Critic has been shown to improve performance in Atari tasks, and as we will see in Chapter 5, learning options can be particularly useful for reusability.

2.1.6 Inverse Reinforcement Learning

While in most scenarios where RL is currently used, the reward function for an MDP is assumed to be known, a proper reward specification can be extremely difficult to handcraft – especially for complex scenarios in continuous control. As such, much recent work has investigated *Inverse Reinforcement Learning (IRL)*. That is, a reward function is learned from demonstrations of an expert agent – possibly a human – performing the proper actions in an MDP.

IRL was first formulated in the context of an MDP by Ng and Russell (2000). In this framework, learning a reward function generally refers to finding a reward function of the expert R^* such that $\mathbb{E}\left[\sum_t \gamma^t R^*(s_t)|\pi^*\right] \geqslant \mathbb{E}\left[\sum_t \gamma^t R^*(s_t)|\pi\right], \forall \pi$. One of the key insights of Ng and Russell (2000) was to point out that this problem leaves ambiguity in the reward function. Since the expert policy π^* is typically unknown, only knowing traces or samples from the policy, this leaves room for several solutions. For example, in such a setting $R = 0$ is a solution.

In later work, a parametrization of the reward function is learned as a linear combination of the state feature expectation so that the hyperdistance between the expert and the novice's feature expectation is minimized (Abbeel and Ng, 2004). It has also been shown that a solution can be formulated using the maximum entropy principle, with the goal of matching feature expectation as well (Ziebart et al., 2008). Such feature matching approaches rely on the theorem from Abbeel and Ng (2004) that for a policy π to perform as well as the expert policy π^*, the feature expectations must match $||\mu(\pi) - \mu(\pi^*)||_1 \leqslant \epsilon$.

This implies that for all w with $||w||_\infty \leqslant 1$, $|w^{*T}\mu(\pi) - w^{*T}\mu(\pi^*)| \leqslant \epsilon$. That is, there can be an optimal mapping of feature vectors. This allows the formulation of an optimization problem as described by Abbeel and Ng (2004) and Ziebart et al. (2008).

Generative adversarial imitation learning (GAIL) (Ho and Ermon, 2016) is a more recent formulation of IRL which makes use of adversarial techniques from Goodfellow et al. (2014) to perform a similar feature expectation matching with deep neural networks. In this case, a discriminator uses state-action pairs (transitions) from the expert demonstrations and novice rollouts to learn a binary classification probability distribution. The probability that a state belongs to an expert demonstration can then be used as the reward for a policy optimization step. A more detailed description of this technique is in Section 5.1 of this thesis.

2.1.7 Benchmark Tasks and Domains

There are many sequential decision-making problems where RL can be applied. Due to its interactive nature, the development of novel RL methods has recently mainly taken place in simulated environments. Board-games (Silver et al., 2016), Atari Games (Mnih et al., 2013), and real-time strategy games (Vinyals et al., 2017) have been used as benchmarking environments for RL algorithm development. However, typically these involve discrete decision-making problems (choosing an action from a finite set). While RL methods have shown great success in discrete action spaces (e.g., playing Atari (Mnih et al., 2015)), new techniques demonstrate viable alternatives to model-based learning in continuous control tasks such as robotic manipulation or locomotion (Schulman et al., 2015; Lillicrap et al., 2015; Schulman et al., 2016, 2017; Ho and Ermon, 2016). Such continuous control tasks add another layer of complexity by requiring an action to be chosen from a large continuous action space.

Such tasks are particularly desirable for the notion of reusability due to the complex and interconnected nature of learning robotics-based tasks (Finn et al., 2017; Christiano et al., 2016; Gupta et al., 2017; Rusu et al., 2017; Higuera et al., 2017). In these settings, many core actions can be reused in the continuous action space. For example, moving a robotic manipulator from one place to another can be a learned macro-action that can be reused in

many different tasks. Due to these complexities and the particularly relevant nature of reuse in these settings, we focus on such continuous control tasks and environments throughout this thesis.

To this end, OpenAI Gym (Brockman et al., 2016) provides several benchmark environments in continuous control settings that we use and expand upon. These are built with the MuJoCo (Todorov et al., 2012) simulator. The main environments on which we focus here are ones involving locomotion: Hopper, HalfCheetah, Walker2d, Humanoid, and Swimmer. The goal of all these agents is to learn to move at the fastest horizontal rate possible, encouraging the development of useful gaits. Briefly, the Hopper environment consists of a single leg that must learn to hop at the fastest velocity possible in the horizontal direction. The HalfCheetah is a bipedal agent with cheetah-like features. Walker2d is a bipedal agent with humanoid legs with no torso. Humanoid is a full anthropomorphic agent with a much larger state and action space. Swimmer is a simple snake-like agent which must learn locomotion in a fluid like environment. In all cases the state space consists of the joint angles and location, while the action space consists of torque applied to the joints. These environments are meant to scale in complexity as the number of joints increase and thus the coordination required to learn a proper gait is multiplied.

2.2 Reusability: Multitask, Transfer, Lifelong Learning

We can define *reusability* in the context of RL as the capability of reusing knowledge from one setting in another (as in transfer learning or multitask learning) through the identification of underlying structure in the underlying policy space. According to (Fernández and Veloso, 2013) the notion of *policy reuse* has these properties:

> Policy Reuse identifies classes of similar policies revealing a basis of core-policies of the domain. (...) In general, Policy Reuse contributes to the overall goal of lifelong reinforcement learning, as (i) it incrementally builds a policy library; (ii) it provides a mechanism to reuse past policies; and (iii) it learns an abstract domain structure in terms of core-policies of the domain.

2.2 Reusability: Multitask, Transfer, Lifelong Learning

We can extend this notion of policy reuse to the context of IRL, by adding to definition the notion of *one-shot transfer learning in IRL*: the concept of transferring knowledge from expert demonstrations in different settings (*exempli gratia*, tasks or environment dynamics) by learning in a new environment with no demonstrations of the new setting. We call this *one-shot* since the novice agent must reuse or transfer information from the expert settings while learning in one-shot on the new environment.

The concept of reusability is particularly useful in the context of multitask, transfer, or lifelong learning. Generally, in these categories of learning, an agent must be able to use or distill knowledge for many tasks into a single policy (though it may be decomposed internally into several sub-policies). This can come in the form of sequential learning (e.g., learning one task at a time while not forgetting the previous tasks) or one-shot (where you can transfer knowledge already known from a different task to the current task at hand or where you learn many tasks at once).

Multitask learning has been defined in several ways in relevant RL literature. For example, in (Calandriello et al., 2014), it is defined as where "the objective is to simultaneously learn multiple tasks and exploit their similarity to improve the performance with respect to single-task learning." While, in (Wilson et al., 2007) it is stated that multitask reinforcement learning is, "where an agent is confronted with a sequence of MDPs chosen independently from a fixed distribution. The agent's goal is to quickly find an optimal policy for each MDP." Several other works in multitask learning for RL derive similar definitions to these two sources (Yang et al., 2017; Mujika, 2016; Teh et al., 2017). Throughout this work, we generally refer to multitask learning as learning to execute multiple tasks while exploiting their similarity either in a sequential fashion or simultaneously.

The sequential version of this multitask learning problem is also referred to as the *lifelong learning* problem. Thrun (1995) provides an eloquent definition, stating that, "Lifelong learning addresses situations where a learner faces a stream of learning tasks. Such scenarios provide the opportunity for synergetic effects that arise if knowledge is transferred across multiple learning tasks." The lifelong learning problem in RL has witnessed many different approaches to varying degrees of success (Ammar et al., 2015b; Brunskill and Li, 2014; Ammar et al., 2015a).

2.2 Reusability: Multitask, Transfer, Lifelong Learning

A single step in lifelong learning can be called *transfer learning*. That is, can knowledge be transferred from solving known problems to a new, previously unseen, problem or environment. A partial overview of transfer learning in the RL setting can be found in (Taylor and Stone, 2009) and several newer works combine deep learning transfer methodologies with RL techniques (Barreto et al., 2017; Parisotto et al., 2015).

Such scenarios are particularly useful for robotics or other continuous control tasks. In these settings, complex controllers must be learned to encompass the fine-grained nature of continuous action spaces. Transferring knowledge or reusing knowledge between tasks or sub-parts of a task is necessary to build efficient controllers and solve complex scenarios. Many such works also investigate transferring knowledge from simulation to the real world (Higuera et al., 2017).

Several works investigate multitask, transfer, or lifelong learning with MuJoCo (Todorov et al., 2012) simulated continuous control tasks. These tasks include: navigating around a wall (where a wall separates an agent from its goal); the OpenAI Gym Reacher environment with an added image state space of the environment; jumping over a wall using a model similar to the OpenAI Half-Cheetah environment (Finn et al., 2017); varying the gravity of various standard OpenAI Gym benchmark environments (Reacher, Hopper, Humanoid, HalfCheetah) and transferring between the modified environments; adding motor noise to the same set of environments (Christiano et al., 2016); simulated grasping and stacking using a Jaco arm (Rusu et al., 2017); and several custom grasping and manipulation tasks to demonstrate learning invariant feature spaces (Gupta et al., 2017).

Other works investigate using classical control systems and robotics simulations with a set of varied hyperparameters for each environment. These include: a simple mass spring damper task, cart-pole with continuous control; a three-link inverted pendulum with continuous control; a quadrotor control task (Ammar et al., 2014); a double-linked pendulum task; a modified cartpole balancing task which can transfer to physical system (Higuera et al., 2017).

2.3 Reproducibility

As Cacioppo et al. (2015) state, "Reproducibility refers to the ability of a researcher to duplicate the results of a prior study using the same materials as were used by the original investigator. (...) Reproducibility is a minimum necessary condition for a finding to be believable and informative." In the context of deep RL, we can extend existing definitions (Cacioppo et al., 2015; Buckheit and Donoho, 1995) to define reproducibility as the ability of a researcher to duplicate prior results, particularly such that algorithm performance holds in similar environmental conditions, complete software is released to reproduce results, a complete description of all methodologies and settings used to generate the reported results is provided, and reported results constitute a fair comparison against other algorithms. The notion of reproducibility in scientific literature has recently gained notoriety as many potentially breakthrough works and experiments have been difficult to replicate across many fields (Baker, 2016; Collins and Tabak, 2014; Kenall et al., 2015; Henderson et al., 2018b). However, this lack of replication is likely not due to fraudulent behaviour by the original authors of non-reproducible works, but rather a deficit in proper experimental methodological theory or practices. As the authors of (Collins and Tabak, 2014) eloquently describe it:

> Let's be clear: with rare exceptions, we have no evidence to suggest that irreproducibility is caused by scientific misconduct. In 2011, the Office of Research Integrity of the US Department of Health and Human Services pursued only 12 such cases. Even if this represents only a fraction of the actual problem, fraudulent papers are vastly outnumbered by the hundreds of thousands published each year in good faith.
>
> Factors include poor training of researchers in experimental design; increased emphasis on making provocative statements rather than presenting technical details; and publications that do not report basic elements of experimental design. Crucial experimental design elements that are all too frequently ignored include blinding, randomization, replication, sample-size calculation and the effect of sex differences. And some scientists reputedly use a 'secret sauce' to make their experiments work – and withhold details from publication or

> describe them only vaguely to retain a competitive edge. What hope is there that other scientists will be able to build on such work to further biomedical progress?

While this problem plagues the machine learning (ML) and other scientific communities (Wagstaff, 2012; Boulesteix et al., 2013; Stodden et al., 2014; Bouckaert and Frank, 2004; Bouckaert, 2004; Vaughan and Wawerla, 2012), it is exacerbated by the interactive nature of RL. Because small changes in stochastic environments can take significantly different directions in learning, it is extremely difficult to reproduce results across various different random initializations. As environment dynamics become more stochastic, these problems are only made worse. While the best evaluation techniques and experimental procedures in this context are still open questions, the work presented here attempts to take steps toward addressing the problems of reproducibility in RL, encouraging and suggesting proper experimental practice, and providing an initial suggestion of evaluation metrics and statistical measures which account for difficult-to-reproduce learning performance.

Part II

Reproducibility

3

Reproducibility in Deep Reinforcement Learning

Reinforcement learning (RL) is the methodology of how an agent can interact with its environment to learn a policy which maximizes expected cumulative rewards for a task. Recently, RL has experienced dramatic growth in attention and interest due to promising results in several areas: controlling continuous systems in robotics (Lillicrap et al., 2015), playing Go (Silver et al., 2016), Atari (Mnih et al., 2013), and competitive video games (Vinyals et al., 2017; Silva and Chaimowicz, 2017). Figure 3.1 illustrates growth of the field through the number of publications per year. To maintain rapid progress in RL research, it is important that existing works can be easily reproduced and compared to accurately judge improvements offered by novel methods.

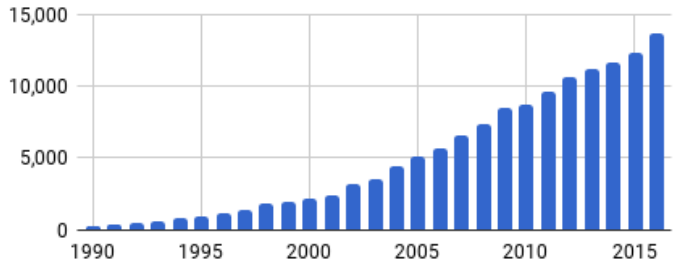

Figure 3.1: Growth of published RL papers. Shown are the number of RL-related publications (y-axis) per year (x-axis) scraped from Google Scholar searches.

However, reproducing deep RL results is seldom straightforward, and the literature reports a wide range of results for the same baseline algorithms (Islam et al., 2017). Reproducibility can be affected by extrinsic factors (e.g., hyperparameters or codebases) and intrinsic factors (e.g., effects of random seeds or environment properties). We investigate these sources of variance in reported results through a representative set of experiments. For clarity, we focus our investigation on policy gradient (PG) methods in continuous control. Policy gradient methods with neural network function approximators have been particularly successful in continuous control (Schulman et al., 2015, 2017; Lillicrap et al., 2015) and are competitive with value-based methods in discrete settings. We note that the diversity of metrics and lack of significance testing in the RL literature creates the potential for misleading reporting of results. We demonstrate possible benefits of significance testing using techniques common in machine learning and statistics.

Several works touch upon evaluating RL algorithms. In (Duan et al., 2016), the authors benchmark several RL algorithms and provide the community with baseline implementations. Generalizable RL evaluation metrics are proposed in (Whiteson et al., 2011). Another work (Machado et al., 2018) revisits the Arcade Learning Environment to propose better evaluation methods in these benchmarks. However, while the question of *reproducibility and good experimental practice* has been examined in related fields (Wagstaff, 2012; Boulesteix et al., 2013; Stodden et al., 2014; Bouckaert and Frank, 2004; Bouckaert, 2004; Vaughan and Wawerla, 2012), to the best of our knowledge this is the first work to address this important question in the context of deep RL.

In each section of our experimental analysis, we pose questions regarding key factors affecting reproducibility. We find that there are numerous sources of non-determinism when reproducing and comparing RL algorithms. To this end, we show that fine details of experimental procedure can be critical. Based on our experiments, we conclude with possible recommendations, lines of investigation, and points of discussion for future works to ensure that deep reinforcement learning is reproducible and continues to matter.

3.1 Technical Background

This work focuses on several model-free policy gradient algorithms with publicly available implementations which appear frequently in the literature as baselines for comparison against novel methods. Section 2.1 of this thesis reviews these methods in more detail, but we briefly recap here. We experiment with Trust Region Policy Optimization (TRPO) (Schulman et al., 2015), Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015), Proximal Policy Optimization (PPO) (Schulman et al., 2017), and Actor Critic using Kronecker-Factored Trust Region (ACKTR) (Wu et al., 2017). These methods have shown promising results in continuous control MuJoCo domain tasks (Todorov et al., 2012) from OpenAI Gym (Brockman et al., 2016). Generally, they optimize

$$\rho(\theta, s_0) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)|s_0\right],$$

using the policy gradient theorem:

$$\frac{\delta\rho(\theta, s_0)}{\delta\theta} = \sum_s \mu_{\pi_\theta}(s|s_0) \sum_a \frac{\delta\pi_\theta(a|s)}{\delta\theta} Q_{\pi_\theta}(s, a).$$

Here, $\mu_{\pi_\theta}(s|s_0) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s|s_0)$ (Sutton et al., 2000). TRPO (Schulman et al., 2015) and PPO (Schulman et al., 2017) use constraints and advantage estimation to perform this update, reformulating the optimization problem as:

$$\max_\theta \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t(s_t, a_t)\right].$$

In this case, A_t is the generalized advantage function (Schulman et al., 2016). TRPO uses conjugate gradient descent as the optimization method with a KL constraint:

$$\mathbb{E}_t\left[\mathrm{KL}\left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right] \leqslant \delta.$$

PPO reformulates the constraint as a penalty (or clipping objective). DDPG and ACKTR use actor-critic methods which estimate $Q(s, a)$ and optimize a policy that maximizes the Q-function based on Monte-Carlo rollouts. DDPG does this using deterministic policies, while ACKTR uses Kronecketer-factored trust regions to ensure stability with stochastic policies.

3.2 Experimental Analysis

We pose several questions about the factors affecting reproducibility of state-of-the-art RL methods. We perform a set of experiments designed to provide insight into the questions posed. In particular, we investigate the effects of: specific hyperparameters on algorithm performance if not properly tuned; random seeds and the number of averaged experiment trials; specific environment characteristics; differences in algorithm performance due to stochastic environments; differences due to codebases with most other factors held constant.

Hyperparameters – that is, configurable parameters not learned in the model – can be crucial in deep learning algorithm performance. For example, in (Melis et al., 2018) it is shown that a simple Long-Term Short-Term memory networks can outperform complex neural network architectures when hyperparameters are properly tuned. The significant effect of these settings, as we demonstrate in this section of the thesis, can be found in RL.

Random seeds can also have a large effect on algorithm performance. In fact, Bengio (2012) suggests tuning random seeds in deep learning algorithms as one would any hyperparameter. As we will show here, random seeds have a significant effect on performance in RL algorithms. This can be problematic when coupled with few numbers of reported random seeds. Tuning random seeds in RL does not constitute a fair comparison, as this can augment the environment conditions (similar to changing the dataset for comparisons in ML).

Furthermore, as RL algorithms can be considered optimization methods, environments are comparable to defining datasets and loss functions in deep learning. The optimization method may not perform as well in all settings, hence we investigate the effect of the task

at hand when comparing different algorithms. We also investigate ways in which these properties might cause performance differences.

Finally, reimplementations of the same algorithm across different publications can result in performance variations due to uncaught code errors or underlying differences in code libraries. However, in research publications, to ensure a fair comparison between algorithms, it is necessary that the used implementation performs as well as the original publication. As such, we compare different algorithm implementations used for baseline comparison in a number of works to determine if the codebases affect the fair reporting of results.

For most of our experiments[1], except for those comparing codebases, we use the OpenAI Baselines[2] implementations of the following algorithms: ACKTR (Wu et al., 2017), PPO (Schulman et al., 2017), DDPG (Plappert et al., 2018), TRPO (Schulman et al., 2017). We use two benchmark MuJoCo (Todorov et al., 2012) environments from OpenAI Gym (Brockman et al., 2016): Hopper-v1 and HalfCheetah-v1. These two environments provide contrasting dynamics (the former being more unstable). We choose these algorithms as they have open-source implementations which are used in several publications. We choose the environments due to their contrasting characteristics. For example, the Hopper-v1 environment provides more unstable dynamics, so the agent can fail an episode easily if exploration is done in an unsafe manner. Conversely, the HalfCheetah-v1 environment provides stable dynamics where exploration can potentially be beneficial.

To ensure fairness we run five experiment trials for each evaluation, each with a different preset random seed (all experiments use the same set of random seeds). In all cases, we highlight important results here, with full descriptions of experimental setups and additional learning curves included in Appendix A.2. Unless otherwise mentioned, we use default settings whenever possible, while modifying only the hyperparameters of interest.

We use multilayer perceptron function approximators in all cases. We denote the hidden layer sizes and activations as $(N, M, \text{activation})$, where N and M are the sizes of the first and second hidden layers, respectively, and activation is the type of activation we use

[1] Specific details can be found in Appendix A.2 and code can be found at: `https://git.io/vFHnf`
[2] `https://www.github.com/openai/baselines`

between all layers – for example, a Rectified Linear Unit (ReLU) or Exponential Linear Unit (ELU). For default settings, we vary the hyperparameters under investigation one at a time. For DDPG we use a network structure of $(64, 64, \text{ReLU})$ for both actor and critic. For TRPO and PPO, we use $(64, 64, \tanh)$ for the policy. For ACKTR, we use $(64, 64, \tanh)$ for the actor and $(64, 64, \text{ELU})$ for the critic.

3.2.1 Hyperparameters

What is the magnitude of the effect hyperparameter settings can have on baseline performance?

Tuned hyperparameters play a large role in eliciting the best results from many algorithms. However, the choice of optimal hyperparameter configuration is often not consistent in related literature, and the range of values considered is often not reported[3]. Furthermore, poor hyperparameter selection can be detrimental to a fair comparison against baseline algorithms. Here, we investigate several aspects of hyperparameter selection on performance.

3.2.2 Network Architecture

How does the choice of network architecture for the policy and value function approximation affect performance?

Islam et al. (2017) show that policy network architecture can significantly impact results in both TRPO and DDPG. Furthermore, certain activation functions such as ReLU have been shown to cause worsened learning performance due to the "dying ReLU" problem (Xu et al., 2015). As such, we examine network architecture and activation functions for both policy and value function approximators. In the literature, similar lines of investigation have shown the differences in performance when comparing linear approximators, radial basis functions (RBFs), and neural networks (Rajeswaran et al., 2017). Tables 3.1 and 3.2 summarize the final evaluation performance of all architectural variations after training on 2M samples (i.e., 2M timesteps in the environment). All learning curves and details on setup can be found in Appendix A.1. We vary hyperparameters one at a time, while using

[3] A sampled literature review can be found in the Appendix A.1.

3.2 Experimental Analysis

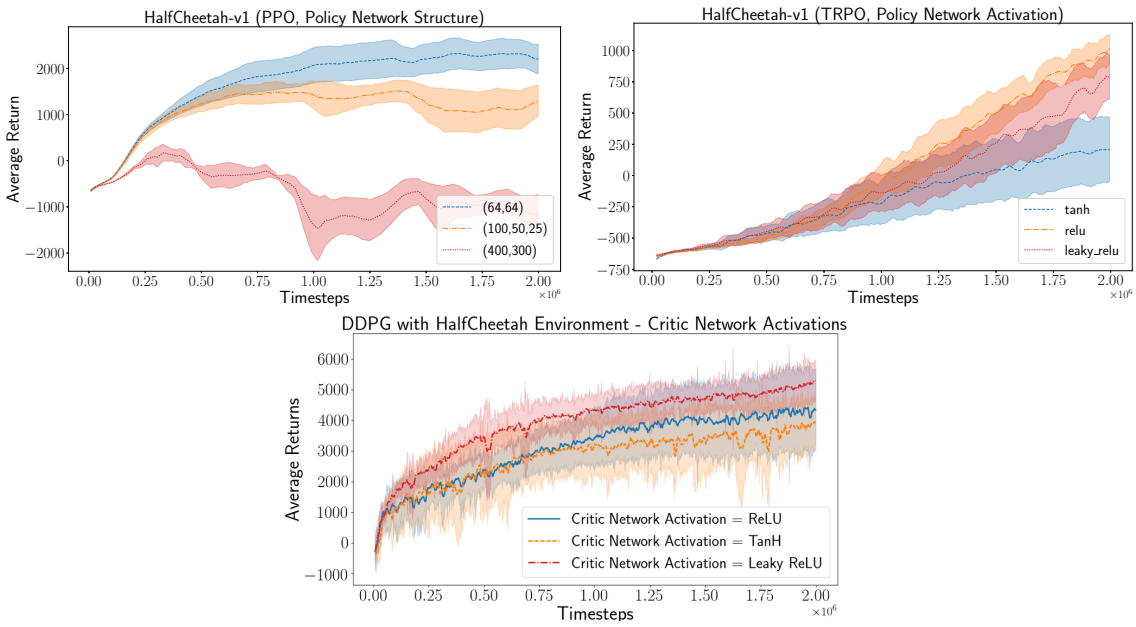

Figure 3.2: Significance of Policy Network Structure and Activation Functions PPO (top-left), TRPO (top-right) and DDPG (bottom).

a default setting for all others. We investigate three multilayer perceptron (MLP) architectures commonly seen in the literature: $(64, 64)$, $(100, 50, 25)$, and $(400, 300)$. Furthermore, we vary the activation functions of both the value and policy networks across tanh, ReLU, and Leaky ReLU activations.

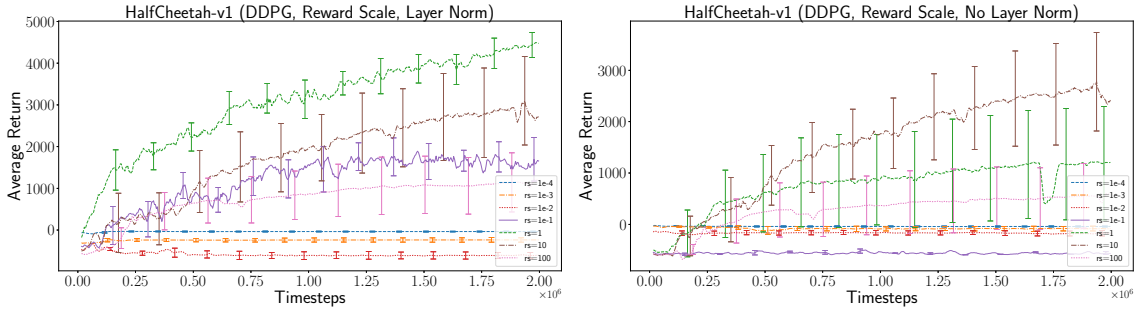

Figure 3.3: DDPG reward rescaling effects on HalfCheetah-v1, with and without layer norm.

Results Figure 3.2 shows how performance can be affected by simple changes to the policy or value network activations. We find that usually ReLU or Leaky ReLU activations

3.2 Experimental Analysis

Algorithm	Environment	400,300	64,64	100,50,25	tanh	ReLU	LeakyReLU
TRPO	Hopper-v1	2980 ± 35	2674 ± 227	3110 ± 78	2674 ± 227	2772 ± 211	-
(Schulman et al., 2015)	HalfCheetah-v1	1791 ± 224	1939 ± 140	2151 ± 27	1939 ± 140	3041 ± 161	-
TRPO	Hopper-v1	1243 ± 55	1303 ± 89	1243 ± 55	1303 ± 89	1131 ± 65	1341± 127
(Duan et al., 2016)	HalfCheetah-v1	738 ± 240	834 ± 317	850±378	834 ± 317	784 ± 352	1139 ±364
TRPO	Hopper-v1	2909 ± 87	2828 ± 70	2812 ± 88	2828 ± 70	2941 ± 91	2865 ± 189
(Schulman et al., 2017)	HalfCheetah-v1	-155 ± 188	205 ± 256	306 ± 261	205 ± 256	1045 ± 114	778 ± 177
PPO	Hopper-v1	61 ± 33	2790 ± 62	2592 ± 196	2790 ± 62	2695 ± 86	2587 ± 53
(Schulman et al., 2017)	HalfCheetah-v1	-1180 ± 444	2201 ± 323	1314 ± 340	2201 ± 323	2971 ± 364	2895 ± 365
DDPG	Hopper-v1	1419 ± 313	1632 ± 459	2142 ± 436	1491 ± 205	1632 ± 459	1384 ± 285
(Plappert et al., 2018)	HalfCheetah-v1	5579 ± 354	4198 ± 606	5600 ± 601	5325 ± 281	4198 ± 606	4094 ± 233
DDPG	Hopper-v1	600 ± 126	593 ± 155	501 ± 129	436 ± 48	593 ± 155	319 ± 127
(Gu et al., 2017)	HalfCheetah-v1	2845 ± 589	2771 ± 535	1638 ± 624	1638 ± 624	2771 ± 535	1405± 511
DDPG	Hopper-v1	506 ± 208	749 ± 271	629 ± 138	354 ± 91	749 ± 271	-
(Duan et al., 2016)	HalfCheetah-v1	850 ± 41	1573 ± 385	1224 ± 553	1311 ± 271	1573 ± 385	-
ACKTR	Hopper-v1	2577 ± 529	1608 ± 66	2287 ± 946	1608 ± 66	2835 ± 503	2718 ± 434
(Wu et al., 2017)	HalfCheetah-v1	2653 ± 408	2691 ± 231	2498 ± 112	2621 ± 381	2160 ± 151	2691 ± 231

Table 3.1: Results for our policy architecture permutations across various implementations and algorithms. Under a single variation, all other parameters left on default settings as discussed in Appendix A.1. Final average ± standard error across 5 trials of returns across the last 100 trajectories after 2M training samples. For ACKTR, we use **ELU** activations instead of leaky ReLU.

perform the best across environments and algorithms. The effects are not consistent across algorithms or environments. This inconsistency demonstrates how interconnected network architecture is to algorithm methodology. For example, using a large network with PPO may require tweaking other hyperparameters such as the trust region clipping or learning rate to compensate for the architectural change[4]. This intricate interplay of hyperparameters is one of the reasons reproducing current policy gradient methods is so difficult. It is exceedingly important to choose an appropriate architecture for proper baseline results. This also suggests a possible need for hyperparameter agnostic algorithms—that is algorithms that incorporate hyperparameter adaptation as part of the design—such that fair comparisons can be made without concern about improper settings for the task at hand.

3.2.3 Reward Scale

How can the reward scale affect results? Why is reward rescaling used?

[4]We find that the KL divergence of updates with the large network $(400, 300)$ seen in Figure 3.2 is on average 33.52 times higher than the KL divergence of updates with the $(64, 64)$ network.

3.2 Experimental Analysis

Algorithm	Environment	400,300	64,64	100,50,25	tanh	ReLU	LeakyReLU
TRPO	Hopper-v1	3011 ± 171	2674 ± 227	2782 ± 120	2674 ± 227	3104 ± 84	-
(Schulman et al., 2015)	HalfCheetah-v1	2355 ± 48	1939 ± 140	1673 ± 148	1939 ± 140	2281 ± 91	-
TRPO	Hopper-v1	2909 ± 87	2828 ± 70	2812 ± 88	2828 ± 70	2829 ± 76	3047 ± 68
(Schulman et al., 2017)	HalfCheetah-v1	178 ± 242	205 ± 256	172 ± 257	205 ± 256	235 ± 260	325 ± 208
PPO	Hopper-v1	2704 ± 37	2790 ± 62	2969 ± 111	2790 ± 62	2687 ± 144	2748 ± 77
(Schulman et al., 2017)	HalfCheetah-v1	1523 ± 297	2201 ± 323	1807 ± 309	2201 ± 323	1288 ± 12	1227 ± 462
DDPG	Hopper-v1	1419 ± 312	1632 ± 458	1569 ± 453	971 ± 137	852 ± 143	843 ± 160
(Plappert et al., 2018)	HalfCheetah-v1	5600 ± 601	4197 ± 606	4713 ± 374	3908 ± 293	4197 ± 606	5324 ± 280
DDPG	Hopper-v1	523 ± 248	343 ± 34	345 ± 44	436 ± 48	343 ± 34	-
(Gu et al., 2017)	HalfCheetah-v1	1373 ± 678	1717 ± 508	1868 ± 620	1128 ± 511	1717 ± 508	-
DDPG	Hopper-v1	1208 ± 423	394 ± 144	380 ± 65	354 ± 91	394 ± 144	-
(Duan et al., 2016)	HalfCheetah-v1	789 ± 91	1095 ± 139	988 ± 52	1311 ± 271	1095 ± 139	-
ACKTR	Hopper-v1	152 ± 47	1930 ± 185	1589 ± 225	691 ± 55	500 ± 379	1930 ± 185
(Wu et al., 2017)	HalfCheetah-v1	518 ± 632	3018 ± 386	2554 ± 219	2547 ± 172	3362 ± 682	3018 ± 38

Table 3.2: Results for our value function (Q or V) architecture permutations across various implementations and algorithms. Under a single variation, all other parameters left on default settings as discussed in Appendix A.1. Final average ± standard error across 5 trials of returns across the last 100 trajectories after 2M training samples. For ACKTR, we use **ELU** activations instead of leaky ReLU.

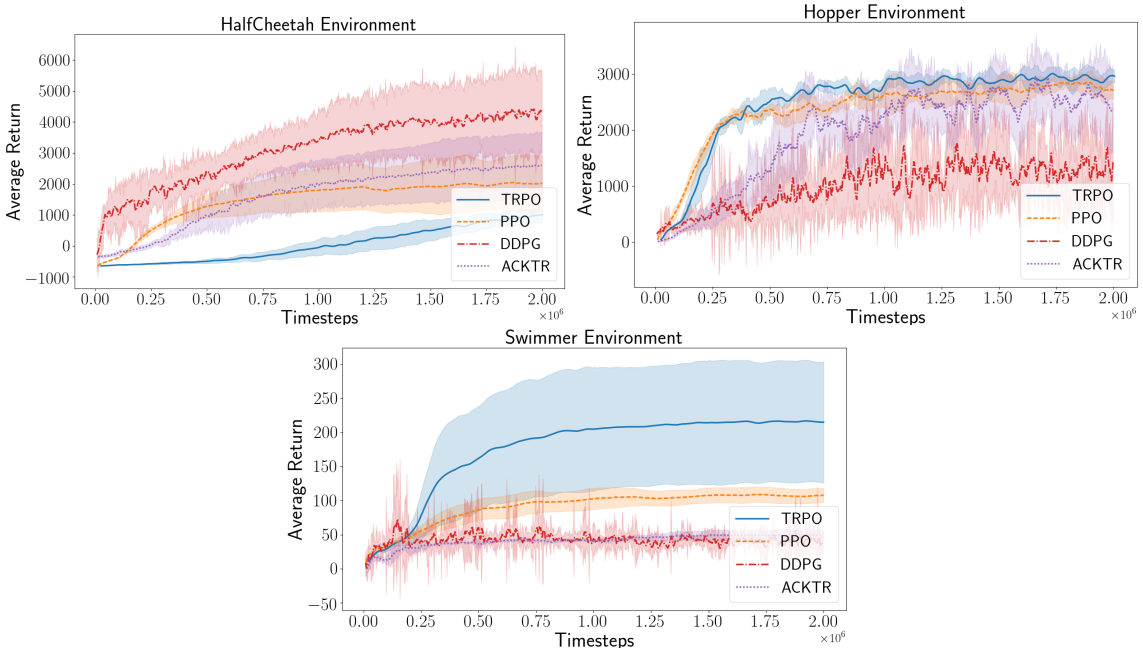

Figure 3.4: Performance of several policy gradient algorithms across benchmark MuJoCo environment suites

3.2 Experimental Analysis

Environment	DDPG	ACKTR	TRPO	PPO
HalfCheetah-v1	5037 (3664, 6574)	3888 (2288, 5131)	1254.5 (999, 1464)	3043 (1920, 4165)
Hopper-v1	1632 (607, 2370)	2546 (1875, 3217)	2965 (2854, 3076)	2715 (2589, 2847)
Walker2d-v1	1582 (901, 2174)	2285 (1246, 3235)	3072 (2957, 3183)	2926 (2514, 3361)
Swimmer-v1	31 (21, 46)	50 (42, 55)	214 (141, 287)	107 (101, 118)

Table 3.3: Bootstrap mean and 95% confidence bounds for a subset of environment experiments. We use $10,000$ bootstrap iterations and the pivotal method.

Reward rescaling has been used in several recent works (Duan et al., 2016; Gu et al., 2017) to improve results for DDPG. This involves simply multiplying the rewards generated from an environment by some scalar ($\hat{r} = r\sigma$) for training. Often, these works report using a reward scale of $\sigma = 0.1$. In Atari domains, this is akin to clipping the rewards to $[0, 1]$. By intuition, in gradient based methods (as used in most deep RL) a large and sparse output scale can result in problems regarding saturation and inefficiency in learning (LeCun et al., 2012; Glorot and Bengio, 2010; Vincent et al., 2015). Therefore clipping or rescaling rewards compresses the space of estimated expected returns in action value function based methods such as DDPG. We run a set of experiments using reward rescaling in DDPG (with and without layer normalization) for insights into how this aspect affects performance.

Results Our analysis shows that reward rescaling can have a large effect (full experiment results can be found in the Appendix), but results were inconsistent across environments and scaling values. Figure 3.3 shows one such example where reward rescaling affects results, causing a failure to learn in small settings below $\sigma = 0.01$. In particular, layer normalization changes how the rescaling factor affects results, suggesting that these impacts are due to the use of deep networks and gradient-based methods. With the value function approximator tracking a moving target distribution, this can potentially affect learning in unstable environments where a deep Q-value function approximator is used. Furthermore, some environments may have untuned reward scales (e.g., the HumanoidStandup-v1 of OpenAI gym which can reach rewards in the scale of millions). Therefore, we suggest that this hyperparameter has the potential to have a large impact if considered properly. Rather than rescaling rewards in some environments, a more principled approach should be taken to address this. An initial foray into this problem is made in (van Hasselt et al., 2016), where the authors adaptively rescale reward targets with normalized stochastic gradient, but further research is needed.

3.2.4 Random Seeds and Trials

Can random seeds drastically alter performance? Can one distort results by averaging an improper number of trials?

A major concern with deep RL is the variance in results due to environment stochasticity or stochasticity in the learning process (e.g., random weight initialization). As such, even averaging several learning results together across totally different random seeds can lead to the reporting of misleading results. We highlight this in the form of an experiment.

Results We perform 10 experiment trials, for the same hyperparameter configuration, only varying the random seed across all 10 trials. We then split the trials into two sets of five and average the five trials together. As shown in Figure 3.5, we find that the performance of algorithms can be drastically different. We demonstrate that the variance between runs is enough to create statistically different distributions just from varying random seeds. Unfortunately, in recent reported results, it is not uncommon for the top-N trials to be selected from among several trials (Wu et al., 2017; Mnih et al., 2016) or averaged over only small number of trials ($N < 5$) (Gu et al., 2017; Wu et al., 2017). Our experiment with random seeds shows that this can be potentially misleading. Particularly for HalfCheetah, it is possible to get learning curves that do not fall within the same distribution at all, just by averaging different runs with the same hyperparameters, but different random seeds. While there can be no specific number of trials specified as a recommendation, it is possible that power analysis methods can be used to give a general idea to this extent as we will discuss later. However, more investigation is needed to answer this open problem.

3.2.5 Environments

How do the environment properties affect variability in reported RL algorithm performance?

To assess how the choice of evaluation environment can affect the presented results, we use our aforementioned default set of hyperparameters across our chosen testbed of algorithms and investigate how well each algorithm performs across an extended suite of continuous control tasks. For these experiments, we use the following environments from OpenAI

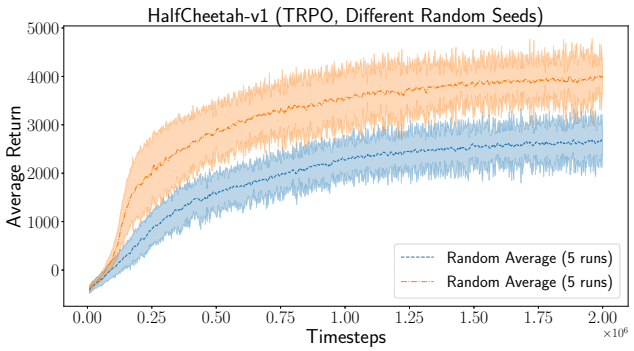

Figure 3.5: TRPO on HalfCheetah-v1 using the same hyperparameter configurations averaged over two sets of 5 different random seeds each. The average 2-sample t-test across entire training distribution resulted in $t = -9.0916$, $p = 0.0016$.

Gym: Hopper-v1, HalfCheetah-v1, Swimmer-v1 and Walker2d-v1. The choice of environment often plays an important role in demonstrating how well a new proposed algorithm performs against baselines. In continuous control tasks, often the environments have random stochasticity, shortened trajectories, or different dynamic properties. We demonstrate that, as a result of these differences, algorithm performance can vary across environments and the best performing algorithm across all environments is not always clear. Thus it is increasingly important to present results for a wide range of environments and not only pick those which show a novel work outperforming other methods.

Results As shown in Figure 3.4, in environments with stable dynamics (e.g., HalfCheetah-v1), DDPG outperforms all other algorithsm. However, as dynamics become more unstable (e.g., in Hopper-v1) performance gains rapidly diminish. As DDPG is an off-policy method, exploration noise can cause sudden failures in unstable environments. Therefore, learning a proper Q-value estimation of expected returns is difficult, particularly since many exploratory paths will result in failure. Since failures in such tasks are characterized by shortened trajectories, a local optimum in this case would be simply to survive until the maximum length of the trajectory (corresponding to one thousand timesteps and similar reward due to a survival bonus in the case of Hopper-v1). As can be seen in Figure 3.4, DDPG with Hopper does exactly this. This is a clear example where showing only the favourable and stable HalfCheetah when reporting DDPG-based experiments would be unfair.

Furthermore, let us consider the Swimmer-v1 environment shown in Figure 3.4. Here, TRPO significantly outperforms all other algorithms. Due to the dynamics of the water-like environment, a local optimum for the system is to curl up and flail without proper swimming. However, this corresponds to a return of $\sim$130. By reaching a local optimum, learning curves can indicate successful optimization of the policy over time, when in reality the returns achieved are not qualitatively representative of learning the desired behaviour, as demonstrated in video replays of the learned policy[5]. Therefore, it is important to show not only returns but demonstrations of the learned policy in action. Without understanding what the evaluation returns indicate, it is possible that misleading results can be reported which in reality only optimize local optima rather than reaching the desired behaviour.

3.2.6 Codebases

Are commonly used baseline implementations comparable?

In many cases, authors implement their own versions of baseline algorithms to compare against. We investigate the OpenAI baselines implementation of TRPO as used in (Schulman et al., 2017), the original TRPO code (Schulman et al., 2015), and the rllab (Duan et al., 2016) Tensorflow implementation of TRPO. We also compare the rllab Theano (Duan et al., 2016), rllab++ (Gu et al., 2017), and OpenAI baselines (Plappert et al., 2018) implementations of DDPG. Our goal is to draw attention to the variance due to implementation details across algorithms. We run a subset of our architecture experiments as with the OpenAI baselines implementations using the same hyperparameters as in those experiments[6].

Results We find that implementation differences which are often not reflected in publications can have dramatic impacts on performance. This can be seen for our final evaluation performance after training on 2M samples in Tables 3.1 and 3.2, as well as a sample comparison in Figure 3.6. This demonstrates the necessity that implementation details be enumerated, codebases packaged with publications, and that performance of baseline experiments in novel works matches the original baseline publication code.

[5] https://youtu.be/lKpUQYjgm80

[6] Differences are discussed in the Appendix (e.g., use of different optimizers for the value function baseline). Leaky ReLU activations are left out to narrow the experiment scope.

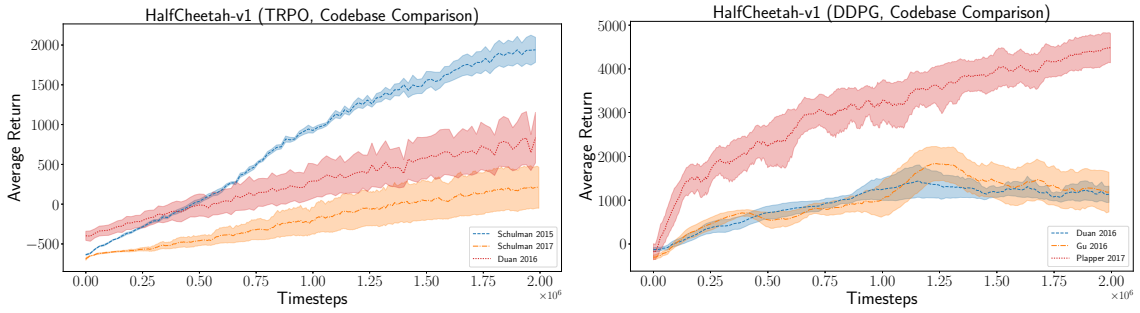

Figure 3.6: TRPO and DDPG codebase comparisons using our default set of hyperparameters (as used in other experiments).

3.3 Reporting Evaluation Metrics

In this section we analyze some of the evaluation metrics commonly used in the reinforcement learning literature. In practice, RL algorithms are often evaluated by simply presenting plots or tables of average cumulative reward (average returns) and, more recently, of maximum reward achieved over a fixed number of timesteps. Due to the unstable nature of many of these algorithms, simply reporting the maximum returns is typically inadequate for fair comparison; even reporting average returns can be misleading as the range of performance across seeds and trials is unknown. Alone, these may not provide a clear picture of an algorithm's range of performance. However, when combined with confidence intervals, this may be adequate to make an informed decision given a large enough number of trials. As such, we investigate using the bootstrap and significance testing as in ML (Kohavi, 1995; Bouckaert and Frank, 2004; Nadeau and Bengio, 2000) to evaluate algorithm performance.

Online View versus Policy Optimization An important distinction when reporting results is the online learning view versus the policy optimization view of RL. In the online view, an agent will optimize the returns across the entire learning process and there is not necessarily an end to the agent's trajectory. In this view, evaluations can use the average cumulative rewards across the entire learning process (balancing exploration and exploitation) as in (Hofer and Gimbert, 2016), or can possibly use offline evaluation as in (Mandel et al., 2016). The alternate view corresponds to policy optimization, where evaluation is

performed using a target policy in an offline manner. In the policy optimization view it is important to run evaluations across the entire length of the task trajectory with a single target policy to determine the average returns that the target can obtain. We focus on evaluation methods for the policy optimization view (with offline evaluation), but the same principles can be applied to the online view.

Confidence Bounds The sample bootstrap has been a popular method to gain insight into a population distribution from a smaller sample (Efron and Tibshirani, 1994). Bootstrap methods are particularly popular for A/B testing, and we can borrow some ideas from this field. Generally a bootstrap estimator is obtained by resampling with replacement many times to generate a statistically relevant mean and confidence bound. Using this technique, we can gain insight into what is the 95% confidence interval of the results from our section on environments. Table 3.3 shows the bootstrap mean and 95% confidence bounds on our environment experiments. Confidence intervals can vary wildly between algorithms and environments. We find that TRPO and PPO are the most stable with small confidence bounds from the bootstrap. In cases where confidence bounds are exceedingly large, it may be necessary to run more trials (i.e., increase the sample size).

Power Analysis Another method to determine if the sample size must be increased is bootstrap power analysis (Tufféry, 2011; Yuan and Hayashi, 2003). If we use our sample and give it some uniform lift (for example, scaling uniformly by 1.25), we can run many bootstrap simulations and determine what percentage of the simulations result in statistically significant values with the lift. If there is a small percentage of significant values, a larger sample size is needed (more trials must be run). We do this across all environment experiment trial runs and indeed find that, in more unstable settings, the bootstrap power percentage leans towards insignificant results in the lift experiment. Conversely, in stable trials (e.g., TRPO on Hopper-v1) with a small sample size, the lift experiment shows that no more trials are needed to generate significant comparisons. These results are provided in the Appendix A.8.

Significance An important factor when deciding on an RL algorithm to use is the significance of the reported gains based on a given metric. Several works have investigated the use of significance metrics to assess the reliability of reported evaluation metrics in ML. However, few works in reinforcement learning assess the significance of reported

metrics. Based on our experimental results which indicate that algorithm performance can vary wildly based simply on perturbations of random seeds, it is clear that some metric is necessary for assessing the significance of algorithm performance gains and the confidence of reported metrics. While more research and investigation is needed to determine the best metrics for assessing RL algorithms, we investigate an initial set of metrics based on results from ML.

In supervised learning, k-fold t-test, corrected resampled t-test, and other significance metrics have been discussed when comparing machine learning results (Bouckaert and Frank, 2004; Nadeau and Bengio, 2000). However, the assumptions pertaining to the underlying data with corrected metrics do not necessarily apply in RL. Further work is needed to investigate proper corrected significance tests for RL. Nonetheless, we explore several significance measures which give insight into whether a novel algorithm is truly performing as the state-of-the-art. We consider the simple 2-sample t-test (sorting all final evaluation returns across N random trials with different random seeds); the Kolmogorov-Smirnov test (Wilcox, 2005); and bootstrap percent differences with 95% confidence intervals. All calculated metrics can be found in Appendix A.8. Generally, we find that the significance values match up to what is to be expected. Take, for example, comparing Walker2d-v1 performance of ACKTR versus DDPG. ACKTR performs slightly better, but this performance is not significant due to the overlapping confidence intervals of the two: $t = 1.03, p = 0.334$, $KS = 0.40, p = 0.697$, bootstrapped percent difference 44.47% (-80.62%, 111.72%).

3.4 Discussion

Through experimental methods focusing on PG methods for continuous control, we investigate problems with reproducibility in deep RL. We find that both intrinsic (e.g., random seeds, environment properties) and extrinsic sources (e.g., hyperparameters, codebases) of non-determinism can contribute to difficulties in reproducing baseline algorithms. Moreover, we find that highly varied results due to intrinsic sources bolster the need for using proper significance analysis. We propose several such methods and show their value on a subset of our experiments.

3.4 Discussion

What recommendations can we draw from our experiments?

Based on our experimental results and investigations, we can provide some general recommendations. Hyperparameters can have significantly different effects across algorithms and environments. Thus it is important to find the working set which at least matches the original reported performance of baseline algorithms through standard hyperparameter searches. Similarly, new baseline algorithm implementations used for comparison should match the original codebase results if available. Overall, due to the high variance across trials and random seeds of reinforcement learning algorithms, many trials must be run with different random seeds when comparing performance. Unless random seed selection is explicitly part of the algorithm, averaging multiple runs over different random seeds gives insight into the population distribution of the algorithm performance on an environment. Similarly, due to these effects, it is important to perform proper significance testing to determine if the higher average returns are in fact representative of better performance.

We highlight several forms of significance testing and find that they give generally expected results when taking confidence intervals into consideration. Furthermore, we demonstrate that bootstrapping and power analysis are possible ways to gain insight into the number of trial runs necessary to make an informed decision about the significance of algorithm performance gains. In general, however, the most important step to reproducibility is to report *all* hyperparameters, implementation details, experimental setup, and evaluation methods for both baseline comparison methods and novel work. Without the publication of implementations and related details, wasted effort on reproducing state-of-the-art works will plague the community and slow down progress.

What are possible future lines of investigation?

Due to the significant effects of hyperparameters (particularly reward scaling), another possibly important line of future investigation is in building hyperparameter agnostic algorithms. Such an approach would ensure that there is no unfairness introduced from external sources when comparing algorithms agnostic to parameters such as reward scale, batch size, or network structure. Furthermore, while we investigate an initial set of significance metrics here, they may not be the best fit for comparing RL algorithms. Several works have begun investigating policy evaluation methods for the purposes of safe RL (Thomas

and Brunskill, 2016; Thomas et al., 2015), but further work is needed in significance testing and statistical analysis. Similar lines of investigation to (Nadeau and Bengio, 2000; Bouckaert and Frank, 2004) would be helpful to determine the best methods for evaluating performance gain significance.

How can we ensure that deep RL matters?

We discuss many different factors affecting reproducibility of RL algorithms. The sensitivity of these algorithms to changes in reward scale, environment dynamics, and random seeds can be considerable and varies between algorithms and settings. Since benchmark environments are proxies for real-world applications to gauge generalized algorithm performance, perhaps more emphasis should be placed on the applicability of RL algorithms to real-world tasks. That is, as there is often no clear winner among all benchmark environments, perhaps recommended areas of application should be demonstrated along with benchmark environment results when presenting a new algorithm. Maybe new methods should be answering the question: in what setting would this work be useful? This is something that is addressed for machine learning in (Wagstaff, 2012) and may warrant more discussion for RL. As a community, we must not only ensure reproducible results with fair comparisons, but we must also consider what are the best ways to demonstrate that RL continues to matter.

4

Benchmark Environments for Multitask Learning in Continuous Domains

As we discuss in Section 2.2, one general definition seen in literature of multitask learning involves training a single agent in a lifelong context across a series of tasks. This definition encompasses three research areas: lifelong learning, multitask learning, and transfer learning. In these settings, when tasks share similar characteristics, there is potential for the learning agent to reuse information, potentially achieving greater performance or learning more rapidly on down-stream tasks than would be possible by learning each task from scratch.

The promise of multitask learning has been previously demonstrated in several contexts. It has been shown that transfer learning on multiple related tasks improves the ability of the agent to learn a larger variety of domains while using less training data overall (Thrun, 1996). This naturally renders the agent more applicable to real-world scenarios. Additionally, by training on multiple tasks the agent can exploit common traits to gain efficiency and improve generalization to unseen settings (Caruana, 1998; Murugesan et al., 2016; Finn et al., 2017).

In discrete domains, several works have investigated the transferring of playing knowledge acquired between various Atari games. It is intuitive that there should be some knowledge transfer between Atari games (Breakout and Pong are similar in catching a ball; DemonAttack, Carnaval, Assault, and AirRaid share a goal in shooting upwards to destroy

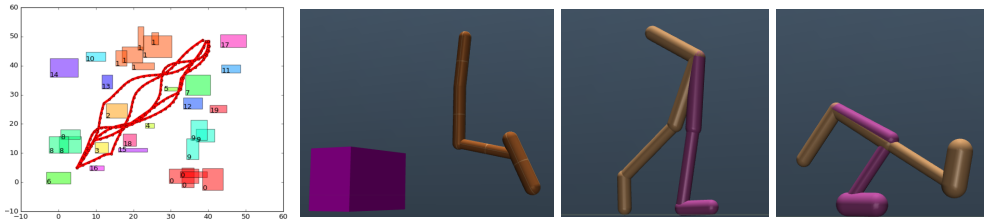

Figure 4.1: Example environments: 2D navigation task with several sample paths, Hopper with a wall, Walker2d with big and small feet, from left to right respectively.

enemies). Several works demonstrate multitask learning in both Atari tasks and the DeepMind Lab Labyrinth (Parisotto et al., 2015; Rusu et al., 2016; Jaderberg et al., 2017). These environments stand as useful benchmarks which can be used to induce further progress in multitask, transfer and lifelong learning in discrete domains.

On the contrary, in continuous domains, there are no de facto benchmark environments. Many novel works for continuous domain multitask learning have utilized a unique set of environments with little mutual overlap. As such, there is a need for such open-source standard benchmarks in this setting. This problem is one that has been recognized by other research groups. Particularly, OpenAI lists the need for benchmark environments and investigation of multitask learning in continuous domains in a request for research[1]. As they mention, the current OpenAI Gym (Brockman et al., 2016) environments do not share enough characteristics to likely pose as effective multitask or lifelong learning benchmarks. The contribution of this work is a set of benchmark environments that are suitable to evaluate continuous domain multitask learning. Our environments are constructed using an expandable software framework built on top of OpenAI Gym.

Here, we show over 50 new environment variations (spread among 12 broad groups of variation types) for challenging new continuous domain tasks. We verify the utility of these environments for evaluating multitask learning by reporting the performance of a well-known reinforcement learning algorithm on our multitask benchmark environments as a simple baseline.

[1] See: https://openai.com/requests-for-research/#multitask-rl-with-continuous-actions

4.1 Environments

In our initial release of the *gym-extensions* framework[2], we include a number of modifications of the standard gym environments as well as novel continuous domains, and provide a framework which allows easy modification of environment characteristics. When choosing how to modify environments or construct new multitask learning tasks, we look to the existing literature described in the previous section. We choose common modifications as a starting point. Furthermore, we look to the description of lifelong learning in (Thrun, 1995) for inspiration of the desiderata: "Such scenarios provide the opportunity for synergetic effects that arise if knowledge is transferred across multiple learning tasks." Hence, we choose modifications where significant portions of multiple environments can be reused. For example, we add a wall in one modification such that a gait from a non-wall environment can be reused for a portion of the walled task.

4.1.1 Continuous Control in MuJoCo

We base our modified environments on the existing "running" (Humanoid, Hopper, HalfCheetah, and Walker2d) and "arm-based" (Pusher and Striker) environments in OpenAI Gym. First, we provide a high-level overview of our modifications and suggested grouping, then we show the specific environment names in our benchmarking results.

Gravity Modifications

For the running agents, we provide ready environments with various scales of simulated earth-like gravity, ranging from one half to one and a half of the normal gravity level (-4.91 to $-12.26\, m \cdot s^{-2}$ in increments of $.25 g_{earth}$). We propose that a successful multitask learning algorithm will extract the underlying walking action structure and reuse the applicable knowledge without forgetting how to walk in varying gravity conditions.

[2]Found at: https://github.com/Breakend/gym-extensions/. Pull requests and issues are welcome. More details for each environment will be provided in the open-source repository as well as a place to upload new benchmark algorithm runs.

4.1 Environments

Wall and Sensor Environments

Inspired by the wall jumping experiment in (Finn et al., 2017), we build a set of similar environments by extending the OpenAI running tasks to use a multi-beam noiseless range sensor. We emit ray-beams from the torso of the runner for the measurements (with an arc of 90 degrees, 10 beams, a maximum sensing distance of 10 meters, and readouts normalized to a range of $[0, 1]$). We provide the usual running tasks with the sensor perception enabled (with no readings since there is no wall), and extra environments with a wall set in the path of the agent at a location drawn from a uniform distribution from 1.8 to 3.8 meters ahead of the agent's start location.

Morphology Modifications

For the running agents, we provide environments which vary the morphology of a specific body part of the agent. The modifications made to each agent are seen in Table 4.2. We define "Big" bodyparts as scaling the mass and width of the limb by 1.25 and "Small" bodyparts as being scaled by 0.75. We also group categories of limbs for environments with multiple appendages (i.e., humanoid torso includes the abdomen; humanoid thigh also includes the hips; all appendages encompass both the left/right or front/back simultaneously such that a modified thigh includes both thighs).

Robot Arm Modifications

In the OpenAI Striker and Pusher tasks, a 7 DoF arm tries to hit a ball into a hole or push a peg to a goal position respectively. We extend these tasks to randomly move the goal position for the Pusher task, and randomly move the ball start position for the Striker task. As in the original tasks, we bound the varied goal or start state within a reasonable area that the arm can reach. The exact bounding areas can be found in the accompanying code.[3]

Humanoid Multitask

We provide a humanoid multitask environment which combines the rewards for standing up and running in the same environment. The reward scale for this task is rather large, but

[3] https://github.com/Breakend/gym-extensions/

aligns with the HumanoidStandup-v1 environment from OpenAI Gym. Additionally we provide a version of each environment with a sensor readout as in Section 4.1.1. When no wall is used, all sensors read zero. When a wall is used, each returns a distance to the wall as previously described.

4.1.2 2D Navigation

We also provide several novel 2D environments that focus on navigation tasks with continuous action spaces to enable benchmarking of learning tasks requiring an implicit memory. The tasks take place in a given occupancy grid map, similar to (Tamar et al., 2016). We opt to make the layout and shape of the obstacles as the only disambiguating feature for localizing within the map. Aside from that information, the environment does not have any texture mapping or other distinctive features.

We provide three different types of navigation tasks, increasing in level of difficulty:

- Image-based navigation where the agent has access to the entire map, including its own position within the map and the destination in the map as part of the image data.
- State-based navigation, where the agent has access to its own position in the map and the distance and bearing to the closest obstacle. A simpler version also contains the destination coordinates.
- Navigation based only on local range-and-bearing data around the agent using ray-tracing. It has to perform mapping and estimate its own position within the map, while at the same time exploring to find the goal location, and learning to avoid obstacles. We also modify this with a simpler version, where the goal and current position are known as well.

We provide a reward of -1 for every timestep, -5 for obstacle collisions, and +10 for reaching the goal (which also ends the task, similarly to the MountainCar-v0 environment in OpenAI Gym). The action space is the bounded velocity to apply in the x and y directions with a maximum velocity of 5 m/s and map dimensions $640m \times 480m$ (where meters are simulated in this case and correspond with pixels).

4.2 Multitask Sets

We develop several sets of intuitive task groups which can serve as simple benchmarks which increase in complexity both within the group and in our listing order. We select these task groupings once again according to the criteria that the "scenarios provide the opportunity for synergetic effects that arise if knowledge is transferred across multiple learning tasks" (Thrun, 1995). More specifically, we suggest groupings that can have significant overlap in task goals, but significant differences in dynamics or settings. The goal is to cause failure when using an overfit policy in two different environments within a single group, but to be able to learn both tasks more efficiently given information reuse. The specific environment names can be found in Table 4.1, 4.2, 4.3, and 4.4. For the navigation tasks, we list the environments inline here.

We introduce the following environment groups:

- Modified environments with different gravity parameters
- Modified environments with sensor readouts (simply reading zero if no wall) and permuted with a random wall in the runner path
- The OpenAI Gym Striker environment with both random start position of the object as well as random goal state
- The OpenAI Gym Pusher environment with both random start position of the object as well as random goal state
- Learning to standup and run for a Humanoid model
- Learning to standup, run, and jump over walls for a Humanoid model
- Learning to run with different sized limbs with the base set of limbs encompassing {Torso, Leg, Thigh, Foot} and specific extra limbs listed below (i.e., example combinations look like: HumanoidBigArm-v0, HopperSmallFoot-v0).
- Learning to navigate and search in 2D environments using only current position and distance to closest obstacles (State-Based-Navigation-2d-Map{0-9}-Goal{0-2}-v0)
- Learning to navigate and search in 2D environments observing current position, distance to closest obstacles, and known goal position (State-Based-Navigation-2d-Map{0-9}-Goal{0-2}-KnownGoalPosition-v0)

- Learning to navigate and search in 2D environments observing only raytracing distance readouts (Limited-Range-Based-Navigation-2d-Map{0-9}-Goal{0-2}-v0)
- Learning to navigate and search in 2D environments observing current position, raytracing distance readouts, and known goal position (Limited-Range-Based-Navigation-2d-Map{0-9}-Goal{0-2}-KnownPositions-v0)
- Learning to navigate and search in 2D environments observing only the 2D map image with goal location and current position highlighted in different colors (Image-Based-Navigation-2d-Map{0-9}-Goal{0-2}-v0)

4.3 Baseline Experiments

We develop a basic experiment to run on the aforementioned groupings of the environments to demonstrate learning on a series of similar tasks consecutively. We then evaluate the generalized performance across all of the environments using the final learned policy. For an initial baseline, we simply run the rllab (Duan et al., 2016) implementation of Trust Region Policy Optimization (Schulman et al., 2015) (TRPO) using an identical policy network to (Gu et al., 2017)[4]. We train the same policy consecutively on each environment in a group in the same order as listed in Tables 4.1-4.4. After having trained on a specific environment, we evaluate the current policy on that environment by running 20 sample rollouts. We then train on the next environment in the group, starting from that same policy. We finally evaluate the reward across 20 sample rollouts on each environment in a group using the final learned policy (which by then has been trained on every variation of the environment). While this is not a specialized multitask learning approach as in (Yang et al., 2017; Mujika, 2016; Teh et al., 2017), this provides basic insights (using a well-known reinforcement learning algorithm) into forward transfer and generalization of a policy on these task groupings.

Our baseline experiment results are found in Tables 4.1, 4.2, 4.3, 4.4. In the case of modified Hopper tasks, modifying gravity and body part size has a profound effect on the system dynamics. As a result, we see that catastrophic forgetting (McCloskey and Cohen,

[4]Size 100, 50, 25 hidden layers with rectified linear activations and a tanh output activation, and hyperparameters: step-size, 0.01; GAE lambda, 1.0; regularization coefficient, $1.0 \cdot 10^{-5}$; number of epochs, 1000; batch size, 50000

4.3 Baseline Experiments

Environment	Fully Trained	After Env Training	First Step	Single Env
HopperGravityHalf-v0	1495.93 ± 823.51	2352.19 ± 580.53	13.48 ± 8.71	1843.89 ± 485.25
HopperGravityThreeQuarters-v0	413.77 ± 252.67	2245.13 ± 872.16	697.96 ± 210.79	2328.09 ± 834.35
Hopper-v1	668.52 ± 159.90	2622.31 ± 1032.45	781.88 ± 262.35	3232.87 ± 582.55
HopperGravityOneAndQuarter-v0	922.76 ± 128.71	3006.17 ± 847.30	818.08 ± 255.85	3028.04 ± 875.81
HopperGravityOneAndHalf-v0	2690.57 ± 1110.39	2792.72 ± 1075.30	658.15 ± 117.14	2169.07 ± 825.75
Average for Grouping	990.95 ± 1022.32	2603.704 ± 881.54	593.91 ± 184.43	2520.39 ± 720.74
Walker2dGravityHalf-v0	1366.07 ± 1126.59	3485.19 ± 1054.06	5.35 ± 10.30	2231.86 ± 902.31
Walker2dGravityThreeQuarters-v0	3686.37 ± 287.96	3962.69 ± 1061.71	1071.95 ± 267.35	2431.87 ± 935.14
Walker2d-v1	4030.00 ± 85.76	3732.04 ± 1314.89	930.92 ± 264.88	2570.15 ± 915.58
Walker2dGravityOneAndQuarter-v0	4115.23 ± 90.33	4090.30 ± 1058.62	926.06 ± 303.76	3505.52 ± 1626.58
Walker2dGravityOneAndHalf-v0	4201.08 ± 684.37	3988.62 ± 971.43	925.93 ± 290.33	2435.21 ± 1391.00
Average for Grouping	3479.76 ± 1230.72	3851.768 ± 1092.1	772.04 ± 227.32	2634.92 ± 1154.12
HalfCheetahGravityHalf-v0	1495.93 ± 823.51	1107.50 ± 784.31	-369.31 ± 113.71	2048.93 ± 761.03
HalfCheetahGravityThreeQuarters-v0	1671.76 ± 594.15	2142.78 ± 818.99	1410.25 ± 529.41	3268.26 ± 703.43
HalfCheetah-v1	1743.97 ± 100.32	2410.50 ± 137.30	1867.99 ± 251.58	2554.01 ± 115.69
HalfCheetahGravityOneAndQuarter-v0	2649.13 ± 143.43	2939.14 ± 164.62	1966.66± 171.88	2572.64 ± 90.80
HalfCheetahGravityOneAndHalf-v0	3421.21 ± 165.60	3402.83 ± 204.00	2143.76 ± 236.60	2276.82 ± 93.30
Average for Grouping	2196.41 ± 867.75	2400.55 ± 421.84	1403.87 ± 260.63	2544.13 ± 352.85
HumanoidGravityHalf-v0	416.41 ± 76.41	421.12 ± 95.61	89.60 ± 10.92	849.29 ± 213.81
HumanoidGravityThreeQuarters-v0	356.74 ± 52.52	385.54 ± 72.98	293.14 ± 66.12	637.33 ± 170.51
Humanoid-v1	310.09 ± 55.31	326.59 ± 59.78	267.11 ± 52.74	483.35 ± 106.12
HumanoidGravityOneAndQuarter-v0	261.01 ± 31.75	269.03 ± 40.59	233.82 ± 39.41	576.98 ± 124.25
HumanoidGravityOneAndHalf-v0	227.17 ± 33.62	226.94 ± 29.09	208.74 ± 34.43	538.24 ± 113.17
Average for Grouping	314.28 ± 85.41	325.84 ± 59.61	218.48 ± 40.72	617.03 ± 145.57

Table 4.1: Average and standard deviation ($\mu \pm \sigma$) of reward across a set of 20 sample rollouts. We show samples immediately after training on a particular environment and the reward obtained by the final trained policy on all previously seen environments. A group of tasks is defined by a bold separator and the total average across all final rollouts is presented. "Fully Trained" lists the final evaluation result using the fully trained policy which has seen all the environments. "After Env Training" lists the evaluation immediately after training on that specific environment (having seen all the previous environments up until that point in the group). The "First Step" column indicates the reward at the first iteration of training on the new environment after having trained on the previous environments in the group. "Single Env" indicates rollouts on a policy trained solely on that environment (with all the same training parameters).

4.3 Baseline Experiments

Environment	Fully Trained	After Env Training	First Step
HopperSmallFoot-v0	591.91 ± 150.73	1330.65 ± 402.07	9.83 ± 4.52
HopperSmallLeg-v0	2074.58 ± 800.61	1359.85 ± 311.91	744.35 ±120.50
HopperSmallThigh-v0	919.87 ± 343.57	1492.44 ± 486.72	1719.34 ±757.42
HopperSmallTorso-v0	1094.85 ± 319.94	1492.97 ± 518.47	1636.30 ±298.22
HopperBigFoot-v0	2823.58 ± 887.25	1819.91 ± 812.33	559.61 ±145.24
HopperBigLeg-v0	1020.13 ± 454.74	2148.57 ± 795.95	689.58 ±96.23
HopperBigThigh-v0	2799.39 ± 748.89	1827.48 ± 767.09	674.14 ± 101.72
HopperBigTorso-v0	1971.50 ± 794.24	2090.68 ± 693.34	1110.46 ± 213.74
Average for Grouping	1661.98 ± 1025.19	1695.31 ± 598.48	892.95 ± 203.69
Walker2dSmallFoot-v0	2497.10 ± 1309.80	531.08 ± 329.00	-2.87 ± 2.56
Walker2dSmallLeg-v0	3181.14 ± 1131.29	1120.19 ± 597.09	318.20 ± 229.00
Walker2dSmallThigh-v0	3106.65 ± 641.34	1735.39 ± 880.44	1317.72 ± 737.66
Walker2dSmallTorso-v0	3132.88 ± 991.48	1838.79 ± 965.60	979.32 ± 582.14
Walker2dBigFoot-v0	2751.34 ± 1216.07	1873.60 ± 1047.41	789.32 ± 289.65
Walker2dBigLeg-v0	2820.94 ± 1108.26	2133.64 ± 1246.53	1743.53 ± 1106.90
Walker2dBigThigh-v0	892.54 ± 212.46	2756.79 ± 1238.62	805.31 ±147.40
Walker2dBigTorso-v0	3097.45 ± 1383.43	2701.94 ± 1473.47	3045.06 ± 967.83
Average for Grouping	2685.01 ± 1280.70	1836.42 ± 972.27	1124.45 ± 507.89
HalfCheetahSmallFoot-v0	2003.46 ± 933.59	898.51 ± 363.85	-502.264 ± 97.99
HalfCheetahSmallLeg-v0	2327.16 ± 702.69	1494.03 ± 310.11	904.82 ± 330.37
HalfCheetahSmallThigh-v0	2555.16 ± 96.80	1672.22 ± 110.11	1311.60 ± 281.24
HalfCheetahSmallTorso-v0	2294.72 ± 109.20	1845.20 ± 86.03	1515.53 ± 94.68
HalfCheetahBigFoot-v0	2211.92 ± 65.81	1997.73 ± 101.36	1789.90 ± 82.31
HalfCheetahBigLeg-v0	2269.78 ± 95.57	2101.74 ± 95.98	1908.53 ± 99.49
HalfCheetahBigThigh-v0	2424.95 ± 94.19	2345.88 ± 381.33	1925.04 ± 347.83
HalfCheetahBigTorso-v0	2686.13 ± 97.96	2620.46 ± 297.88	2456.04 ± 421.03
Average for Grouping	2346.66 ± 464.81	1871.97 ± 218.33	1413.64 ± 207.11
HumanoidSmallFoot-v0	391.70 ± 124.75	228.46 ± 62.80	94.85 ± 106.57
HumanoidSmallLeg-v0	438.90 ± 113.80	290.88 ± 82.86	253.81 ± 68.12
HumanoidSmallThigh-v0	378.47 ± 113.70	347.38 ± 99.89	322.97 ± 93.28
HumanoidSmallTorso-v0	433.04 ± 88.76	341.22 ± 89.69	313.71 ± 82.52
HumanoidBigFoot-v0	456.39 ± 85.60	399.96 ± 95.84	355.16 ± 92.07
HumanoidBigLeg-v0	430.82 ± 105.41	380.20 ± 97.58	347.26 ± 87.78
HumanoidBigThigh-v0	365.79 ± 72.06	331.80 ± 84.06	303.13 ± 77.17
HumanoidBigTorso-v0	397.91 ± 109.04	392.66 ± 108.20	374.94 ± 102.12
HumanoidSmallHead-v0	422.33 ± 112.20	395.75 ± 101.70	386.14 ± 96.37
HumanoidBigHead-v0	507.29 ± 146.50	409.66 ± 109.13	411.98 ± 119.99
HumanoidSmallArm-v0	429.93 ± 113.26	416.41 ± 94.45	400.16 ± 91.75
HumanoidBigArm-v0	466.13 ± 129.87	411.23 ± 111.20	392.53 ± 115.21
HumanoidSmallHand-v0	450.07 ± 76.72	423.29 ± 101.61	417.45 ± 99.08
HumanoidBigHand-v0	409.46 ± 69.38	420.65 ± 100.29	415.04 ± 108.38
Average for Grouping	427.02 ± 112.56	370.68 ± 95.66	342.08 ± 95.23

Table 4.2: Results for modified body-part running task groups. Same parameters as described in Table 4.1. Number of training iterations lowered to 500 per environment due to the larger number of environments. Environments are grouped together by motif and trained in order as listed here.

4.3 Baseline Experiments

Environment	Fully Trained	After Env Training	First Step
HopperWithSensor-v0	747.67 ± 27.06	2881.79 ± 623.11	15.51 ± 14.88
HopperWall-v0	687.58 ± 58.81	695.00 ± 93.70	695.94 ± 102.96
Walker2dWithSensor-v0	1897.74 ± 1101.13	3357.76 ± 1142.85	-2.27 ± 8.84
Walker2dWall-v0	1271.78 ± 881.57	974.83 ± 664.29	635.45 ± 303.73
HalfCheetahWithSensor-v0	2924.83 ± 165.69	2754.58 ± 151.81	-296.32 ± 110.45
HalfCheetahWall-v0	2022.90 ± 826.91	2159.17 ± 805.27	2043.85 ± 807.09
HumanoidWithSensor-v0	339.37 ± 47.38	285.70 ± 51.99	67.37 ± 8.31
HumanoidWall-v0	334.03 ± 51.49	328.90 ± 57.41	284.55 ± 49.07
Humanoid-v1	252.38 ± 12.05	269.23 ± 75.11	72.398 ± 2.56
HumanoidStandup-v0	75861.96 ± 19951.32	75906.45 ± 22390.07	70659.6 ± 19479.4
HumanoidStandupAndRun-v0	71269.19 ± 16689.99	73919.85 ± 19215.23	70021.9±18660.3
HumanoidWithSensor-v0	112.74 ± 23.09	114.75 ± 13.48	64.6059 ± 1.76
HumanoidStandupWithSensor-v0	53124.35 ± 15136.53	58335.81 ± 16259.60	52029.5 ± 13585.10
HumanoidStandupAndRunWithSensor-v0	59263.15 ± 12285.51	62570.26 ± 14258.35	55929.7 ± 15432.20
HumanoidStandupAndRunWall-v0	61468.03 ± 16135.02	66789.60 ± 14405.80	61764.5 ± 15150.20

Table 4.3: Results for modified running tasks with sensors, walls, or multiple goals. Environments are grouped together by motif and trained in order as listed here.

Environment	Fully Trained	After Env Training	First Step
Striker-v0	-124.87 ± 47.33	-114.61 ± 36.93	-590.61 ± 78.77
StrikerMovingStart-v0	-163.08 ± 76.29	-146.06 ± 60.21	-171.06 ± 92.10
Average for Grouping	-143.97 ± 66.29	-130.33 ± 43.57	-380.83 ± 85.44
Pusher-v0	-24.83 ± 2.39	-24.59 ± 4.01	-209.57 ± 7.46
PusherMovingGoal-v0	-28.01 ± 7.24	-27.76 ± 6.20	-34.90 ± 9.38
Average for Grouping	-26.42 ± 5.62	-26.17 ± 5.11	-122.24 ± 8.42

Table 4.4: Results for arm-based task groups. Same parameters as described in Table 4.1. Environments are grouped together by motif and trained in order as listed here.

1989) in the policy prevents generalization to earlier tasks. This can be seen in Tables 4.1 and 4.2. First, when evaluating the final policy on all of the previously trained environments (the "Fully Trained" column), performance decreases monotonically as we move backwards over the environments. Additionally, immediately after training on the earlier environments (the "After Env Training" column), the performance on the sample rollouts is much higher than that of the final policy (which has seen all the environments). This indicates that these groups of environments are good indicators for demonstrating and overcoming catastrophic forgetting in multitask learning.

In other environment variations (modified HalfCheetah and Walker2d environments), the agent's final policy outperforms both training from scratch (as in Table 4.1) and the 'After Env Training" result (as in Tables 4.1 and 4.2), which is evidence of positive forward transfer. The dynamics of these environments are not significantly perturbed by changes in physics, as the models have inherent stability. There remains significant room for future improvement upon our baseline. Future methods may achieve more efficient forward transfer between sequential environments. Furthermore, generalization across multiple tasks may come at a cost of higher variance in the policy (e.g., in Walker2d environments). Future improvements may also focus on generalization with constrained variance across trials (and thus higher safety when learning on new environments).

For the Humanoid-exclusive variations and Wall variations (as in Tables 4.2 and 4.3), TRPO is not able to learn a policy which can jump over a wall or learn a good policy on Humanoid tasks in the small number of iterations which we ran (1000 iterations). The results we see are on a comparable scale to (Duan et al., 2016).

In our new map navigation tasks, rewards remain at -1000, which is the initial lowest reward. That is, the agent never learns to find the goal using our default parameters and TRPO. This is to be expected as TRPO may not be suited for such a navigation task which requires large amounts of exploration with an extremely delayed reward. Other methods which encourage principled exploration and have a memory component to the policy may be more suitable for such tasks. We nevertheless share these environments with the community in an effort to drive investigation into creating complex policies for simultaneous localization, exploration and goal searching in settings where goals and obstacles vary between tasks.

4.4 Related Work

Several works investigate multitask or transfer learning with MuJoCo tasks. These tasks include: navigating around a wall (where a wall separates an agent from its goal); the OpenAI Gym Reacher environment with an added image state space of the environment; jumping over a wall using a model similar to the OpenAI Half-Cheetah environment (Finn et al., 2017); varying the gravity of various standard OpenAI Gym benchmark environments (Reacher, Hopper, Humanoid, HalfCheetah) and transferring between the modified environments; adding motor noise to the same set of environments (Christiano et al., 2016); simulated grasping and stacking using a Jaco arm (Rusu et al., 2017); and several custom grasping and manipulation tasks to demonstrate learning invariant feature spaces (Gupta et al., 2017).

Other works investigate using classical control systems and robotics simulations with a set of varied hyperparameters for each environment. These include: a simple mass spring damper task, cart-pole with continuous control; a three-link inverted pendulum with continuous control; a quadrotor control task (Ammar et al., 2014); a double-linked pendulum task; a modified cartpole balancing task which can transfer to physical system (Higuera et al., 2017).

4.5 Discussion

Our initial release investigates adding flexibility to standard OpenAI gym MuJoCo environments: modifying gravity, adding sensor readouts and a random wall obstacle, perturbing body-part sizes, and adding random goal/start state positions for arm environments. We also add an original set of environments for learning policies in continuous navigation tasks. In future releases we also plan to add standard environments for adding motor noise, arm environments where the end-goal position has a velocity (such that the arm must track the target), and making the sensor-based environments more realistic (and thus more transferable to real-world systems).

The release of benchmark multitask learning environments for RL in continuous domains is an important endeavor. As we have seen in the previous section, reproducibility

for RL algorithms is already difficult. In continuous control multitask learning settings, this difficulty is multiplied since often different novel works compare their algorithms in specialized environments – specific only to that work and commonly unreleased to the public. Without a standard set of benchmark environments, it is extremely difficult to discern which is in fact the best multitask learning approach. Here, we present an initial set of such environments. While, they are relatively simple in nature, we hope that these environments will be expanded and used by the community in future evaluations. We hope that as algorithms master these problems, more complex ones can be added to this benchmark set for reproducibility in settings addressing reusability.

Part III

Reusability

5

OptionGAN: Learning Joint Reward-Policy Options using Generative Adversarial Inverse Reinforcement Learning

A long term goal of Inverse Reinforcement Learning (IRL) is to be able to learn underlying reward functions and policies solely from human video demonstrations. We call such a case, *one-shot transfer learning in IRL*. This is the concept of transferring knowledge from expert demonstrations in different settings (e.g., tasks or environment dynamics) by learning in a new environment with no demonstrations in the new setting. We call this *one-shot* since the novice agent must reuse or transfer information from the expert settings while learning in one-shot on the new environment. For example, given only demonstrations of a human walking on earth, can an agent learn to walk on the moon?

However, such demonstrations would undoubtedly come from a wide range of settings and environments and may not conform to a single reward function. This proves detrimental to current methods which might over-generalize and cause poor performance. In forward RL, decomposing a policy into smaller specialized policy options has been shown to improve results for exactly such cases (Sutton et al., 1999; Bacon et al., 2017). Thus, we extend the options framework to IRL and decompose both the reward function and policy.

Our method is able to learn deep policies which can specialize to the set of best-fitting experts. Hence, it excels at one-shot transfer learning where single-approximator methods waver.

To accomplish this, we make use of the Generative Adversarial Imitation Learning (GAIL) framework (Ho and Ermon, 2016) and formulate a method for learning joint reward-policy options with adversarial methods in IRL. As such, we call our method OptionGAN. This method can implicitly learn divisions in the demonstration state space and accordingly learn policy and reward options. Leveraging a correspondence between Mixture-of-Experts and one-step options, we learn a decomposition of rewards and the policy-over-options in an end-to-end fashion. This decomposition is able to capture simple problems and learn any of the underlying rewards in one shot. This gives flexibility and benefits for a variety of future applications (both in reinforcement learning and standard machine learning).

We evaluate OptionGAN in the context of continuous control locomotion tasks, considering both simulated MuJoCo locomotion OpenAI Gym environments (Brockman et al., 2016), modifications of these environments for task transfer (Henderson et al., 2017), and a more complex Roboschool task (Schulman et al., 2017). We show that the final policies learned using joint reward-policy options outperform a single reward approximator and policy network in most cases, and particularly excel at one-shot transfer learning.

5.1 Preliminaries and Notation

Reinforcement Learning, Markov Decision Processes (MDPs), and Policy Gradients For an introduction to these topics, please refer to Section 2.1.

The Options framework We will briefly review again the notion of options and contextualize it within the work for this section. In reinforcement learning, an option ($\omega \in \Omega$) can be defined by a triplet ($I_\omega, \pi_\omega, \beta_\omega$). In this definition, π_ω is called an intra-policy option, $I_\omega \subseteq S$ is an initiation set, and $\beta_\omega : S \rightarrow [0, 1]$ is a termination function (i.e., the probability that an option ends at a given state) (Sutton et al., 1999). Furthermore, π_Ω is the policy-over-options. That is, π_Ω determines which option π_ω an agent picks to use until the termination function β_ω indicates that a new option should be chosen. Other works explicitly formulate *call-and-return* options, but we instead simplify to *one-step* options, where

$\beta_\omega(s) = 1; \forall \omega \in \Omega, \forall s \in S$. One-step options have long been discussed as an alternative to temporally extended methods and often provide advantages in terms of optimality and value estimation (Sutton et al., 1999; Dietterich, 2000; Daniel et al., 2012). Furthermore, we find that our options still converge to temporally extended and interpretable actions.

Mixture-of-Experts The idea of creating a Mixture-of-Experts was initially formalized to improve learning of neural networks by dividing the input space among several networks and then combining their outputs through a soft weighted average (Jacobs et al., 1991). It has since come into prevalence for generating extremely large neural networks (Shazeer et al., 2017). In our formulation of joint reward-policy options, we leverage a correspondence between Mixture-of-Experts and options. In the case of one-step options, the policy-over-options (π_Ω) can be viewed as a specialized gating function over experts:

$$\sum_{\omega} \pi_\Omega(\omega|s)\pi_\omega(a|s), \tag{5.1}$$

where intra-option policies are denoted by $\pi_\omega(a|s)$. Several works investigate convergence to a sparse and specialized Mixture-of-Experts (Jacobs et al., 1991; Shazeer et al., 2017). We leverage these works to formulate a Mixture-of-Experts which converges to one-step options.

Generative Adversarial Networks Borrowing from game theoretic principles in adversarial games, Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) are a method for training generative models. Typically a GAN consists of a generator G and a discriminator D. The goal of the discriminator is to label samples as belonging to an expert (the real data) or a novice (the generated data). The goal for the generator is to generate samples from noise such that they fool the discriminator into labeling the samples as expert-made. This can be formulated as a minimax game (as will be described later). GANs have been used for a myriad of applications including image and video generation (Dong et al., 2017; Tulyakov et al., 2017), natural language generation (Subramanian et al., 2017), and even malware generation (Hu and Tan, 2017). However, for discrete domains the end-to-end back-propagation proposed in the original setting of (Goodfellow et al., 2014) is typically modified. This is due to the fact that additive noise is used to generate samples

in the original setting, while in discrete spaces it is very difficult to add small amounts of noise as this would constitute a step to an entirely different discrete state. Instead, methods in such settings (Li et al., 2017a) leverage reinforcement learning and the log likelihood trick. This approach is nearly identical to the approach taken in generative adversarial imitation learning (Ho and Ermon, 2016), which we will describe in the next section.

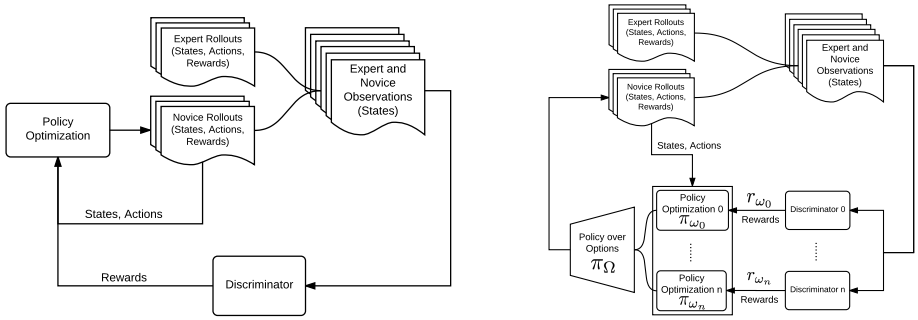

Figure 5.1: Generative Adversarial Inverse Reinforcement Learning (left) and OptionGAN (right) Architectures

Inverse Reinforcement Learning Inverse Reinforcement Learning was first formulated in the context of MDPs by (Ng and Russell, 2000). In later work, a parametrization of the reward function is learned as a linear combination of the state feature expectation so that the hyperdistance between the expert and the novice's feature expectation is minimized (Abbeel and Ng, 2004). It has also been shown that a solution can be formulated using the maximum entropy principle, with the goal of matching feature expectation as well (Ziebart et al., 2008). Generative adversarial imitation learning (GAIL) make use of adversarial techniques from (Goodfellow et al., 2014) to perform a similar feature expectation matching (Ho and Ermon, 2016). In this case, a discriminator uses state-action pairs (transitions) from the expert demonstrations and novice rollouts to learn a binary classification probability distribution. The probability that a state belongs to an expert demonstration can then be used as the reward for a policy optimization step. However, unlike GAIL, we do not assume knowledge of the expert actions. Rather, we rely solely on observations in the discriminator problem. We therefore refer to our baseline approach as Generative Adversarial Inverse Reinforcement Learning (IRLGAN) as opposed to imitation learning. It is important to note that IRLGAN is GAIL without known actions, we adopt the different

naming scheme to highlight this difference. As such, our adversarial game optimizes:

$$\max_{\pi_\Theta} \min_{R_{\hat{\Theta}}} -[\mathbb{E}_{\pi_\Theta}[\log R_{\hat{\Theta}}(s)] + \mathbb{E}_{\pi_E}[\log(1 - R_{\hat{\Theta}}(s))]] \tag{5.2}$$

where π_Θ and π_E are the policy of the novice and expert parameterized by Θ and E, respectively, and $R_{\hat{\Theta}}$ is the discriminator probability that a sample state belongs to an expert demonstration (parameterized by $\hat{\Theta}$). We use this notation since in this case the discriminator approximates a reward function. We use TRPO during the policy optimization step for simple tasks. However, for complex tasks we adopt PPO. We use these methods as they are the most stable methods seen in Chapter 3. We use TRPO in simple settings as we find it achieves more stable results than PPO. We use PPO in more complex settings as it is able to adapt the trust region to allow for more exploration. We find that this significantly decreases the number of samples required to achieve acceptable returns as compared to TRPO. Figure 5.1 and Algorithm 2 show an outline for the general IRLGAN process.

Algorithm 2: IRLGAN

Input : Expert trajectories $\tau_E \sim \pi_E$.
Initialize $\Theta, \hat{\Theta}$
for $i = 0, 1, 2, \ldots$ **do**
  Sample trajectories $\tau_N \sim \pi_{\Theta_i}$
  Update discriminator parameters ($\hat{\Theta}$) according to:

$$L_{\hat{\Theta}} = \mathbb{E}_{s\sim\tau_N}[\log R_{\hat{\Theta}}(s)] + \mathbb{E}_{s\sim\tau_E}[\log(1 - R_{\hat{\Theta}}(s))]$$

  Update policy (with constrained update step and parameters θ) according to:

$$\mathbb{E}_{\tau_N}[\nabla_\Theta \log \pi_{\Theta_i}(a|s)\mathbb{E}_{\tau_N}[\log(R_{\hat{\Theta}_{i+1}}(s))|s_0 = \bar{s}]]$$

end

5.2 Reward-Policy Options Framework

Based on the need to infer a decomposition of underlying reward functions from a wide range of expert demonstrations in one-shot transfer learning, we extend the options framework for decomposing rewards as well as policies. In this way, intra-option policies, de-

composed rewards, and the policy-over-options can all be learned in concert in a cohesive framework. In this case, an option is formulated by a tuple: $(I_\omega, \pi_\omega, \beta_\omega, r_\omega)$. Here, r_ω is a reward option from which a corresponding intra-option policy π_ω is derived. That is, each policy option is optimized with respect to its own local reward option. The policy-over-options not only chooses the intra-option policy, but the reward option as well: $\pi_\Omega \rightarrow (r_\omega, \pi_\omega)$. For simplicity, we refer to the policy-over-reward-options as r_Ω (in our formulation, $r_\Omega = \pi_\Omega$). There is a parallel to be drawn from this framework to Feudal RL (Dayan and Hinton, 1993), but here the intrinsic reward function is statically bound to each worker (policy option), whereas in that framework the worker dynamically receives a new intrinsic reward from the manager.

To learn joint reward-policy options, we present a method which fits into the framework of IRLGAN. We reformulate the discriminator as a Mixture-Of-Experts and re-use the gating function when learning a set of policy options. We show that by properly formulating the discriminator loss function, the Mixture-Of-Experts converges to one-step options. This formulation also allows us to use regularizers which encourage distribution of information, diversity, and sparsity in both the reward and policy options.

5.3 Learning Joint Reward-Policy Options

The use of one-step options allows us to learn a policy-over-options in an end-to-end fashion as a Mixture-of-Experts formulation. In the one-step case, selecting an option $(\pi_{\omega,\theta})$ using the policy-over-options $(\pi_{\Omega,\zeta})$ can be viewed as a mixture of completely specialized experts such that: $\pi_\Theta(a|s) = \sum_\omega \pi_{\Omega,\zeta}(\omega|s)\pi_{\omega,\theta}(a|s)$. The reward for a given state is composed as: $R_{\Omega,\hat{\Theta}}(s) = \sum_\omega \pi_{\Omega,\zeta}(\omega|s) r_{\omega,\hat{\theta}}(s)$, where $\zeta, \theta \in \Theta, \hat{\theta} \in \hat{\Theta}$ are the parameters of the policy-over-options, policy options, and reward options, respectively. Thus, we reformulate our discriminator loss as a weighted mixture of completely specialized experts in Eq. 5.3. This allows us to update the parameters of the policy-over-options and reward options together during the discriminator update:

$$L_\Omega = \mathbb{E}_\omega \left[\pi_{\Omega,\zeta}(\omega|s) L_{\hat{\theta},\omega} \right] + L_{reg}. \tag{5.3}$$

Here, $L_{\hat{\theta},\omega}$ is the sigmoid cross-entropy loss of the reward options (discriminators). As will be discussed later on, L_{reg} is a penalty, or set of penalties, which can encourage certain properties of the policy-over-options or the overall reward signal. As can be seen in Algorithm 3 and Figure 5.1, this loss function can fit directly into the IRLGAN framework.

Algorithm 3: OptionGAN

Input : Expert trajectories $\tau_E \sim \pi_E$.
Initialize $\theta, \hat{\theta}$
for $i = 0, 1, 2, \ldots$ **do**
  Sample trajectories $\tau_N \sim \pi_{\Theta_i}$
  Update discriminator options parameters $\hat{\theta}, \omega$ and policy-over-options parameters ζ, to minimize:

$$L_\Omega = \mathbb{E}_\omega \left[\pi_{\Omega,\zeta}(\omega|s) L_{\hat{\theta},\omega} \right] + L_{reg}$$

$$L_{\hat{\theta},\omega} = \mathbb{E}_{\tau_N}[\log r_{\hat{\theta},\omega}(s)] + \mathbb{E}_{\tau_E}[\log(1 - r_{\hat{\theta},\omega}(s))]$$

  Update policy options (with constrained update step and parameters $\theta_\omega \in \Theta_\Omega$) according to:

$$\mathbb{E}_{\tau_N}[\nabla_\theta \log \pi_\Theta(a|s) \mathbb{E}_{\tau_N}[\log(R_{\Omega,\hat{\Theta}}(s))|s_0 = \bar{s}]]$$

end

Having updated the parameters of the policy-over-options and reward options, standard PG methods can be used to optimize the parameters of the intra-option policies. This can be done by weighting the average of the intra-option policy actions with the policy-over-options $\pi_{\Omega,\zeta}$. While it is possible to update each intra-option policy separately as in (Bacon et al., 2017), this Mixture-of-Experts formulation is equivalent, as discussed in the next section. Once the gating function specializes over the options, all gradients except for those related to the intra-option policy selected would be weighted by zero. We find that this end-to-end parameter update formulation leads to easier implementation and smoother learning with constraint-based methods.

5.4 Mixture-of-Experts as Options

To ensure that our Mixture-of-Experts formulation converges to options in the optimal case, we must properly formulate our loss function such that the gating function specializes over experts. While it may be possible to force a sparse selection of options through a top-k choice as in (Shazeer et al., 2017), we find that this leads to instability since for $k = 1$ the top-k function is not differentiable. As is specified in (Jacobs et al., 1991), a loss function of the form $L = (y - \frac{1}{||\Omega||}\sum_{\omega} \pi_{\Omega}(\omega|s)y_{\omega}(s))^2$ draws cooperation between experts, but a reformulation of the loss, $L = \frac{1}{||\Omega||}\sum_{\omega} \pi_{\Omega}(\omega|s)(y - y_{\omega}(s))^2$, encourages specialization.

If we view our policy-over-options as a softmax (i.e., $\pi_{\Omega}(\omega|s) = \frac{\exp(z_{\omega}(s))}{\sum_i \exp(z_i(s))}$), then the derivative of the loss function with respect to the gating function becomes:

$$\frac{dL}{dz_{\omega}} = \frac{1}{||\Omega||}\pi_{\Omega}(\omega|s)\left((y - y_{\omega}(s))^2 - L\right) \tag{5.4}$$

This can intuitively be interpreted as encouraging the gating function to increase the likelihood of choosing an expert when its loss is less than the average loss of all the experts. The gating function will thus move toward deterministic selection of experts.

As we can see in Eq. 5.3, we formulate our discriminator loss in the same way, using each reward option and the policy-over-options as the experts and gating function respectively. This ensures that the policy-over-options specializes over the state space and converges to a deterministic selection of experts. Hence, we can assume that in the optimal case, our formulation of an Mixture-of-Experts-style policy-over-options is equivalent to one-step options. Our characterization of this notion of Mixture-of-Experts-as-options is further backed by experimental results. Empirically, we still find temporal coherence across option activation despite not explicitly formulating call-and-return options as in (Bacon et al., 2017).

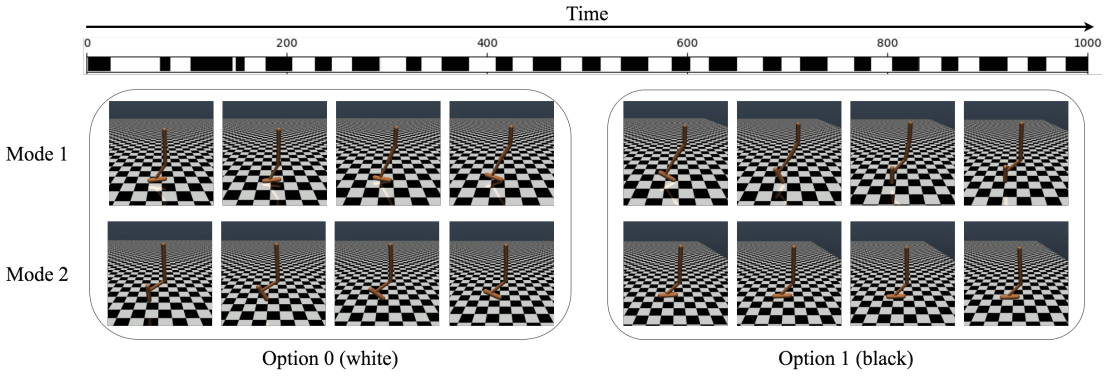

Figure 5.2: The policy-over-options elicits two interpretable behaviour modes per option, but temporal cohesion and specialization is seen between these behaviour modes across time within a sample rollout trajectory.

5.4.1 Regularization Penalties

Due to our formulation of Mixture-of-Experts as options, we can learn our policy-over-options in an end-to-end manner. This allows us to add additional terms to our loss function to encourage the appearance of certain target properties.

Sparsity and Variance Regularization

To ensure an even distribution of activation across the options, we look to conditional computation techniques that encourage sparsity and diversity in hidden layer activations and apply these to our policy-over-options (Bengio et al., 2016). We borrow three penalty terms L_b, L_e, L_v (adopting a similar notation). In the minibatch setting, these are formulated as:

$$L_b = \sum_{\omega} ||\mathbb{E}_s[\pi_\Omega(\omega|s)] - \tau||_2, \tag{5.5}$$

$$L_e = \mathbb{E}_s\left[|| \left(\frac{1}{||\Omega||} \sum_{\omega} \pi_\Omega(\omega|s) \right) - \tau||_2 \right], \tag{5.6}$$

$$L_v = -\sum_{\omega} \text{var}_\omega\{\pi_\Omega(\omega|s)\}, \tag{5.7}$$

where τ is the target sparsity rate (which we set to $\tau = 0.5$ for all cases). Here, L_b encourages the activation of the policy-over-options with target sparsity τ "in expectation over the data" (Bengio et al., 2016). Essentially, L_b encourages a uniform distribution of options over the data while L_e drives toward a target sparsity of activations per example (doubly encouraging our mixtures to be sparse). Here, L_v also encourages varied π_Ω activations while discouraging uniform selection.

Mutual Information Penalty

To ensure the specialization of each option to a specific partition of the state space, a mutual information (MI) penalty is added.[1] In a similar manner as Liu and Yao (2002), we thus minimize mutual information pairwise between option distributions

$$I(F_i; F_j) = -\frac{1}{2}\log(1 - \rho_{ij}^2), \tag{5.8}$$

where F_i and F_j are the outputs of reward options i and j respectively, and ρ_{ij} the correlation coefficient of F_i and F_j, defined as $\rho_{ij} = \frac{E[(F_i - E[F_i])(F_j - E[F_j])]}{\sigma_i^2 \sigma_j^2}$.

The resulting loss term is thus computed as:

$$L_{\text{MI}} = \sum_{\omega \in \Omega} \sum_{\substack{\hat{\omega} \in \Omega \\ \omega \neq \hat{\omega}}} I(\pi_\omega, \pi_{\hat{\omega}}). \tag{5.9}$$

Thus the overall regularization term becomes:

$$\mathcal{L}_{reg} = \lambda_b L_b + \lambda_e L_e + \lambda_v L_v + \lambda_{\text{MI}} L_{\text{MI}}. \tag{5.10}$$

[1] While it may be simpler to use an entropy regularizer, we found that in practice it performs worse. Entropy regularization encourages exploration (Mnih et al., 2016). In the OptionGAN setting, this results in unstable learning, while the mutual information term encourages diversity in the options while providing stable learning.

5.5 Experiments

To evaluate our method of learning joint reward-policy options, we investigate continuous control tasks. We divide our experiments into 3 settings: simple locomotion tasks, one-shot transfer learning, and complex tasks. We compare OptionGAN against IRLGAN in all scenarios, investigating whether dividing the reward and policy into options improves performance against the single approximator case.[2] Table 5.1 shows the overall results of our evaluations and we highlight a subset of learning curves in Figure 5.3. We find that in nearly every setting, the final optionated policy learned by OptionGAN outperforms the single approximator case.

Task	Expert	IRLGAN	OptionGAN (2ops)	OptionGAN (4ops)
Hopper-v1	3778.8 $\pm$ 0.3	**3736.3 $\pm$ 152.4**	*3641.2 $\pm$ 105.9*	*3715.5 $\pm$ 17.6*
HalfCheetah-v1	4156.9 $\pm$ 8.7	3212.9 $\pm$ 69.9	**3714.7 $\pm$ 87.5**	*3616.1 $\pm$ 127.3*
Walker2d-v1	5528.5 $\pm$ 7.3	*4158.7 $\pm$ 247.3*	*3858.5 $\pm$ 504.9*	**4239.3 $\pm$ 314.2**
Hopper (One-Shot)	3657.7 $\pm$ 25.4	2775.1 $\pm$ 203.3	*3409.4 $\pm$ 80.8*	**3464.0 $\pm$ 67.8**
HalfCheetah (One-Shot)	4156.9 $\pm$ 51.3	1296.3 $\pm$ 177.8	1679.0 $\pm$ 284.2	**2219.4 $\pm$ 231.8**
Walker (One-Shot)	4218.1 $\pm$ 43.1	3229.8 $\pm$ 145.3	**3925.3 $\pm$ 138.9**	*3769.40 $\pm$ 170.4*
HopperSimpleWall-v0	3218.2 $\pm$ 315.7	*2897.5 $\pm$ 753.5*	*3140.3 $\pm$ 674.3*	**3272.3 $\pm$ 569.0**
RoboschoolHumanoidFlagrun-v1	2822.1 $\pm$ 531.1	*1455.2 $\pm$ 567.6*	*1868.9 $\pm$ 723.7*	**2113.6 $\pm$ 862.9**

Table 5.1: True Average Return with the standard error across 10 trials on the 25 final evaluation rollouts using the final policy.

5.5.1 Experimental Setup

All shared hyperparameters are held constant between IRLGAN and OptionGAN evaluation runs. All evaluations are averaged across 10 trials, each using a different random seed. We use the average return of the true reward function across 25 sample rollouts as the evaluation metric. Multilayer perceptrons are used for all approximators as in (Ho and Ermon, 2016). For the OptionGAN intra-option policy and reward networks, we use shared hidden layers. That is $r_\omega, \forall \omega \in \Omega$ all share hidden layers and $\pi_\omega, \forall \omega \in \Omega$ share hidden layers. We use separate parameters for the policy-over-options π_Ω. Shared layers are used to ensure a fair comparison against a single network of the same number of hidden layers. For simple settings we use fully connected layers of sizes $(64, 64)$ and for complex experiments are $(128, 128)$ – that is, two fully connected layers of equal size in each case. For the 2-options

[2] Extended experimental details and results can be found in Appendix B.1. Code is located at: https://github.com/Breakend/OptionGAN.

case we set $\lambda_e = 10.0, \lambda_b = 10.0, \lambda_v = 1.0$ based on a simple hyperparameter search and reported results from (Bengio et al., 2016). For the 4-options case we relax the regularizer that encourages a uniform distribution of options (L_b), setting $\lambda_b = 0.01$.

5.5.2 Simple Tasks

First, we investigate simple settings without transfer learning for a set of benchmark locomotion tasks provided in OpenAI Gym (Brockman et al., 2016) using the MuJoCo simulator (Todorov et al., 2012). We use the Hopper-v1, HalfCheetah-v1, and Walker2d-v1 locomotion environments. The results of this experiment are shown in Table 5.1 and sample learning curves for Hopper and HalfCheetah can be found in Figure 5.3. We use 10 expert rollouts from a policy trained using TRPO for 500 iterations.

In these simple settings, OptionGAN converges to policies which perform as well or better than the single approximator setting. Importantly, even in these simple settings, the options which our policy selects have a notion of temporal coherence and interpretability despite not explicitly enforcing this in the form of a termination function. This can be seen in the two option version of the Hopper-v1 task in Figure 5.2. We find that generally each option takes on two behaviour modes. The first option handles: (1) the rolling of the foot during hopper landing; (2) the folding in of the foot in preparation for floating. The second option handles: (1) the last part of take-off where the foot is hyper-extended and body flexed; (2) the part of air travel without any movement.

5.5.3 One-Shot Transfer Learning

We also investigate one-shot transfer learning. In this scenario, the novice is trained on a target environment, while expert demonstrations come from a similar task, but from environments with altered dynamics (i.e., one-shot transfer from varied expert demonstrations to a new environment). To demonstrate the effectiveness of OptionGAN in these settings, we use expert demonstrations from environments with varying gravity conditions as seen in (Henderson et al., 2017; Christiano et al., 2016). We vary the gravity (0.5, 0.75, 1.25, 1.5 of Earth's gravity) and train experts using TRPO for each of these. We gather 10 expert trajectories from each gravity variation, for a total of 40 expert rollouts, to train a novice

agent on the normal Earth gravity environment (the default -v1 environment as provided in OpenAI Gym). As aforementioned, the target environment has no expert demonstrations in it, and the novice agent must infer a behaviour for its current environment from the varied settings of the expert demonstrations. We repeat this for Hopper-v1, HalfCheetah-v1, and Walker2D-v1.

These gravity tasks are selected due to the demonstration in Chapter 4 that learning sequentially on these varied gravity environments causes catastrophic forgetting of the policy on environments seen earlier in training. This suggests that the dynamics are varied enough that trajectories are difficult to generalize across, yet still share some state representations and task goals. As seen in Figure 5.3, using options can cause significant performance increases in this area, but performance gains can vary across the number of options and the regularization penalty as seen in Table 5.1.

5.5.4 Complex Tasks

Lastly, we investigate slightly more complex tasks. We utilize the HopperSimpleWall-v0 environment provided by the gym-extensions framework (Henderson et al., 2017), as well as the RoboschoolHumanoidFlagrun-v1 environment used in (Schulman et al., 2017). In the first, a wall is placed randomly in the path of the Hopper-v1 agent and simplified sensor readouts are added to the observations as in (Wang et al., 2017). In the latter, the goal is to run and reach a frequently changing target. This is an especially complex task with a highly varied state space. In both cases we use an expert trained with TRPO and PPO respectively, to generate 40 expert rollouts. For the Roboschool environment, we find that TRPO does not allow enough exploration to perform adequately, and thus we switch our policy optimization method to the clipping-objective version of PPO.

5.6 Ablation Investigations

Convergence of Mixtures to Options

To show that our formulation of Mixture-of-Experts decomposes to options in the optimal case, we investigate the distributions of our policy-over-options. We find that across

5.6 Ablation Investigations

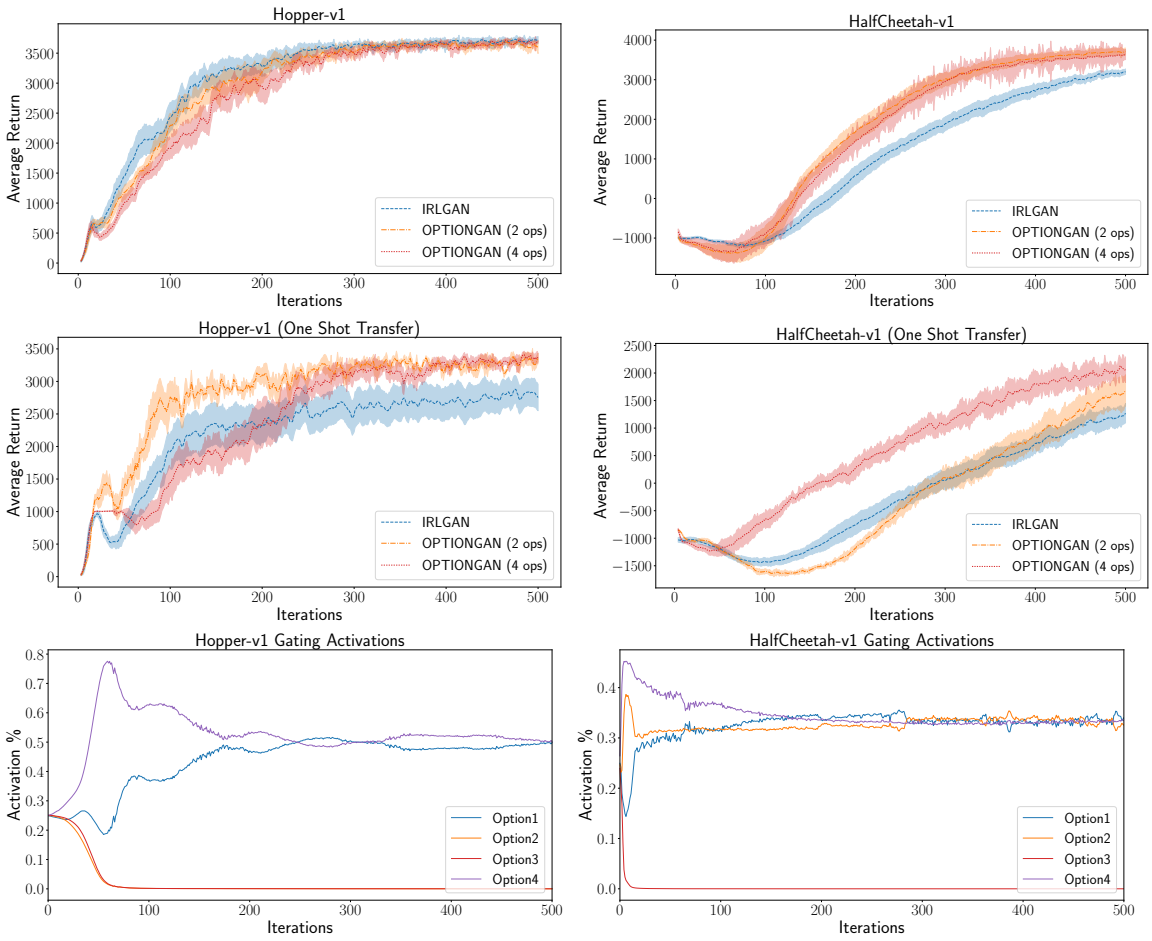

Figure 5.3: Top Row: Simple locomotion curves. Error bars indicate standard error of average returns across 10 trials on 25 evaluation rollouts. Middle Row: One-shot transfer experiments with 40 expert demonstrations from varied gravity environments without any demonstrations on the novice training environment training on demonstrations from 0.5G, 0.75G, 1.25G, 1.5G gravity variations. Bottom Row: Activations of policy-over-options over time with 4 options on training samples in the one-shot transfer setting with $\lambda_b = 0.01$.

5.6 Ablation Investigations

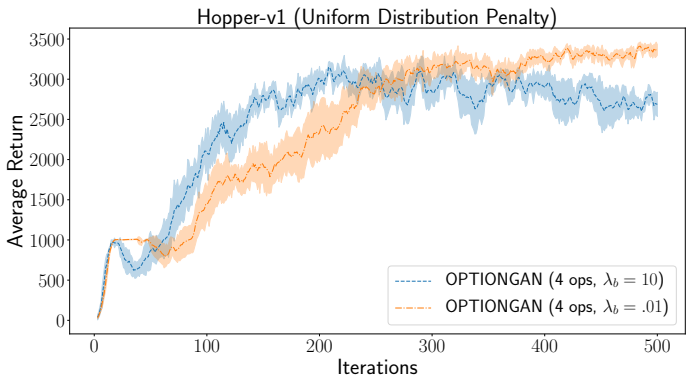

Figure 5.4: Effect of uniform distribution regularizer. Average π_Ω across final sample novice rollouts: $\lambda_b = 10.0$, $[.27, .21, .25, .25]$; $\lambda_b = .01$, $[0., 0., .62, .38]$.

40 trials, 100% of activations fell within a reasonable error bound of deterministic selection across 1M samples. That is, in 40 total trials across 4 environments (Hopper-v1, HalfCheetah-v1, Walker2d-v1, RoboschoolHumanoidFlagrun-v1), policies were trained for 500 iterations (or 5k iterations in the case of RoboschoolHumanoidFlagrun-v1). We collected 25k samples at the end of each trial. Among the gating activations across the samples, we recorded the number of gating activations within the range $\{0 + \epsilon, 1 - \epsilon\}$ for $\epsilon = 0.1$. All of the recorded activations (100%) fell within this range, and 98.72% fell within range $\epsilon = 1^{-3}$. Thus at convergence, both intuitively and empirically we can refer to our gating function over experts as the policy-over-options and each of the experts as options.

Effect of Uniform Distribution Regularizer

We find that forcing a uniform distribution over options can potentially be harmful. This can be seen in the experiment in Figure 5.4, where we evaluate the 4 option case with $\lambda_b = \{0.1, 10\}$. However, relaxing the uniform constraint results in rapid performance increases, particularly in the HalfCheetah-v1 environment where, as seen in Figure 5.3, there are increases learning speed with 4 options.

There is an intuitive explanation for this. In the 4-option case, with a relaxed uniform distribution penalty, we allow options to drop out during training. In the case of Hopper and Walker tasks, generally 2 options drop out slowly over time, but in HalfCheetah, only

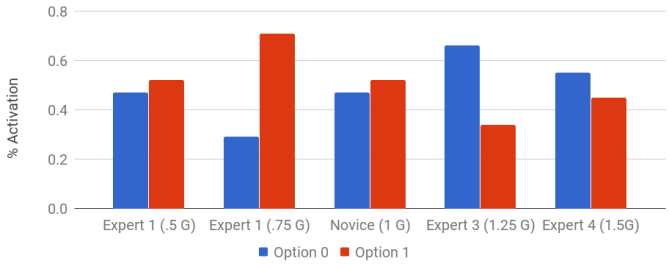

Figure 5.5: Probability distribution of π_Ω over options on expert demonstrations. Inherent structure is found in the underlying demonstrations. The .75G demonstration state spaces are significantly assigned to Option 1 and similarly, the 1.25G state spaces to Option 0.

one option drops out in the first 20 iterations with a uniform distribution remaining across the remaining options as seen in Figure 5.3. We posit that in the case of HalfCheetah there is enough mutually exclusive information in the environment state space to divide across 3 options, quickly causing a rapid gain in performance, while the Hopper tasks do not settle as quickly and thus do not see that large gain in performance.

Latent Structure in Expert Demonstrations

Another benefit of using options in the IRL transfer setting is that the underlying latent division of the original expert environments is learned by the policy-over-options. As seen in Figure 5.5, the expert demonstrations have a clear separation among options. We suspect that options further away from the target gravity are not as specialized due to the fact that their state spaces are covered significantly by a mixture of the closer options (see Appendix B.4 for supporting projected state space mappings). This indicates that the policy-over-options specializes over the experts and is thus inherently beneficial for use in one-shot transfer learning.

5.7 Related Work

One goal in robotics research is to create a system which learns how to accomplish complex tasks simply from observing an expert's actions (such as videos of humans performing actions). While IRL has been instrumental in working towards this goal, it has become clear

that fitting a single reward function which generalizes across many domains is difficult. To this end, several works investigate decomposing the underlying reward functions of expert demonstrations and environments in both IRL and RL (Krishnan et al., 2016; Sermanet et al., 2017; Choi and eung Kim, 2012; Babes et al., 2011; van Seijen et al., 2017). For example, in (Krishnan et al., 2016), reward functions are decomposed into a set of subtasks based on segmenting expert demonstration transitions (known state-action pairs) by analyzing the changes in "local linearity with respect to a kernel function". Similarly, in (Sermanet et al., 2017), techniques in video editing based on information-similarity are adopted to divide a video demonstration into distinct sections which can then be recombined into a differentiable reward function.

However, simply decomposing the reward function may not be enough, the policy must also be able to adapt to different tasks. Several works have investigated learning a latent dimension along with the policy for such a purpose (Hausman et al., 2017; Wang et al., 2017; Li et al., 2017b). This latent dimension allows multiple tasks to be learned by one policy and elicited via the latent variable. In contrast, our work focuses on one-shot transfer learning. In the former work, the desired latent variable must be known and provided, whereas in our formulation the latent structure is inherently encoded in an unsupervised manner. This is inherently accomplished while learning to solve a task composed of a wide range of underlying reward functions and policies in a single framework. Overall, this work contains parallels to all of the aforementioned and other works emphasizing hierarchical policies (Daniel et al., 2012; Dietterich, 2000; Merel et al., 2017), but specifically focuses on leveraging Mixture-of-Experts and reward decompositions to fit into the *options* framework for efficient one-shot transfer learning in IRL.

5.8 Discussion

We propose a direct extension of the options framework by adding joint reward-policy options. We learn these options in the context of generative adversarial inverse reinforcement learning and show that this method outperforms the single policy case in a variety of tasks – particularly in transfer settings. Furthermore, the learned options demonstrate temporal and interpretable cohesion without specifying a call-and-return termination function.

5.8 Discussion

Our formulation of joint reward-policy options as a Mixture-of-Experts allows for: potential upscaling to extremely large networks as in (Shazeer et al., 2017), reward shaping in forward RL, and using similarly specialized Mixture-of-Experts in generative adversarial networks. This work presents an effective and extendable framework. Our optionated networks capture the problem structure effectively, which allows strong generalization in one-shot transfer learning. Moreover, as adversarial methods are now commonly used across a myriad of communities, we believe the embedding of options within this methodology is an excellent delivery mechanism to exploit the benefits of hierarchical RL in many new fields.

Part IV

Final Conclusion & Future Work

6

Final Conclusion & Future Work

6.1 Summary

This thesis has presented several lines of investigation, methods, and tools into reproducibility and reusability in deep reinforcement learning. Overall, we hope that the content presented in this thesis represents a building block which can be used to further the field and be used in other fields. In the context of reproducibility, we hope that novel works begin to use and expand on our methodology proposed here to ensure that correct experimental methodology pushes the field further at a faster rate. Furthermore, our goal with the introduction of OptionGAN is a framework which can be leveraged in both reinforcement learning and in other fields for better ways to build robust algorithms which can reuse information.

6.1.1 Reproducibility

We discuss several factors affecting reproducibility of reinforcement learning algorithms. The sensitivity of these algorithms to changes in reward scale, environment dynamics, and random seeds can be considerable and varies between algorithms and settings. We suggest experimental methodology and evaluation methods which can begin to help address the problem with reporting misleading or irreproducible results. This includes: running many trials with different random seeds and presenting the confidence intervals of the policies across trials; using statistical testing methods to evaluate whether reporting boosts in per-

formance were in fact significant; avoiding reporting only maximum metrics or top-N results without the full range; careful detailing of all hyperparameters; release of codebases. We present these methodologies such that deep reinforcement learning can move forward as a field without misdirection from misleading or irreproducible results.

We also present a set of benchmark environments that begin to help standardize evaluation for multitask, transfer, or lifelong learning. While many current works in the area use custom evaluation environments that are not open-source, these benchmark environments can be used as a tool to better understand standardized performance of algorithms on an open-source available set of environments. We also are actively maintaining this repository for expansion to more interesting and complex environments.

6.1.2 Reusability

Lastly, we investigate the notion of reusability from the perspective of inverse reinforcement learning. The aim is to reuse information from expert demonstration in environments with different dynamics to learn a task in an unseen environment. We leverage the benchmark environments previously mentioned to investigate this notion of one-shot transfer learning (i.e., reusability).

We propose a direct extension of the options framework by adding joint reward-policy options. We learn these options in the context of generative adversarial inverse reinforcement learning and show that this method outperforms the single policy case in a variety of tasks – particularly in transfer settings. This work presents an effective and extendable framework. Our optionated networks capture the problem structure effectively, which allows strong generalization in one-shot transfer learning. Moreover, as adversarial methods are now commonly used across a myriad of communities, we believe the embedding of options within this methodology is an excellent delivery mechanism to exploit the benefits of hierarchical reinforcement learning and reusability in many new fields.

6.2 Future Work

While this work presents a step forward in both the notions of reproducibility and reusability in deep reinforcement learning, there is still much work to be done in both of these areas.

6.2.1 Reproducibility

Due to the high variance in learning for these algorithms, there are many possible lines of investigations to address the issues of reproducibility. As aforementioned, due to the significant effects of hyperparameters (particularly reward scaling), one potential line of future investigation is in building hyperparameter agnostic algorithms. Such an approach would ensure that there is no unfairness introduced from external sources when comparing algorithms agnostic to parameters such as reward scale, batch size, or network structure. Furthermore, while we investigate an initial set of evaluation methods here, they may not be the best fit for comparing reinforcement learning algorithms. Several works have begun investigating policy evaluation methods for the purposes of safe reinforcement learning (Thomas and Brunskill, 2016; Thomas et al., 2015), but further work is needed in significance testing and statistical analysis. Similar lines of investigation to (Nadeau and Bengio, 2000; Bouckaert and Frank, 2004) would be helpful to determine the best methods for evaluating performance gain significance.

Furthermore, since benchmark environments are proxies for real-world applications to gauge generalized algorithm performance, perhaps more emphasis should be placed on the applicability of reinforcement learning algorithms to real-world tasks. That is, as there is often no clear winner among all benchmark environments, perhaps recommended areas of application should be demonstrated along with benchmark environment results when presenting a new algorithm. Perhaps novel works in reinforcement learning should also be answering the question: in what setting would this work be useful? This puts an emphasis on the usefulness of reinforcement learning algorithms in different fields rather than on performance gains in difficult to reproduce benchmarks.

That being said, the difficulty in reproducing reinforcement learning algorithms in simple benchmarks is perhaps also a problem with the stability of the algorithms themselves. Overall, increased effort is also needed to bring these algorithms to stable learning and reproducible behaviour.

6.2.2 Reusability

The notion of reusability in deep reinforcement learning covers a large range of techniques and problems. While we focus here on reusability of demonstrations from noisy and differing settings, there are many other areas where the techniques we present here can be applied. The optionated reward functions shown here can be used for optionated model-based reinforcement learning or reward shaping. Furthermore, the convergence of Mixture-of-Experts-as-options can be utilized in building large distributed optionated architectures for large-scale deep reinforcement learning. This work presents a stepping stone to future novel improvements in reusability in deep reinforcement learning in many areas.

6.2.3 Biologically Plausible Reinforcement Learning

One line of investigation that we would also like to highlight is the notion of biologically plausible deep reinforcement learning – an area of research (Rivest et al., 2005; Florian, 2007) that can gain inspiration from similar approaches in deep learning (Bengio et al., 2015, 2017). This notion could help to address some issues in reproducibility and reusability. While reinforcement learning in general finds a basis in human learning processes, there are still significant differences in learning mechanisms which some works try to address (Rivest et al., 2005; Florian, 2007). Some efforts in deep learning have begun to investigate possible methods for aligning methodologies with how the human brain learns (Bengio et al., 2015, 2017). However, with notions such as hyperparameters and simple random seeds affecting results of deep reinforcement learning algorithms, perhaps, to avoid such problems, more emphasis should be placed in reusing information about *how the human brain uses reinforcement learning*. For example, let us take the work we present: OptionGAN. While here, we add another set of hyperparameters (e.g., the number of options), perhaps a more biologically plausible notion of reward signal learning can avoid such added pre-specifications by using notions of neuroplasticity (e.g., rewiring the

options or actively changing the number of options). Overall, aligning methodologies with biological plausibility may be beneficial to both reinforcement learning techniques and the understanding of our own neurocomputational processes. Current improvements to reinforcement learning that hold no basis in biological plausibility may be useful – and may even outperform current biologically plausible algorithms as we understand them. However, as we improve algorithms which *are* biologically plausible, we step closer to understanding and replicating our own intelligence such that *reproducibility and reusability* can be ensured.

List of Publications

Published or submitted during Master of Science Program:

- **Peter Henderson**, Wei-Di Chang, Pierre-Luc Bacon, David Meger, Joelle Pineau, and Doina Precup. "OptionGAN: Learning Joint Reward-Policy Options using Generative Adversarial Inverse Reinforcement Learning." In *AAAI*. 2018.
- **Peter Henderson**, Wei-Di Chang, Florian Shkurti, Johanna Hansen, David Meger, and Gregory Dudek. "Benchmark Environments for Multitask Learning in Continuous Domains." In *ICML Lifelong Learning: A Reinforcement Learning Approach Workshop*. 2017.
- **Peter Henderson**, Matthew Vertescher, David Meger, and Mark Coates. "Cost Adaptation for Robust Decentralized Swarm Behaviour." In Submission to *IROS*. 2018.
- **Peter Henderson**, Koustuv Sinha, Nicolas Angelard-Gontier, Nan Rosemary Ke, Genevieve Fried, Ryan Lowe, and Joelle Pineau. "Ethical Challenges in Data-Driven Dialogue Systems." In *AIES*. 2018.
- **Peter Henderson***, Riashat Islam*, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. "Deep reinforcement learning that matters." In *AAAI*. 2018.
- **Peter Henderson***, Thang Doan*, Riashat Islam, and David Meger. "Bayesian Policy Gradients via Alpha Divergence Dropout Inference." In *Bayesian Deep Learning Workshop at NIPS*. 2017.
- Riashat Islam*, **Peter Henderson***, Maziar Gomrokchi, and Doina Precup. "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control." In *ICML Reproducibility in ML Workshop*. 2017.
- Florian Shkurti, Wei-Di Chang, **Peter Henderson**, Md Jahidul Islam, Juan Camilo Gamboa Higuera, Jimmy Li, Travis Manderson, Anqi Xu, Gregory Dudek, and Ju-

naed Sattar. "Underwater multi-robot convoying using visual tracking by detection." In *IROS*. 2017.

- Iulian Vlad Serban, Ryan Lowe, **Peter Henderson**, Laurent Charlin, and Joelle Pineau. "A survey of available corpora for building data-driven dialogue systems." In *Dialogue and Discourse*. 2018.

* indicates shared first authorship

Bibliography

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1–, New York, NY, USA. ACM.

Aleksandrov, V. M., Sysoyev, V. I., and Shemeneva, V. V. (1968). Stochastic Optimization. *Engineering Cybernetics*, 5:11–16.

Ammar, H. B., Eaton, E., Luna, J. M., and Ruvolo, P. (2015a). Autonomous Cross-Domain Knowledge Transfer in Lifelong Policy Gradient Reinforcement Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3345–3351.

Ammar, H. B., Eaton, E., Ruvolo, P., and Taylor, M. (2014). Online Multi-Task Learning for Policy Gradient Methods. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1206–1214.

Ammar, H. B., Tutunov, R., and Eaton, E. (2015b). Safe Policy Search for Lifelong Reinforcement Learning with Sublinear Regret. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2361–2369.

Babes, M., Marivate, V., Subramanian, K., and Littman, M. L. (2011). Apprenticeship Learning about Multiple Intentions. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 897–904.

Bacon, P.-L., Harb, J., and Precup, D. (2017). The Option-Critic Architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1726–1734.

Baker, M. (2016). 1,500 Scientists Lift the Lid on Reproducibility. *Nature*, 533(7604):452–454.

BIBLIOGRAPHY

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., Silver, D., and van Hasselt, H. P. (2017). Successor Features for Transfer in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 4056–4066.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A Distributional Perspective on Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bellman, R. (1957). A Markovian Decision Process. *Journal of Mathematics and Mechanics*, pages 679–684.

Bengio, E., Bacon, P.-L., Pineau, J., and Precup, D. (2016). Conditional Computation in Neural Networks for Faster Models. *Proceedings of the International Conference on Learning Representations (ICLR) Workshop*.

Bengio, Y. (2012). Practical Recommendations for Gradient-based Training of Deep Architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer.

Bengio, Y., Lee, D.-H., Bornschein, J., Mesnard, T., and Lin, Z. (2015). Towards Biologically Plausible Deep Learning. *arXiv preprint arXiv:1502.04156*.

Bengio, Y., Mesnard, T., Fischer, A., Zhang, S., and Wu, Y. (2017). STDP-compatible Approximation of Backpropagation in an Energy-based Model. *Neural computation*.

Bouckaert, R. R. (2004). Estimating Replicability of Classifier Learning Experiments. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Bouckaert, R. R. and Frank, E. (2004). Evaluating the Replicability of Significance Tests for Comparing Learning Algorithms. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 3–12. Springer.

Boulesteix, A.-L., Lauer, S., and Eugster, M. J. (2013). A Plea for Neutral Comparison Studies in Computational Sciences. *PloS one*, 8(4):e61562.

BIBLIOGRAPHY

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. *arXiv preprint arXiv:1606.01540*.

Brunskill, E. and Li, L. (2014). PAC-inspired Option Discovery in Lifelong Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 316–324.

Buckheit, J. B. and Donoho, D. L. (1995). Wavelab and Reproducible Research. In *Wavelets and statistics*, pages 55–81. Springer.

Cacioppo, J. T., Kaplan, R. M., Krosnick, J. A., Olds, J. L., and Dean, H. (2015). Social, Behavioral, and Economic Sciences Perspectives on Robust and Reliable Science.

Calandriello, D., Lazaric, A., and Restelli, M. (2014). Sparse Multi-Task Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 819–827.

Caruana, R. (1998). Multitask Learning. In *Learning to Learn*, pages 95–133. Springer.

Choi, J. and eung Kim, K. (2012). Nonparametric Bayesian Inverse Reinforcement Learning for Multiple Reward Functions. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, pages 305–313. Curran Associates, Inc.

Christiano, P., Shah, Z., Mordatch, I., Schneider, J., Blackwell, T., Tobin, J., Abbeel, P., and Zaremba, W. (2016). Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model. *arXiv preprint arXiv:1610.03518*.

Collins, F. S. and Tabak, L. A. (2014). NIH Plans to Enhance Reproducibility. *Nature*, 505(7485):612.

Daniel, C., Neumann, G., and Peters, J. R. (2012). Hierarchical Relative Entropy Policy Search. In *International Conference on Artificial Intelligence and Statistics*, pages 273–281.

Dayan, P. and Hinton, G. E. (1993). Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 271–278.

BIBLIOGRAPHY

Dietterich, T. G. (2000). Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

Dong, H., Yu, S., Wu, C., and Guo, Y. (2017). Semantic Image Synthesis via Adversarial Learning. In *The IEEE International Conference on Computer Vision (ICCV)*.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking Deep Reinforcement Learning for Continuous Control. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Efron, B. and Tibshirani, R. J. (1994). *An Introduction to the Bootstrap*. CRC press.

Fernández, F. and Veloso, M. (2013). Learning Domain Structure through Probabilistic Policy Reuse in Reinforcement Learning. *Progress in Artificial Intelligence*, 2(1):13–27.

Finn, C., Yu, T., Fu, J., Abbeel, P., and Levine, S. (2017). Generalizing Skills with Semi-Supervised Reinforcement Learning. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Florian, R. V. (2007). Reinforcement Learning through Modulation of Spike-Timing-Dependent Synaptic Plasticity. *Neural Computation*, 19(6):1468–1502.

François-Lavet, V., Fonteneau, R., and Ernst, D. (2015). How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies. *arXiv preprint arXiv:1512.02011*.

Glorot, X. and Bengio, Y. (2010). Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

Glynn, P. W. (1987). Likelilood Ratio Gradient Estimation: An Overview. In *Proceedings of the 19th Conference on Winter Simulation*, pages 366–375. ACM.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.

BIBLIOGRAPHY

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. (2017). Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., Schölkopf, B., and Levine, S. (2017). Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, pages 3846–3855. Curran Associates, Inc.

Gupta, A., Devin, C., Liu, Y., Abbeel, P., and Levine, S. (2017). Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Hausman, K., Chebotar, Y., Schaal, S., Sukhatme, G., and Lim, J. J. (2017). Multi-Modal Imitation Learning from Unstructured Demonstrations using Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*.

Hawasly, M. and Ramamoorthy, S. (2013). Lifelong Transfer Learning with an Option Hierarchy. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1341–1346.

Henderson, P., Chang, W.-D., Bacon, P.-L., Meger, D., Pineau, J., and Precup, D. (2018a). OptionGAN: Learning Joint Reward-Policy Options using Generative Adversarial Inverse Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Henderson, P., Chang, W.-D., Shkurti, F., Hansen, J., Meger, D., and Dudek, G. (2017). Benchmark Environments for Multitask Learning in Continuous Domains. *Lifelong Learning: A Reinforcement Learning Approach Workshop at ICML*.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018b). Deep Reinforcement Learning that Matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

BIBLIOGRAPHY

Hesse, C., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2017). OpenAI Baselines.

Higuera, J. C. G., Meger, D., and Dudek, G. (2017). Adapting Learned Robotics Behaviours through Policy Adjustment. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Ho, J. and Ermon, S. (2016). Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573.

Hofer, L. and Gimbert, H. (2016). Online Reinforcement Learning for Real-Time Exploration in Continuous State and Action Markov Decision Processes. *arXiv preprint arXiv:1612.03780*.

Hu, W. and Tan, Y. (2017). Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *arXiv preprint arXiv:1702.05983*.

Islam, R., Henderson, P., Gomrokchi, M., and Precup, D. (2017). Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. *Reproducibility in Machine Learning Workshop at ICML*.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2017). Reinforcement Learning with Unsupervised Auxiliary Tasks. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Kenall, A., Shanahan, D. R., Goodman, L., Bal, L., Flintoft, L., Edmunds, S., and Shipley, T. (2015). Better Reporting for Better Research: A Checklist for Reproducibility.

Kingma, D. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 14, pages 1137–1145.

Krishnan, S., Garg, A., Liaw, R., Miller, L., Pokorny, F. T., and Goldberg, K. (2016). HIRL: Hierarchical Inverse Reinforcement Learning for Long-Horizon Tasks with Delayed Rewards. *arXiv preprint arXiv:1604.06508*.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient Backprop. In *Neural Networks: Tricks of the Trade*. Springer.

Li, J., Monroe, W., Shi, T., Jean, S., Ritter, A., and Jurafsky, D. (2017a). Adversarial Learning for Neural Dialogue Generation. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Li, Y., Song, J., and Ermon, S. (2017b). InfoGAIL: Interpretable Imitation Learning from Visual Demonstrations. In *Advances in Neural Information Processing Systems*, pages 3815–3825.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous Control with Deep Reinforcement Learning. *arXiv preprint arXiv:1509.02971*.

Liu, Y. and Yao, X. (2002). Learning and Evolution by Minimization of Mutual Information. In *International Conference on Parallel Problem Solving from Nature*, pages 495–504. Springer.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018). Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research*, 61:523–562.

Mandel, T., Liu, Y.-E., Brunskill, E., and Popovic, Z. (2016). Offline Evaluation of Online Reinforcement Learning Algorithms. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation*, 24:109–165.

Melis, G., Dyer, C., and Blunsom, P. (2018). On the State of the Art of Evaluation in Neural Language Models. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Merel, J., Tassa, Y., Srinivasan, S., Lemmon, J., Wang, Z., Wayne, G., and Heess, N. (2017). Learning Human Behaviors from Motion Capture by Adversarial Imitation. *arXiv preprint arXiv:1707.02201*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. In *Deep Learning Workshop, NIPS*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533.

Mujika, A. (2016). Multi-Task Learning with Deep Model Based Reinforcement Learning. *arXiv preprint arXiv:1611.01457*.

Murugesan, K., Liu, H., Carbonell, J., and Yang, Y. (2016). Adaptive Smoothed Online Multi-Task Learning. In *Advances in Neural Information Processing Systems*, pages 4296–4304.

Nadeau, C. and Bengio, Y. (2000). Inference for the Generalization Error. In *Advances in Neural Information Processing Systems*.

Ng, A. Y. and Russell, S. J. (2000). Algorithms for Inverse Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 663–670, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Parisotto, E., Ba, J., and Salakhutdinov, R. (2015). Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2018). Parameter Space Noise for Exploration. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Precup, D. (2000). *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst.

Rajeswaran, A., Lowrey, K., Todorov, E. V., and Kakade, S. M. (2017). Towards Generalization and Simplicity in Continuous Control. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, pages 6550–6561. Curran Associates, Inc.

Rivest, F., Bengio, Y., and Kalaska, J. (2005). Brain Inspired Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 1129–1136.

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2016). Policy Distillation. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. (2017). Sim-to-Real Robot Learning from Pixels with Progressive Nets. *Proceedings of the 1st Annual Conference on Robot Learning (CoRL)*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust Region Policy Optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

Sermanet, P., Xu, K., and Levine, S. (2017). Unsupervised Perceptual Rewards for Imitation Learning. In *Robotics: Science and Systems*.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Silva, V. d. N. and Chaimowicz, L. (2017). MOBA: A New Arena for Game AI. *arXiv preprint arXiv:1705.10443*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587):484–489.

Stadie, B. C., Abbeel, P., and Sutskever, I. (2017). Third-Person Imitation Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Stodden, V., Leisch, F., and Peng, R. D. (2014). *Implementing Reproducible Research*. CRC Press.

Stolle, M. and Precup, D. (2002). Learning Options in Reinforcement Learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 212–223. Springer.

Subramanian, S., Rajeswar, S., Dutil, F., Pal, C., and Courville, A. C. (2017). Adversarial Generation of Natural Language. In *Proceedings of the 2nd Workshop on Representation Learning for NLP, ACL*.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A Framework For Temporal Abstraction in Reinforcement Learning. *Artificial intelligence*, 112(1-2):181–211.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. (2016). Value Iteration Networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162.

Taylor, M. E. and Stone, P. (2009). Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

Teh, Y., Bapst, V., Pascanu, R., Heess, N., Quan, J., Kirkpatrick, J., Czarnecki, W. M., and Hadsell, R. (2017). Distral: Robust Multitask Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 4497–4507.

Thomas, P. (2014). Bias in Natural Actor-Critic Algorithms. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 441–448.

Thomas, P. and Brunskill, E. (2016). Data-Efficient Off-Policy Policy Evaluation for Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2139–2148.

Thomas, P. S., Theocharous, G., and Ghavamzadeh, M. (2015). High-Confidence Off-Policy Evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Thrun, S. (1995). Lifelong Learning: A Case Study. Technical Report CMU-CS-95-208, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

Thrun, S. (1996). Is Learning The n-th Thing Any Easier Than Learning The First? In *Advances in Neural Information Processing Systems*, pages 640–646. The MIT Press.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A Physics Engine for Model-based Control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033.

Tufféry, S. (2011). *Data Mining and Statistics for Decision Making*, volume 2. Wiley Chichester.

Tulyakov, S., Liu, M.-Y., Yang, X., and Kautz, J. (2017). MoCoGAN: Decomposing Motion and Content for Video Generation. *arXiv preprint arXiv:1707.04993*.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2094–2100.

van Hasselt, H. P., Guez, A., Hessel, M., Mnih, V., and Silver, D. (2016). Learning Values Across Many Orders of Magnitude. In *Advances in Neural Information Processing Systems*, pages 4287–4295.

van Seijen, H., Fatemi, M., Romoff, J., Laroche, R., Barnes, T., and Tsang, J. (2017). Hybrid Reward Architecture for Reinforcement Learning. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Vaughan, R. and Wawerla, J. (2012). Publishing Identifiable Experiment Code and Configuration is Important, Good and Easy. *arXiv preprint arXiv:1204.2235*.

Vincent, P., de Brébisson, A., and Bouthillier, X. (2015). Efficient exact gradient update for training deep networks with very large sparse targets. In *Advances in Neural Information Processing Systems*, pages 1108–1116.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., et al. (2017). StarCraft II: A New Challenge for Reinforcement Learning. *arXiv preprint arXiv:1708.04782*.

Wagstaff, K. (2012). Machine Learning that Matters. In *Proceedings of the International Coference on International Conference on Machine Learning (ICML)*.

Wang, Z., Merel, J., Reed, S., Wayne, G., de Freitas, N., and Heess, N. (2017). Robust Imitation of Diverse Behaviors. *arXiv preprint arXiv:1707.02747*.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge.

Whiteson, S., Tanner, B., Taylor, M. E., and Stone, P. (2011). Protecting Against Evaluation Overfitting in Empirical Reinforcement Learning. In *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, pages 120–127.

Wilcox, R. (2005). Kolmogorov–Smirnov test. *Encyclopedia of Biostatistics*.

Williams, R. J. (1992). Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning. *Machine learning*, 8(3-4):229–256.

Wilson, A., Fern, A., Ray, S., and Tadepalli, P. (2007). Multi-Task Reinforcement Learning: A Hierarchical Bayesian Approach. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1015–1022. ACM.

Wu, Y., Mansimov, E., Liao, S., Grosse, R., and Ba, J. (2017). Scalable Trust-Region Method for Deep Reinforcement Learning using Kronecker-factored Approximation. *arXiv preprint:1708.05144*.

Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical Evaluation of Rectified Activations in Convolutional Network. *Deep Learning Workshop, ICML*.

Yang, Z., Merrick, K., Abbass, H., and Jin, L. (2017). Multi-Task Deep Reinforcement Learning for Continuous Action Control. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3301–3307.

Yuan, K.-H. and Hayashi, K. (2003). Bootstrap Approach to Inference and Power Analysis Based on Three Test Statistics for Covariance Structure Models. *British Journal of Mathematical and Statistical Psychology*, 56(1):93–110.

Ziebart, B. D., Maas, A., Bagnell, A. J., and Dey, A. K. (2008). Maximum Entropy Inverse Reinforcement Learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, pages 1433–1438. AAAI Press.

A

Supplemental Material : Reproducibility

In this supplemental material, we include a detailed review of experiment configurations of related work with policy gradient methods in continuous control MuJoCo (Todorov et al., 2012) environment tasks from OpenAI Gym (Brockman et al., 2016). We include a detailed list of the hyperparameters and reported metrics typically used in policy gradient literature in deep RL. We also include all our experimental results, with baseline algorithms DDPG (Lillicrap et al., 2015), TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017) and ACKTR (Wu et al., 2017)) as discussed in the paper. Our experimental results include figures with different hyperparameters (network architectures, activation functions) to highlight the differences this can have across algorithms and environments. Finally, as discussed in the paper, we include discussion of significance metrics and show how these metrics can be useful for evaluating deep RL algorithms.

A.1 Literature Reviews

A.1.1 Hyperparameters

In this section, we include a list of hyperparameters that are reported in related literature, as shown in figure A.1. Our analysis shows that often there is no consistency in the type of network architectures and activation functions that are used in related literature. As shown in the paper and from our experimental results in later sections, we find, however, that

these hyperparameters can have a significant effect in the performance of algorithms across benchmark environments typically used.

Table A.1: Evaluation Hyperparameters of baseline algorithms reported in related literature

Related Work (Algorithm)	**Policy Network**	**Policy Network Activation**	**Value Network**	**Value Network Activation**	**Reward Scaling**	**Batch Size**
DDPG	64x64	ReLU	64x64	ReLU	1.0	128
TRPO	64x64	TanH	64x64	TanH	-	5k
PPO	64x64	TanH	64x64	TanH	-	2048
ACKTR	64x64	TanH	64x64	ELU	-	2500
Q-Prop (DDPG)	100x50x25	TanH	100x100	ReLU	0.1	64
Q-Prop (TRPO)	100x50x25	TanH	100x100	ReLU	-	5k
IPG (TRPO)	100x50x25	TanH	100x100	ReLU	-	10k
Param Noise (DDPG)	64x64	ReLU	64x64	ReLU	-	128
Param Noise (TRPO)	64x64	TanH	64x64	TanH	-	5k
Benchmarking (DDPG)	400x300	ReLU	400x300	ReLU	0.1	64
Benchmarking (TRPO)	100x50x25	TanH	100x50x25	TanH	-	25k

A.1.2 Reported Results on Benchmarked Environments

We then demonstrate how experimental reported results, on two different environments (HalfCheetah-v1 and Hopper-v1) can vary across different related work that uses these algorithms for baseline comparison. We further show the results we get, using the same hyperparameter configuration, but using two different codebase implementations (note that

these implementations are often used as baseline codebase to develop algorithms). We highlight that, depending on the codebase used, experimental results can vary significantly.

Table A.2: Comparison with Related Reported Results with Hopper Environment

Metric	rllab	QProp	IPG	TRPO	Our Results (rllab)	Our Results (Baselines)
Number of Iterations	500	500	500	500	500	500
Average Return	1183	-	-	-	2021	2965
Max Average Return	-	2486		3669	3229	3034

Table A.3: Comparison with Related Reported Results with HalfCheetah Environment

Metric	rllab	QProp	IPG	TRPO	Our Results (rllab)	Our Results (Baselines)
Number of Iterations	500	500	500	500	500	500
Average Return	1914	-	-	-	3576	1046
Max Average Return	-	4734	2889	4855	5197	1046

Work	Number of Trials
(Mnih et al., 2016)	top-5
(Schulman et al., 2017)	3-9
(Duan et al., 2016)	5 (5)
(Gu et al., 2017)	3
(Lillicrap et al., 2015)	5
(Schulman et al., 2015)	5
(Wu et al., 2017)	top-2, top-3

Table A.4: Number of trials reported during evaluation in various works.

A.1.3 Reported Evaluation Metrics in Related Work

In Table A.5 we show the evaluation metrics, and reported results in further details across related work.

Table A.5: Reported Evaluation Metrics of baseline algorithms in related literature

Related Work (Algorithm)	**Environments**	**Timesteps or Episodes or Iterations**	**Evaluation Metrics**		
			Average Return	**Max Return**	**Std Error**
PPO	HalfCheetah Hopper	1M	~1800 ~2200	-	-
ACKTR	HalfCheetah Hopper	1M	~2400 ~3500	-	-
Q-Prop (DDPG)	HalfCheetah Hopper	6k (eps)	~6000 -	7490 2604	- -
Q-Prop (TRPO)	HalfCheetah Hopper	5k (timesteps)	~4000 -	4734 2486	- -
IPG (TRPO)	HalfCheetah Hopper	10k (eps)	~3000 -	2889	- -
Param Noise (DDPG)	HalfCheetah Hopper	1M	~1800 ~500	- -	- -
Param Noise (TRPO)	HalfCheetah Hopper	1M	~3900 ~2400	- -	- -
Benchmarking (DDPG)	HalfCheetah Hopper	500 iters (25k eps)	~2148 ~267	- -	702 43
Benchmarking (TRPO)	HalfCheetah Hopper	500 iters (925k eps)	~1914 ~1183	- -	150 120

A.2 Experimental Setup

In this section, we show detailed analysis of our experimental results, using same hyper-parameter configurations used in related work. Experimental results are included for the OpenAI Gym (Brockman et al., 2016) Hopper-v1 and HalfCheetah-v1 environments, using the policy gradient algorithms including DDPG, TRPO, PPO and ACKTR. Our exper-

iments are done using the available codebase from OpenAI rllab (Duan et al., 2016) and OpenAI Baselines. Each of our experiments are performed over 5 experimental trials with different random seeds, and results averaged over all trials. Unless explicitly specified as otherwise (such as in hyperparameter modifications where we alter a hyperparameter under investigation), hyperparameters were as follows.

- DDPG
  - Policy Network: (64, relu, 64, relu, tanh); Q Network (64, relu, 64, relu, linear)
  - Normalized observations with running mean filter
  - Actor LR: $1e-4$; Critic LR: $1e-3$
  - Reward Scale: 1.0
  - Noise type: O-U 0.2
  - Soft target update $\tau = .01$
  - $\gamma = 0.995$
  - batch size = 128
  - Critic L2 reg $1e-2$
- PPO
  - Policy Network: (64, tanh, 64, tanh, Linear) + Standard Deviation variable; Value Network (64, tanh, 64, tanh, linear)
  - Normalized observations with running mean filter
  - Timesteps per batch 2048
  - clip param = 0.2
  - entropy coeff = 0.0
  - Optimizer epochs per iteration = 10
  - Optimizer step size $3e-4$
  - Optimizer batch size 64
  - Discount $\gamma = 0.995$, GAE $\lambda = 0.97$
  - learning rate schedule is constant

- TRPO
  - Policy Network: (64, tanh, 64, tanh, Linear) + Standard Deviation variable; Value Network (64, tanh, 64, tanh, linear)
  - Normalized observations with running mean filter
  - Timesteps per batch 5000
  - max KL=0.01
  - Conjugate gradient iterations = 20
  - CG damping = 0.1
  - VF Iterations = 5
  - VF Batch Size = 64
  - VF Step Size = $1e-3$
  - entropy coeff = 0.0
  - Discount $\gamma = 0.995$, GAE $\lambda = 0.97$
- ACKTR
  - Policy Network: (64, tanh, 64, tanh, Linear) + Standard Deviation variable; Value Network (64, elu, 64, elu, linear)
  - Normalized observations with running mean filter
  - Timesteps per batch 2500
  - desired KL = .002
  - Discount $\gamma = 0.995$, GAE $\lambda = 0.97$

A.2.1 Modifications to Baseline Implementations

To ensure fairness of comparison, we make several modifications to the existing implementations. First, we change evaluation in DDPG (Plappert et al., 2018) such that during evaluation at the end of an epoch, 10 full trajectories are evaluated. In the current implementation, only a partial trajectory is evaluated immediately after training such that a full trajectory will be evaluated across several different policies, this corresponds more closely

to the online view of evaluation, while we take a policy optimization view when evaluating algorithms.

A.2.2 Hyperparameters: Network Architecture

Below, we examine the significance of the network configurations used for the non-linear function approximators in policy gradient methods. Several related work have used different sets of network configurations (network sizes and activation functions). We use the reported network configurations from other works, and demonstrate the significance of careful fine tuning that is required. We demonstrate results using the network activation functions, ReLU, TanH and Leaky ReLU, where most papers use ReLU and TanH as activation functions without detailed reporting of the effect of these activation functions. We analyze the significance of using different activations in the policy and action value networks. Previously, we included a detailed table showing average reward with standard error obtained for each of the hyperparameter configurations. In the results below, we show detailed results of how each of these policy gradient algorithms are affected by the choice of the network configuration.

A.2.3 Proximal Policy Optimization (PPO)

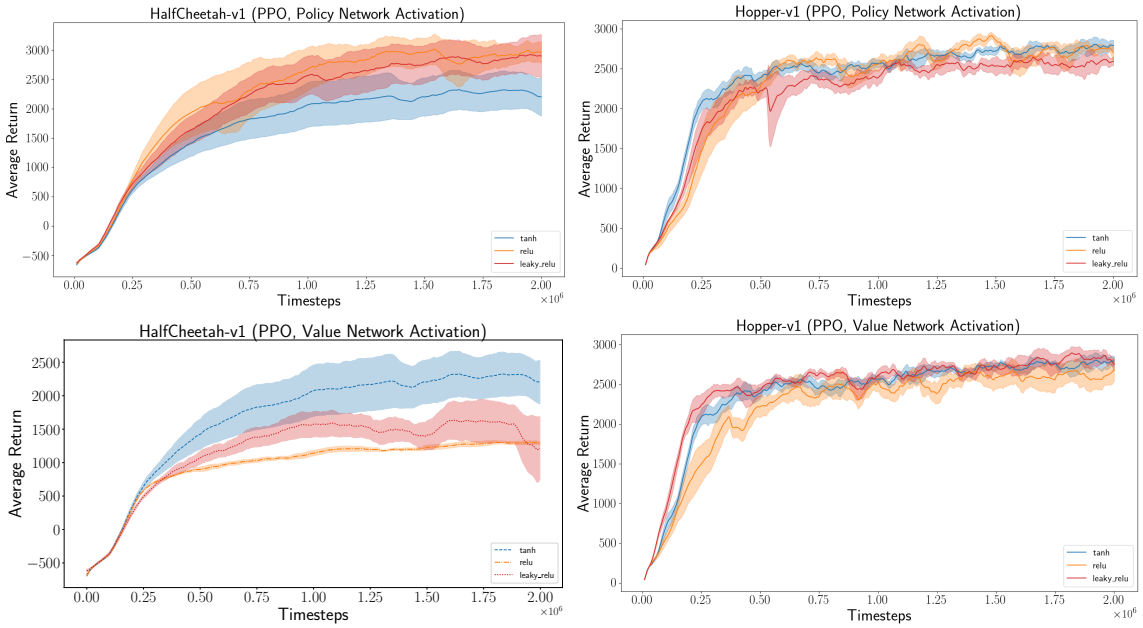

Figure A.1: PPO Policy and Value Network activation

Experiment results in Figure A.1, A.2, and A.3 in this section show the effect of the policy network structures and activation functions in the Proximal Policy Optimization (PPO) algorithm.

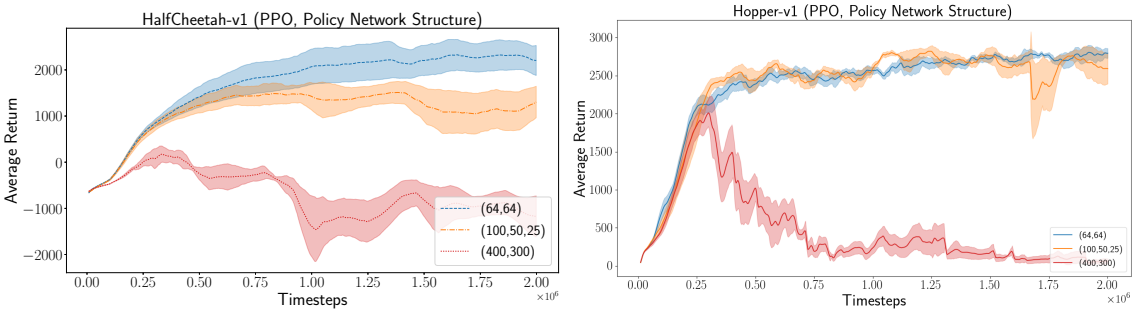

Figure A.2: PPO Policy Network structure

A.2 Experimental Setup

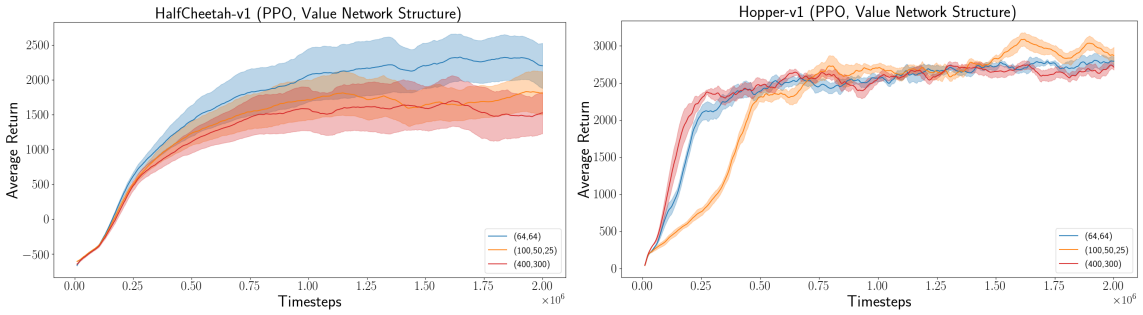

Figure A.3: PPO Value Network structure

A.2.4 Actor Critic using Kronecker-Factored Trust Region (ACKTR)

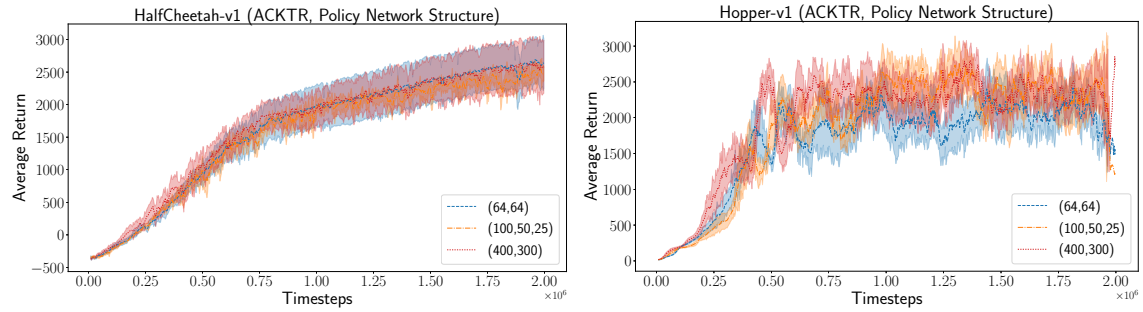

Figure A.4: ACKTR Policy Network structure

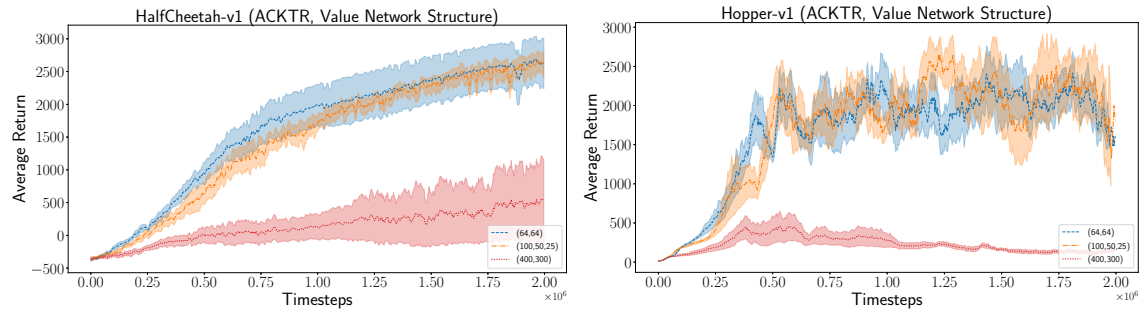

Figure A.5: ACKTR Value Network structure

A.2 Experimental Setup

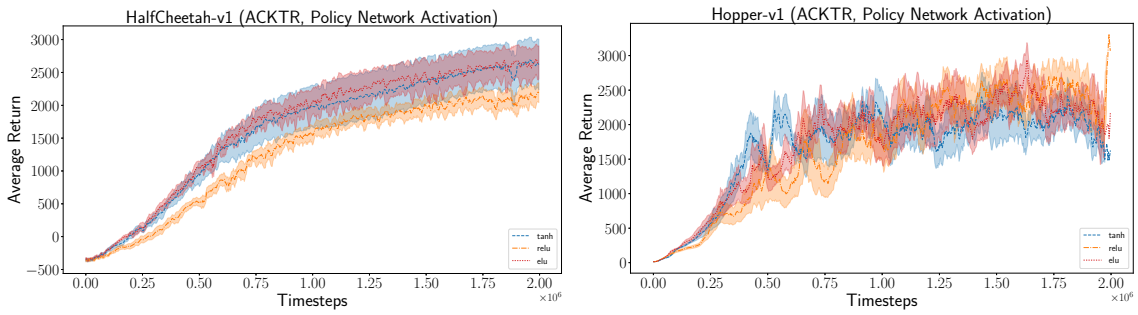

Figure A.6: ACKTR Policy Network Activation

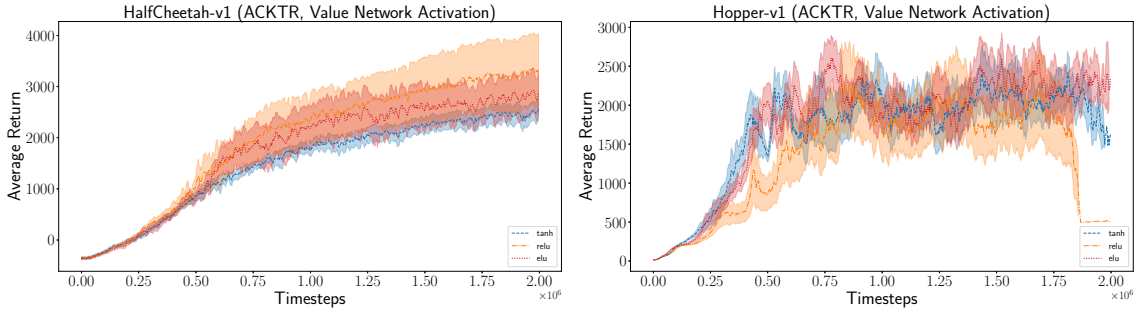

Figure A.7: ACKTR Value Network Activation

We then similarly, show the significance of these hyperparameters in the ACKTR algorithm. Our results show that the value network structure can have a significant effect on the performance of ACKTR algorithm.

A.2.5 Trust Region Policy Optimization (TRPO)

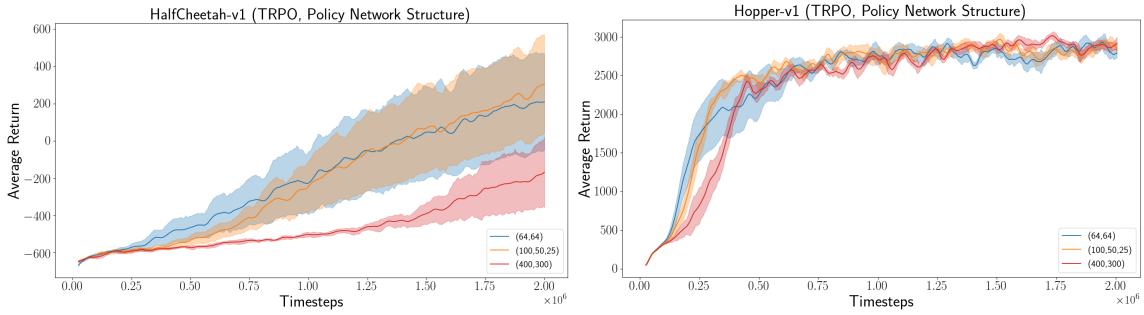

Figure A.8: TRPO Policy Network structure

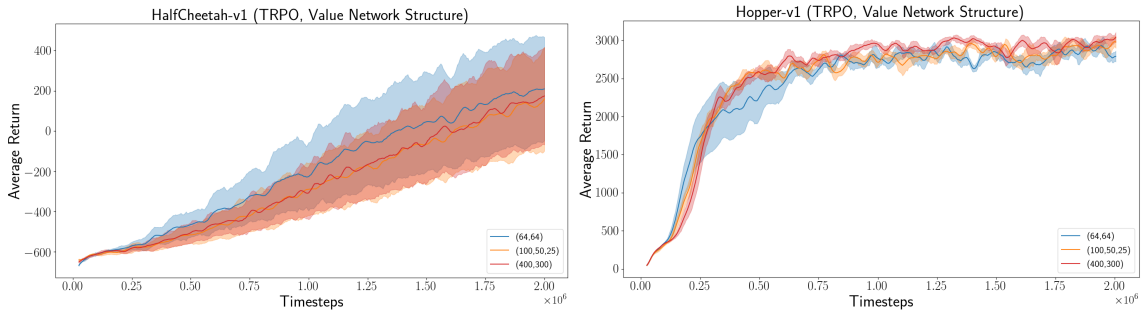

Figure A.9: TRPO Value Network structure

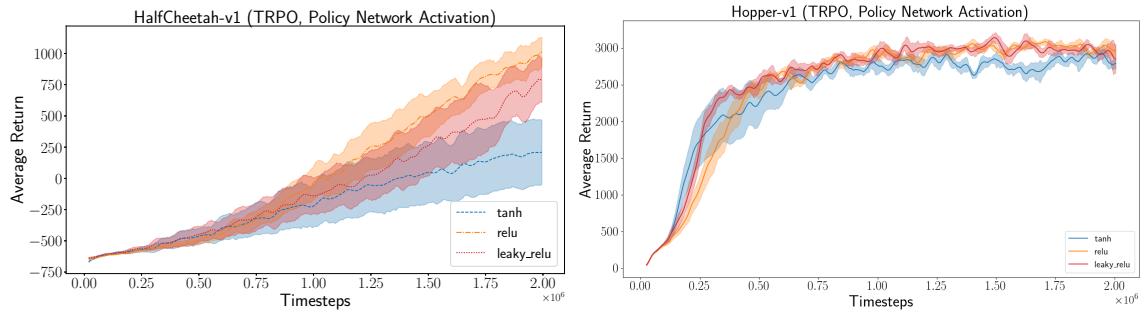

Figure A.10: TRPO Policy and Value Network activation

A.2 Experimental Setup

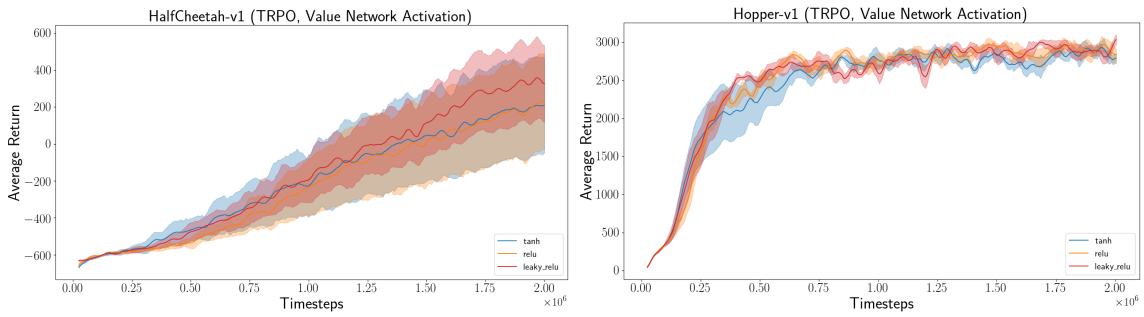

Figure A.11: TRPO Policy and Value Network activation

In Figures A.8, A.9, A.10, and A.11 we show the effects of network structure on the OpenAI baselines implementation of TRPO. In this case, only the policy architecture seems to have a large effect on the performance of the algorithm's ability to learn.

A.2.6 Deep Deterministic Policy Gradient (DDPG)

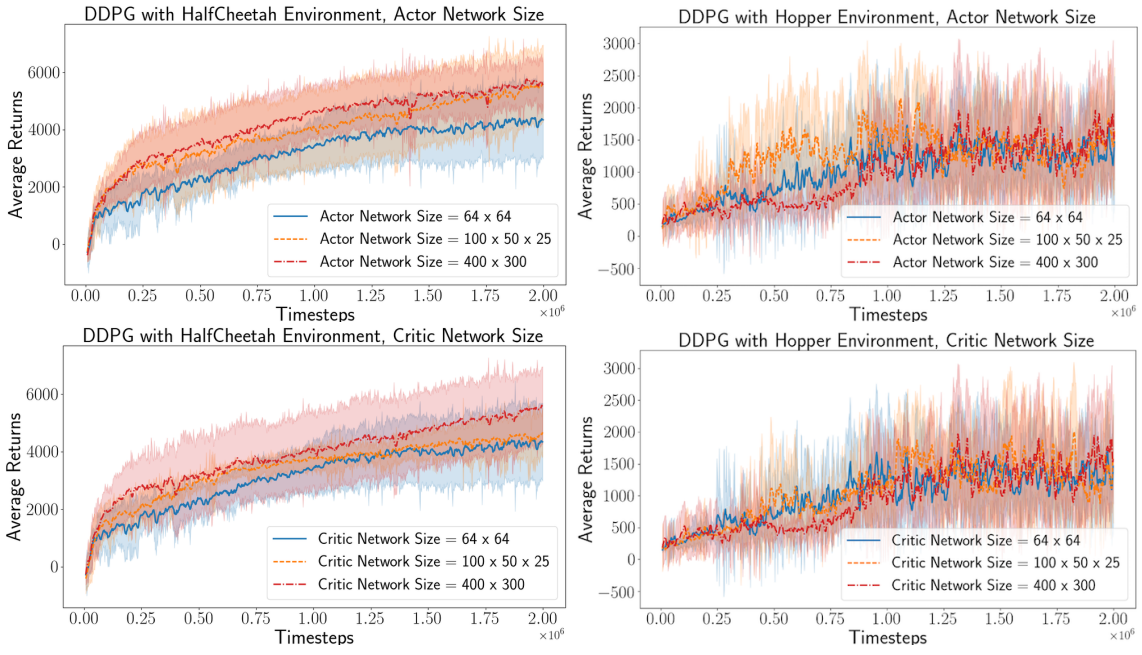

Figure A.12: Policy or Actor Network Architecture experiments for DDPG on HalfCheetah and Hopper Environment

A.2 Experimental Setup

We further analyze the actor and critic network configurations for use in DDPG. As in default configurations, we first use the ReLU activation function for policy networks, and examine the effect of different activations and network sizes for the critic networks. Similarly, keeping critic network configurations under default setting, we also examine the effect of actor network activation functions and network sizes.

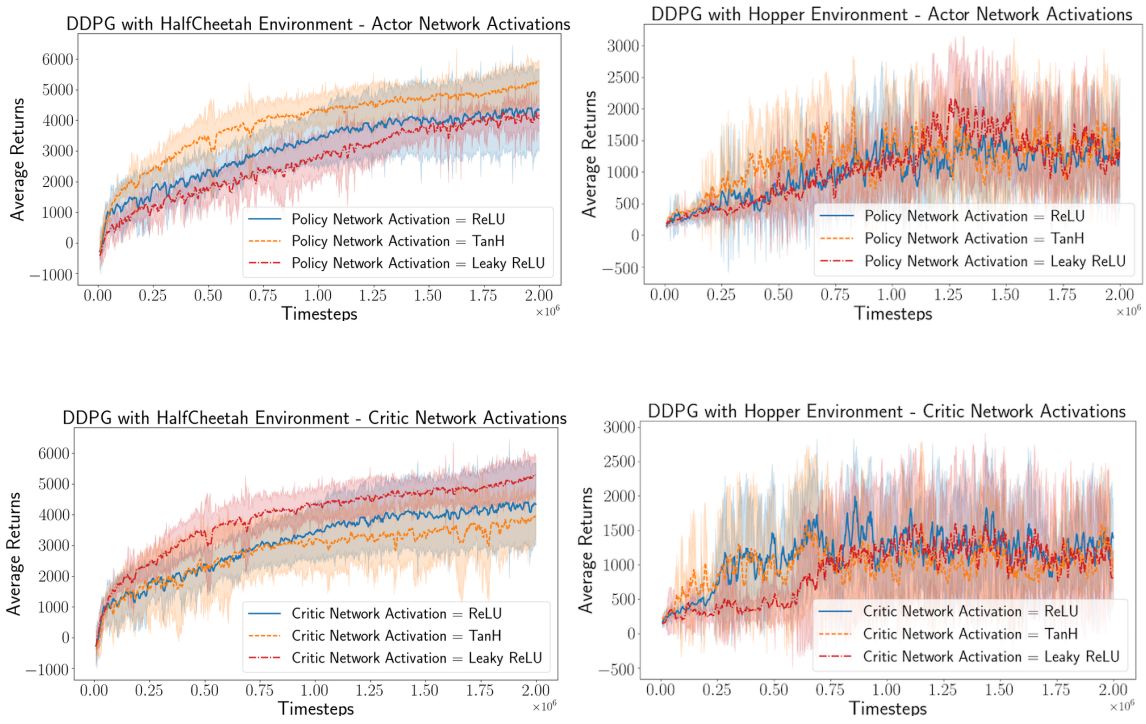

Figure A.13: Significance of Value Function or Critic Network Activations for DDPG on HalfCheetah and Hopper Environment

A.3 Reward Scaling Parameter in DDPG

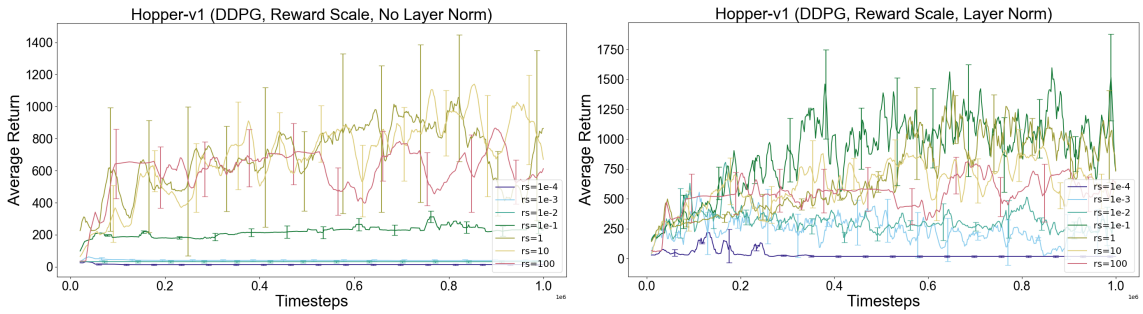

Figure A.14: DDPG reward rescaling on Hopper-v1, with and without layer norm.

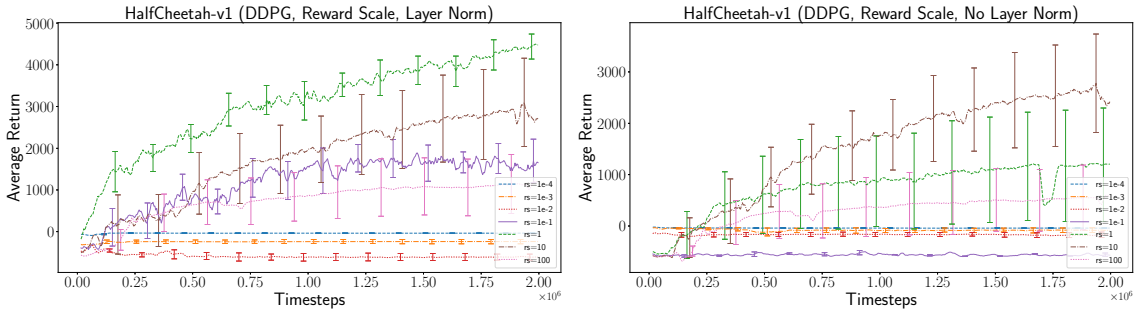

Figure A.15: DDPG reward rescaling on HalfCheetah-v1, with and without layer norm.

Several related work (Gu et al., 2017; Gu et al., 2017; Duan et al., 2016) have often reported that for DDPG the reward scaling parameter often needs to be fine-tuned for stabilizing the performance of DDPG. It can make a significant impact in performance of DDPG based on the choice of environment. We examine several reward scaling parameters and demonstrate the effect this parameter can have on the stability and performance of DDPG, based on the HalfCheetah and Hopper environments. Our experiment results, as demonstrated in Figure A.15 and A.14, show that the reward scaling parameter indeed can have a significant impact on performance. Our results show that, very small or negligible reward scaling parameter can significantly detriment the performance of DDPG across all environments. Furthermore, a scaling parameter of 10 or 1 often performs well. Based on our analysis,

we suggest that every time DDPG is reported as a baseline algorithm for comparison, the reward scaling parameter should be fine-tuned, specific to the algorithm.

A.4 Batch Size in TRPO

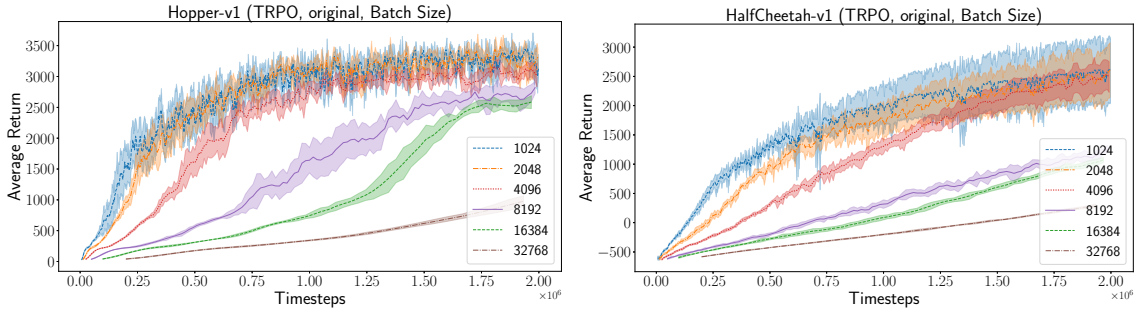

Figure A.16: TRPO (Schulman et al., 2015) original code batch size experiments.

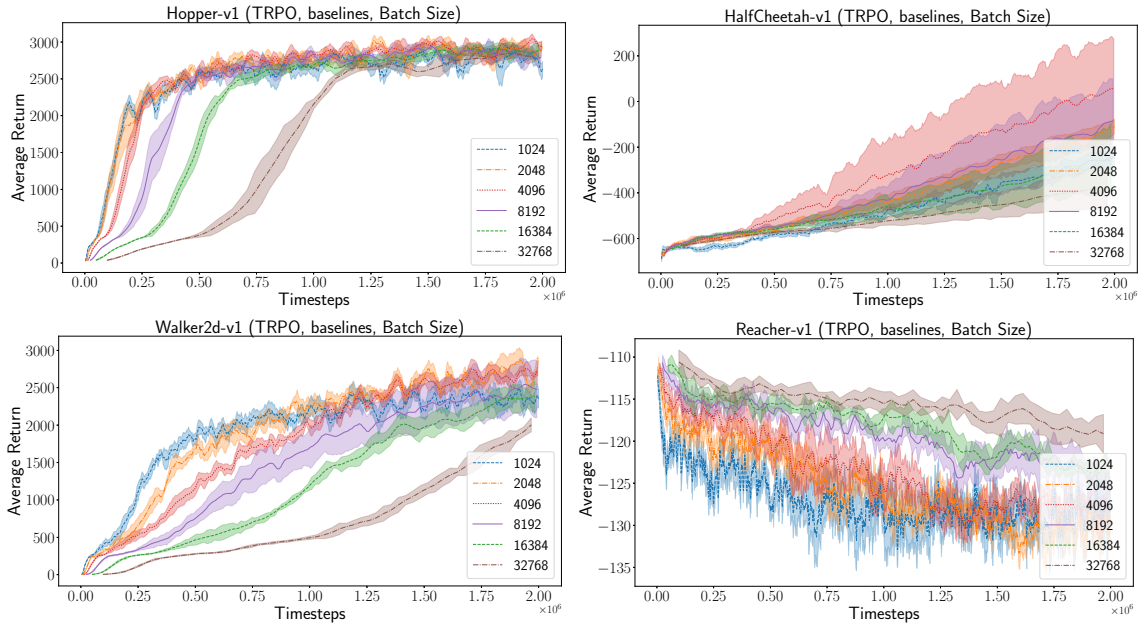

Figure A.17: TRPO (Schulman et al., 2017) baselines code batch size experiments.

We run batch size experiments using the original TRPO code (Schulman et al., 2015) and the OpenAI baselines code (Schulman et al., 2017). These results can be found in Experiment results in Figure A.16 and Figure A.17, show that for both HalfCheetah-v1 and Hopper-v1 environments, a batch size of 1024 for TRPO performs best, while perform degrades consecutively as the batch size is increased.

A.5 Random Seeds

To determine much random seeds can affect results, we run 10 trials total on two environments using the default previously described settings usign the (Gu et al., 2017) implementation of DDPG and the (Duan et al., 2016) version of TRPO. We divide our trials random into 2 partitions and plot them in Figures A.18 and Fig A.19. As can be seen, statistically different distributions can be attained just from the random seeds with the same exact hyperparameters. As we will discuss later, bootstrapping off of the sample can give an idea for how drastic this effect will be, though too small a bootstrap will still not give concrete enough results.

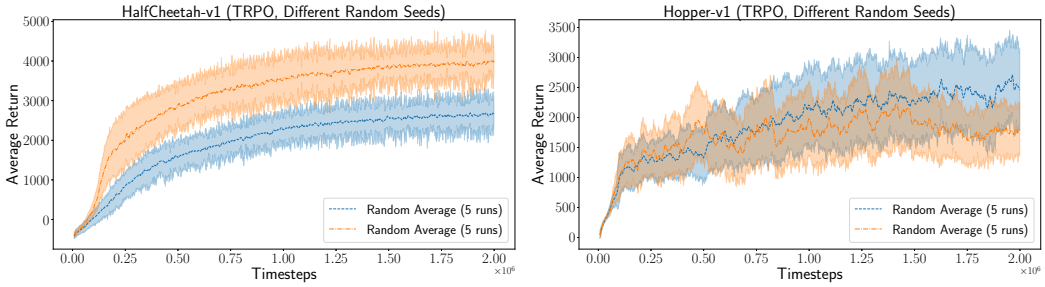

Figure A.18: Two different TRPO experiment runs, with same hyperparameter configurations, averaged over two splits of 5 different random seeds.

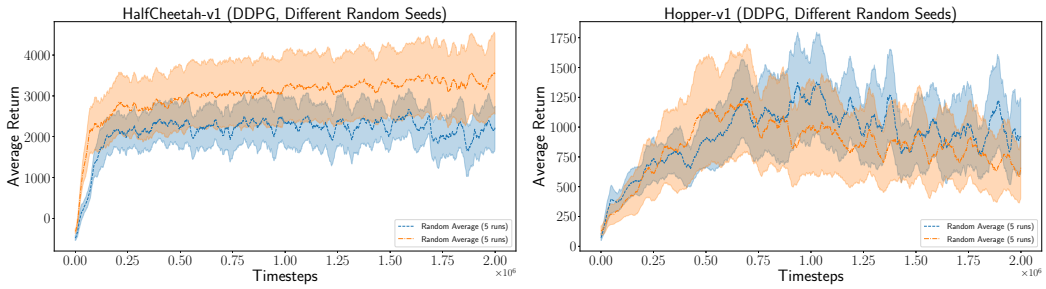

Figure A.19: Two different DDPG experiment runs, with same hyperparameter configurations, averaged over two splits of 5 different random seeds.

A.6 Environments

We previously demonstrated that the performance of policy gradient algorithms can be highly biased based on the choice of the environment. In this section, we include further results examining the impact the choice of environment can have. We show that no single algorithm can perform consistently better in all environments. This is often unlike the results we see with DQN networks in Atari domains, where results can often be demonstrated across a wide range of Atari games. Our results, for example, shows that while TRPO can perform significantly better than other algorithms on the Swimmer environment, it may perform quite poorly n the HalfCheetah environment, and marginally better on the Hopper environment compared to PPO. We demonstrate our results using the OpenAI MuJoCo Gym environments including Hopper, HalfCheetah, Swimmer and Walker environments. It is notable to see the varying performance these algorithms can have even in this small set of environment domains. The choice of reporting algorithm performance results can therefore often be biased based on the algorithm designer's experience with these environments.

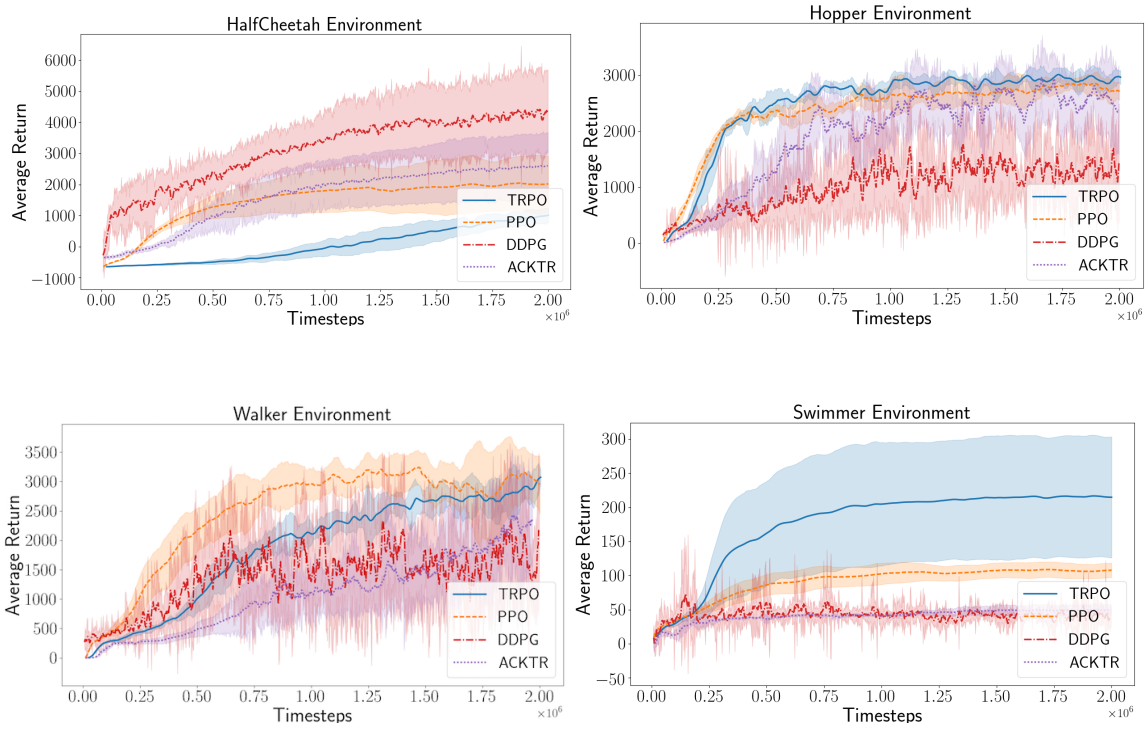

Figure A.20: Comparing Policy Gradients across various environments

A.7 Codebases

We include a detailed analysis of performance comparison, with different network structures and activations, based on the choice of the algorithm implementation codebase.

A.7 Codebases

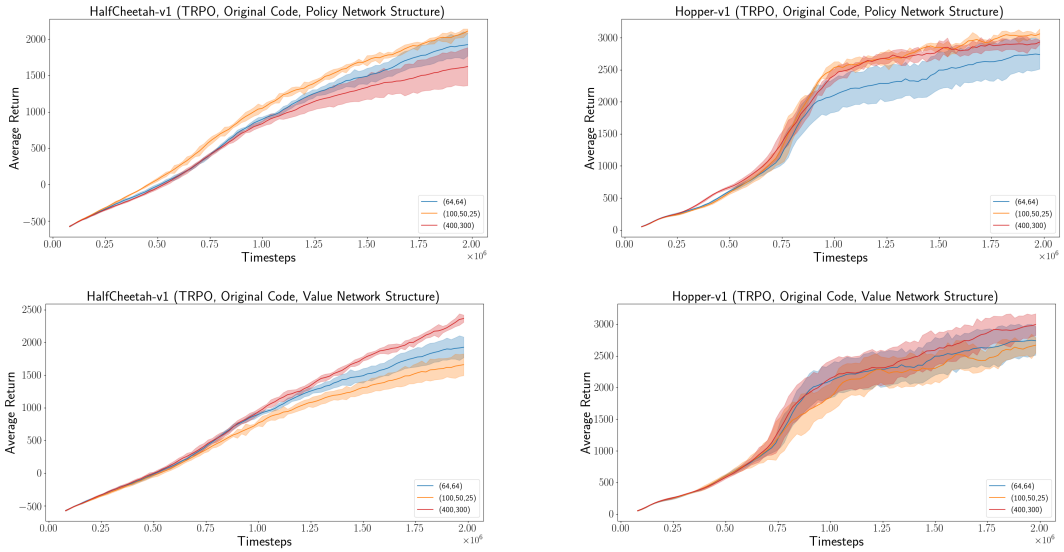

Figure A.21: TRPO Policy and Value Network structure

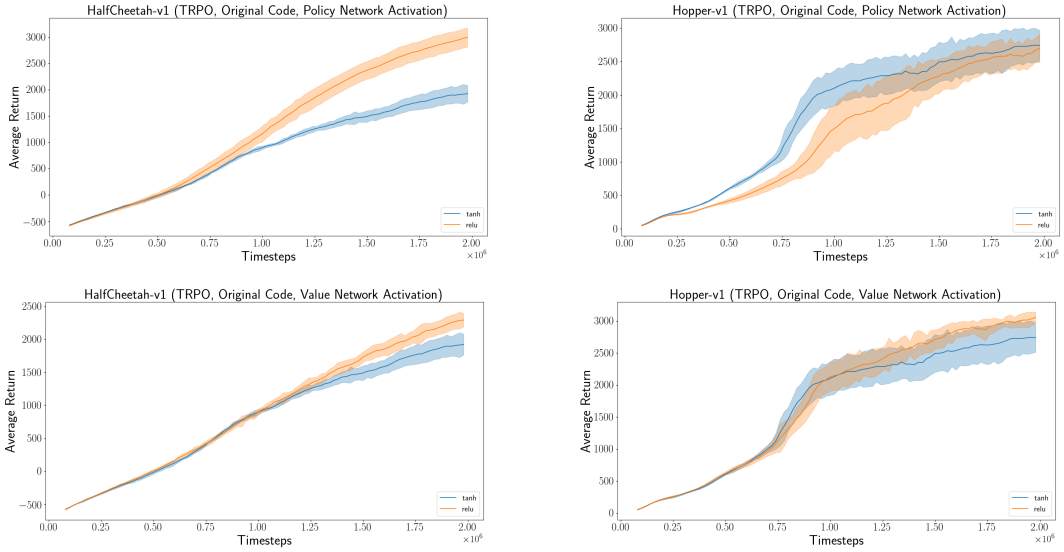

Figure A.22: TRPO Policy and Value Network activations.

A.7 Codebases

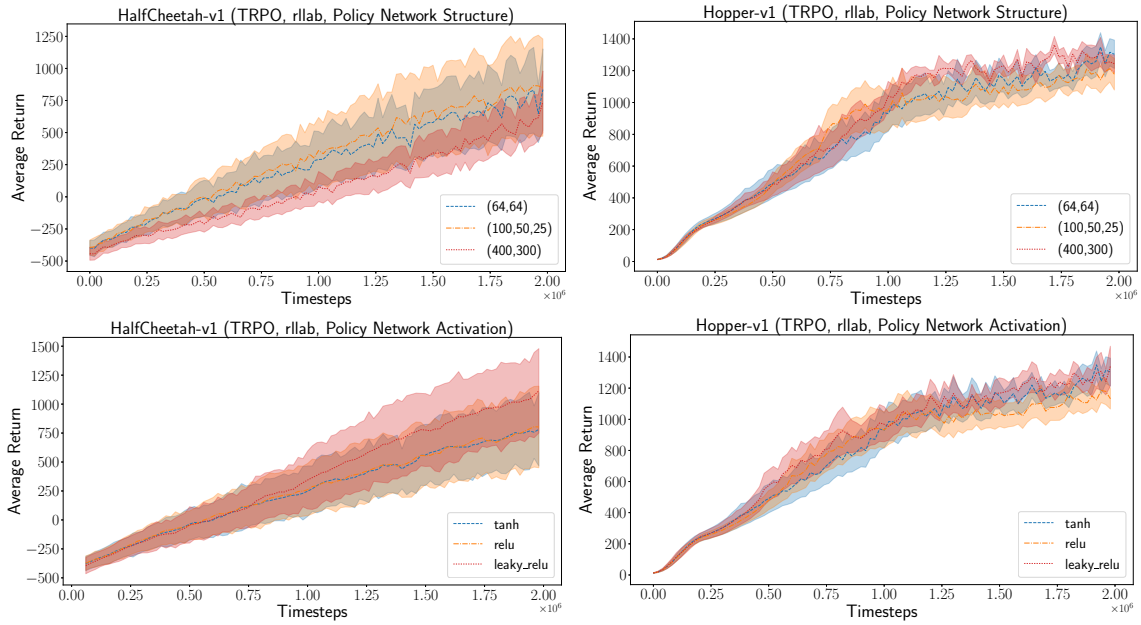

Figure A.23: TRPO rllab Policy Structure and Activation

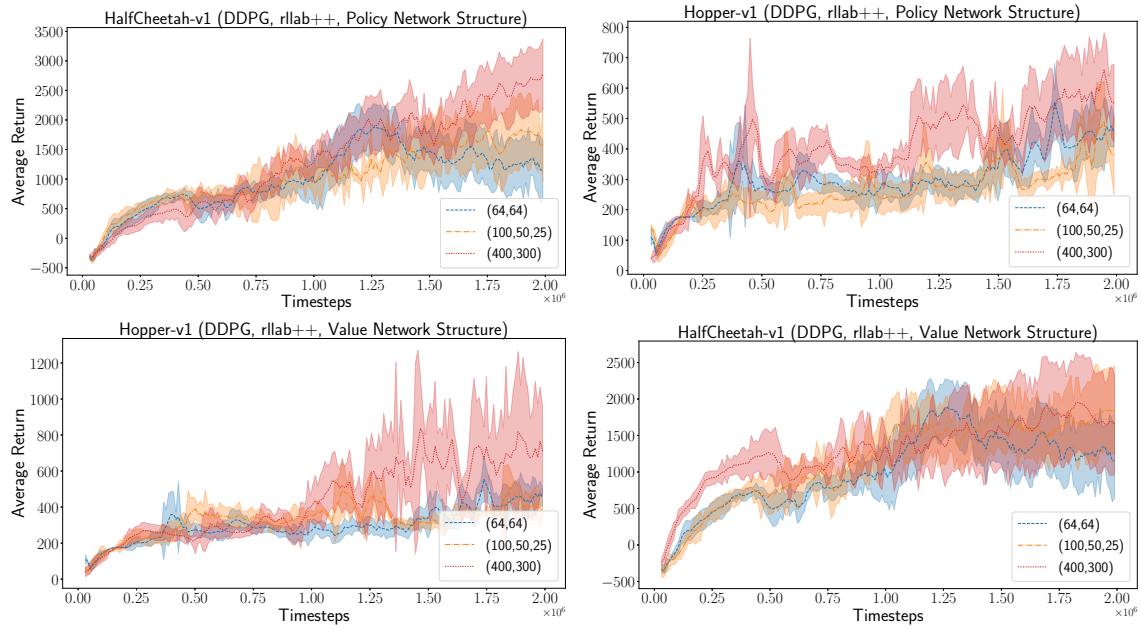

Figure A.24: DDPG rllab++ Policy and Value Network structure

A.7 Codebases

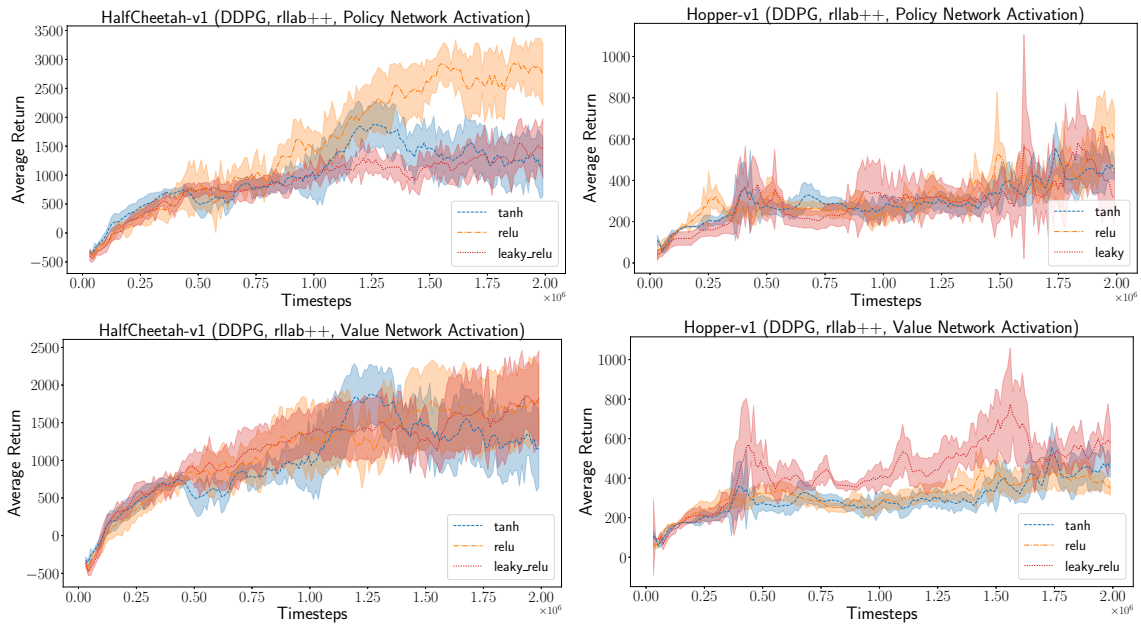

Figure A.25: DDPG rllab++ Policy and Value Network activations.

Similarly, Figures A.26 and A.27 show the same network experiments for DDPG with the Theano implementation of rllab code (Duan et al., 2016).

A.7 Codebases

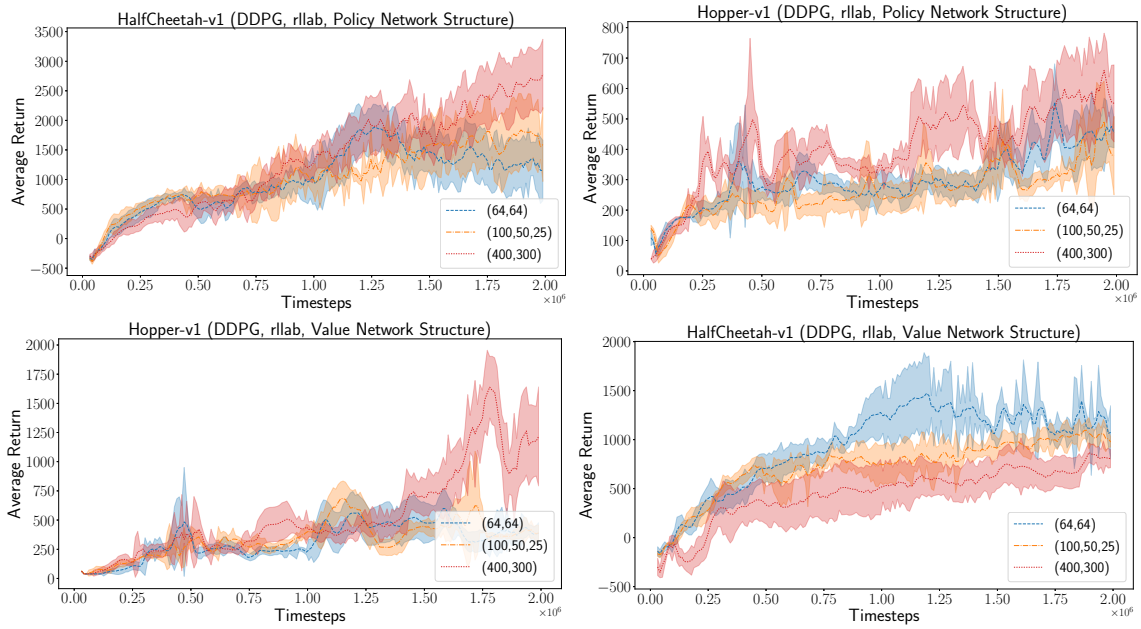

Figure A.26: DDPG rllab Policy and Value Network structure

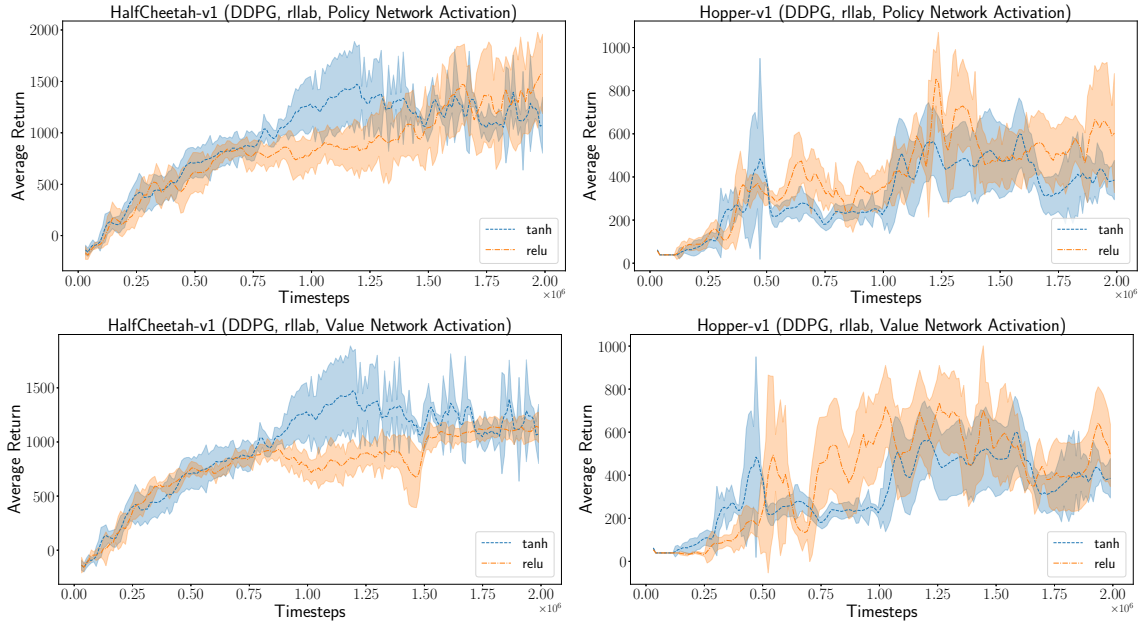

Figure A.27: DDPG rllab Policy and Value Network activations.

A.7 Codebases

Often in related literature, there is different baseline codebase people use for implementation of algorithms. One such example is for the TRPO algorithm. It is a commonly used policy gradient method for continuous control tasks, and there exists several implementations from OpenAI Baselines (Plappert et al., 2018), OpenAI rllab (Duan et al., 2016) and the original TRPO codebase (Schulman et al., 2015). In this section, we perform an analysis of the impact the choice of algorithm codebase can have on the performance. Figures A.21 and A.22 summarizes our results with TRPO policy network and value networks, using the original TRPO codebase from (Schulman et al., 2015). Figure A.23 shows the results using the rllab implementation of TRPO using the same hyperparameters as our default experiments aforementioned. Note, we use a linear function approximator rather than a neural network due to the fact that the Tensorflow implementation of OpenAI rllab does not provide anything else. We note that this is commonly used in other works (Duan et al., 2016; Stadie et al., 2017), but may cause differences in performance. Furthermore, we leave out our value function network experiments due to this.

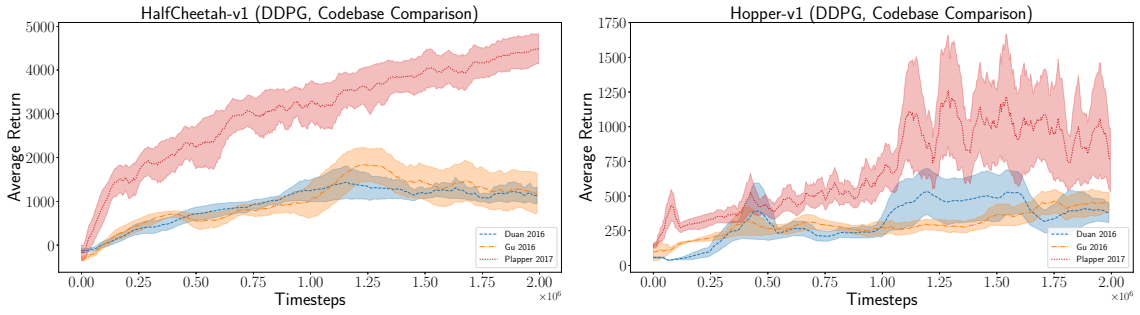

Figure A.28: DDPG codebase comparison using our default set of hyperparameters (as used in other experiments).

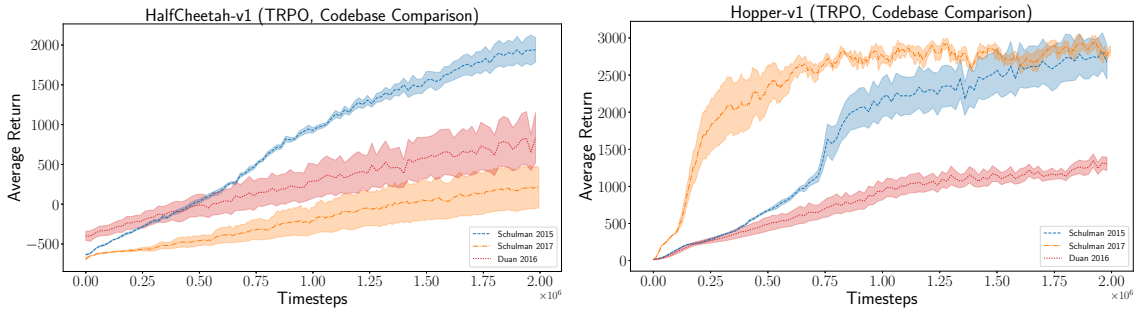

Figure A.29: TRPO codebase comparison using our default set of hyperparameters (as used in other experiments).

Figure A.29 shows a comparison of the TRPO implementations using the default hyperparamters as specified earlier in the supplemental. Note, the exception is that we use a larger batch size for rllab and original TRPO code of 20k samples per batch, as optimized in a second set of experiments. Figure A.24 and A.25 show the same network experiments for DDPG with the rllab++ code (Gu et al., 2017). We can then compare the performance of the algorithm across 3 codebases (keeping all hyperparameters constant at the defaults), this can be seen in Figure A.28.

A.8 Significance

Our full results from significance testing with difference metrics can be found in Table A.6 and Table A.7. Our bootstrap mean and confidence intervals can be found in Table A.10. Bootstrap power analysis can be found in Table A.11. To performance significance testing, we use our 5 sample trials to generate a bootstrap with 10k bootstraps. From this confidence intervals can be obtained. For the t-test and KS-test, the average returns from the 5 trials are sorted and compared using the normal 2-sample versions of these tests. Scipy[1] and Facebook Boostrapped[2] are used for the KS test, t-test, and bootstrap analysis. For power

[1] https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.ks_2samp.html
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html

[2] https://github.com/facebookincubator/bootstrapped

analysis, we attempt to determine if a sample is enough to game the significance of a 25% lift. This is commonly used in A/B testing (Tufféry, 2011).

-	DDPG	ACKTR	TRPO	PPO
DDPG	-	$t = 1.85, p = 0.102$ $KS = 0.60, p = 0.209$ 61.91 % (-32.27 %, 122.99 %)	$t = 4.59, p = 0.002$ $KS = 1.00, p = 0.004$ 301.48 % (150.50 %, 431.67 %)	$t = 2.67, p = 0.029$ $KS = 0.80, p = 0.036$ 106.91 % (-37.62 %, 185.26 %)
ACKTR	$t = -1.85, p = 0.102$ $KS = 0.60, p = 0.209$ -38.24 % (-75.42 %, -15.19 %)	-	$t = 2.78, p = 0.024$ $KS = 0.80, p = 0.036$ 147.96 % (30.84 %, 234.60 %)	$t = 0.80, p = 0.448$ $KS = 0.60, p = 0.209$ 27.79 % (-67.77 %, 79.56 %)
TRPO	$t = -4.59, p = 0.002$ $KS = 1.00, p = 0.004$ -75.09 % (-86.44 %, -68.36 %)	$t = -2.78, p = 0.024$ $KS = 0.80, p = 0.036$ -59.67 % (-81.70 %, -46.84 %)	-	$t = -2.12, p = 0.067$ $KS = 0.80, p = 0.036$ -48.46 % (-81.23 %, -32.05 %)
PPO	$t = -2.67, p = 0.029$ $KS = 0.80, p = 0.036$ -51.67 % (-80.69 %, -31.94 %)	$t = -0.80, p = 0.448$ $KS = 0.60, p = 0.209$ -21.75 % (-75.99 %, 11.68 %)	$t = 2.12, p = 0.067$ $KS = 0.80, p = 0.036$ 94.04 % (2.73 %, 169.06 %)	-

Table A.6: HalfCheetah Significance values and metrics for different algorithms. Rows in cells are: sorted 2-sample t-test, Kolmogorov-Smirnov test, bootstrap A/B comparison % difference with 95% confidence bounds.

-	DDPG	ACKTR	TRPO	PPO
DDPG	-	$t = -1.41, p = 0.196$ $KS = 0.60, p = 0.209$ -35.92 % (-85.62 %, -5.38 %)	$t = -2.58, p = 0.033$ $KS = 0.80, p = 0.036$ -44.96 % (-78.82 %, -20.29 %)	$t = -2.09, p = 0.070$ $KS = 0.80, p = 0.036$ -39.90 % (-77.12 %, -12.95 %)
ACKTR	$t = 1.41, p = 0.196$ $KS = 0.60, p = 0.209$ 56.05 % (-87.98 %, 123.15 %)	-	$t = -1.05, p = 0.326$ $KS = 0.60, p = 0.209$ -14.11 % (-37.17 %, 9.11 %)	$t = -0.42, p = 0.686$ $KS = 0.40, p = 0.697$ -6.22 % (-31.58 %, 18.98 %)
TRPO	$t = 2.58, p = 0.033$ $KS = 0.80, p = 0.036$ 81.68 % (-67.76 %, 151.64 %)	$t = 1.05, p = 0.326$ $KS = 0.60, p = 0.209$ 16.43 % (-27.92 %, 41.17 %)	-	$t = 2.57, p = 0.033$ $KS = 0.60, p = 0.209$ 9.19 % (2.37 %, 15.58 %)
PPO	$t = 2.09, p = 0.070$ $KS = 0.80, p = 0.036$ 66.39 % (-67.80 %, 130.16 %)	$t = 0.42, p = 0.686$ $KS = 0.40, p = 0.697$ 6.63 % (-33.54 %, 29.59 %)	$t = -2.57, p = 0.033$ $KS = 0.60, p = 0.209$ -8.42 % (-14.08 %, -2.97 %)	-

Table A.7: Hopper Significance values and metrics for different algorithms. Rows in cells are: sorted 2-sample t-test, Kolmogorov-Smirnov test, bootstrap A/B comparison % difference with 95% confidence bounds.

A.8 Significance

-	DDPG	ACKTR	TRPO	PPO
DDPG	-	$t = -1.03, p = 0.334$ $KS = 0.40, p = 0.697$ -30.78 % (-91.35 %, 1.06 %)	$t = -4.04, p = 0.004$ $KS = 1.00, p = 0.004$ -48.52 % (-70.33 %, -28.62 %)	$t = -3.07, p = 0.015$ $KS = 0.80, p = 0.036$ -45.95 % (-70.85 %, -24.65 %)
ACKTR	$t = 1.03, p = 0.334$ $KS = 0.40, p = 0.697$ 44.47 % (-80.62 %, 111.72 %)	-	$t = -1.35, p = 0.214$ $KS = 0.60, p = 0.209$ -25.63 % (-61.28 %, 5.54 %)	$t = -1.02, p = 0.338$ $KS = 0.60, p = 0.209$ -21.91 % (-61.53 %, 11.02 %)
TRPO	$t = 4.04, p = 0.004$ $KS = 1.00, p = 0.004$ 94.24 % (-22.59 %, 152.61 %)	$t = 1.35, p = 0.214$ $KS = 0.60, p = 0.209$ 34.46 % (-60.47 %, 77.32 %)	-	
PPO	$t = 3.07, p = 0.015$ $KS = 0.80, p = 0.036$ 85.01 % (-31.02 %, 144.35 %)	$t = 1.02, p = 0.338$ $KS = 0.60, p = 0.209$ 28.07 % (-65.67 %, 71.71 %)	$t = -0.57, p = 0.582$ $KS = 0.40, p = 0.697$ -4.75 % (-19.06 %, 10.02 %)	-

Table A.8: Walker2d Significance values and metrics for different algorithms. Rows in cells are: sorted 2-sample t-test, Kolmogorov-Smirnov test, bootstrap A/B comparison % difference with 95% confidence bounds.

-	DDPG	ACKTR	TRPO	PPO
DDPG	-	$t = -2.18, p = 0.061$ $KS = 0.80, p = 0.036$ -36.44 % (-61.04 %, -6.94 %)	$t = -4.06, p = 0.004$ $KS = 1.00, p = 0.004$ -85.13 % (-97.17 %, -77.95 %)	$t = -8.33, p = 0.000$ $KS = 1.00, p = 0.004$ -70.41 % (-80.86 %, -56.52 %)
ACKTR	$t = 2.18, p = 0.061$ $KS = 0.80, p = 0.036$ 57.34 % (-80.96 %, 101.11 %)	-	$t = -3.69, p = 0.006$ $KS = 1.00, p = 0.004$ -76.61 % (-90.68 %, -70.06 %)	$t = -8.85, p = 0.000$ $KS = 1.00, p = 0.004$ -53.45 % (-62.22 %, -47.30 %)
TRPO	$t = 4.06, p = 0.004$ $KS = 1.00, p = 0.004$ 572.61 % (-73.29 %, 869.24 %)	$t = 3.69, p = 0.006$ $KS = 1.00, p = 0.004$ 327.48 % (165.47 %, 488.66 %)	-	$t = 2.39, p = 0.044$ $KS = 0.60, p = 0.209$ 99.01 % (28.44 %, 171.85 %)
PPO	$t = 8.33, p = 0.000$ $KS = 1.00, p = 0.004$ 237.97 % (-59.74 %, 326.85 %)	$t = 8.85, p = 0.000$ $KS = 1.00, p = 0.004$ 114.80 % (81.85 %, 147.33 %)	$t = -2.39, p = 0.044$ $KS = 0.60, p = 0.209$ -49.75 % (-78.58 %, -36.43 %)	-

Table A.9: Swimmer Significance values and metrics for different algorithms. Rows in cells are: sorted 2-sample t-test, Kolmogorov-Smirnov test, bootstrap A/B comparison % difference with 95% confidence bounds.

Environment	DDPG	ACKTR	TRPO	PPO
HalfCheetah-v1	5037.26 (3664.11, 6574.01)	3888.85 (2288.13, 5131.96)	1254.55 (999.52, 1464.86)	3043.1 (1920.4, 4165.86)
Hopper-v1	1632.13 (607.98, 2370.21)	2546.89 (1875.79, 3217.98)	2965.33 (2854.66, 3076.00)	2715.72 (2589.06, 2847.93)
Walker2d-v1	1582.04 (901.66, 2174.66)	2285.49 (1246.00, 3235.96)	3072.97 (2957.94, 3183.10)	2926.92 (2514.83, 3361.43)
Swimmer-v1	31.92 (21.68, 46.23)	50.22 (42.47, 55.37)	214.69 (141.52, 287.92)	107.88 (101.13, 118.56)

Table A.10: Envs bootstrap mean and 95% confidence bounds

A.8 Significance

Environment	DDPG	ACKTR	TRPO	PPO
HalfCheetah-v1	100.00 0.00 0.00	79.03 11.53 9.43	79.47 20.53 0.00	61.07 10.50 28.43
Hopper-v1	60.90 10.00 29.10	79.60 11.00 9.40	0.00 100.00 0.00	0.00 100.00 0.00
Walker2d-v1	89.50 0.00 10.50	60.33 9.73 29.93	0.00 100.00 0.00	59.80 31.27 8.93
Swimmer-v1	89.97 0.00 10.03	59.90 40.10 0.00	89.47 0.00 10.53	40.27 59.73 0.00

Table A.11: Power Analysis for predicted significance of 25% lift. Rows in cells are: % insignificant simulations,% positive significant, % negative significant.

B

Supplemental Material : Reusability

B.1 Expanded Equations

The expectation over the discriminator loss for the option case can be expanded to

$$\begin{aligned} L_\Omega &= \mathbb{E}_\omega \left[\pi_{\Omega,\zeta}(\omega|s) L_{\hat{\theta},\omega} \right] + L_{reg} \\ &= \sum_\omega \pi_{\Omega,\zeta}(\omega|s) L_{\hat{\theta},\omega} + L_{reg}. \end{aligned} \tag{B.1}$$

Likewise, the regularization terms can also be expanded:

$$\begin{aligned} L_b &= \sum_\omega ||\mathbb{E}[\pi_\Omega(\omega)] - \tau||_2 \\ &\approx \sum_\omega ||\frac{1}{m_b} \sum_i^{m_b} (\pi_\Omega(\omega|s_i)) - \tau||_2 \end{aligned} \tag{B.2}$$

$$\begin{aligned} L_e &= \mathbb{E}\left[|| \left(\frac{1}{||\Omega||} \sum_\omega \pi_\Omega(\omega) \right) - \tau||_2 \right] \\ &\approx \frac{1}{m_b} \sum_i^{m_b} || \left(\frac{1}{||\Omega||} \sum_\omega \pi_\Omega(\omega|s_i) \right) - \tau||_2 \end{aligned} \tag{B.3}$$

$$
\begin{aligned}
L_v &= -\sum_{\omega} var_{\omega}\{\pi_{\Omega}(s)\} \\
&\approx -\sum_{\omega} \frac{1}{m_b} \sum_{i}^{m_b} \left(\pi_{\Omega}(\omega|s_i) - \left(\frac{1}{m_b} \sum_{i}^{m_b} \pi_{\Omega}(\omega|s_i) \right) \right)
\end{aligned}
\tag{B.4}
$$

B.2 Expert Collection

The expert demonstration rollouts (state sequences) for all OpenAI Gym (Brockman et al., 2016) environments were obtained from policies trained for 1000 iterations using Trust Region Policy Optimization (Schulman et al., 2015) with parameters $KL_{\max} = 0.01$, generalized advantage estimation $\lambda = 0.97$, discount factor $\gamma = 0.99$ and batch size $25,000$ (rollout timesteps shared when updating the discriminator and policy). For the Roboschool (Schulman et al., 2017) Flagrun-v1 environment, the rollouts were obtained using the PPO pre-trained expert provided with Roboschool.

B.3 Experimental Setup and Hyperparameters

Observations are **not** normalized in all cases as we found that it did not help or hurt performance. In all cases for advantage estimation we use a value approximator as in (Schulman et al., 2015), which uses L-BFGS optimization with a mixing fraction of 0.1. That is, it uses the current prediction $V_{\theta}(s)$ and mixes it with the actual discounted returns with 0.1 belonging to the actual discounted returns and 0.9 belonging to the current prediction. This is identical to the original Trust Region Policy Optimization (TRPO) code as provided at: `https://github.com/joschu/modular_rl/`. We perform a maximum of 20 L-BFGS iterations per value function update. For all the environments, we let the agent act until the maximum allowed timesteps of the specific environment (as set by default in OpenAI Gym), gather the rollouts and keep the number of timesteps per batch desired. For all policy optimization steps in both IRLGAN and OPTIONGAN, we use TRPO with the with parameters set to the same values as the ones used for the expert collection ($KL_{\max} = 0.01$, generalized advantage estimation $\lambda = 0.97$, discount factor $\gamma = 0.99$ and

batch size $25,000$), except for the Roboschool Flagrun Experiment where PPO was used instead, as explained in its respective section below.

B.3.1 Simple Tasks and Transfer Tasks

For simple tasks we use 10 expert rollouts while for transfer tasks we use 40 expert rollouts (10 from each environment variation).

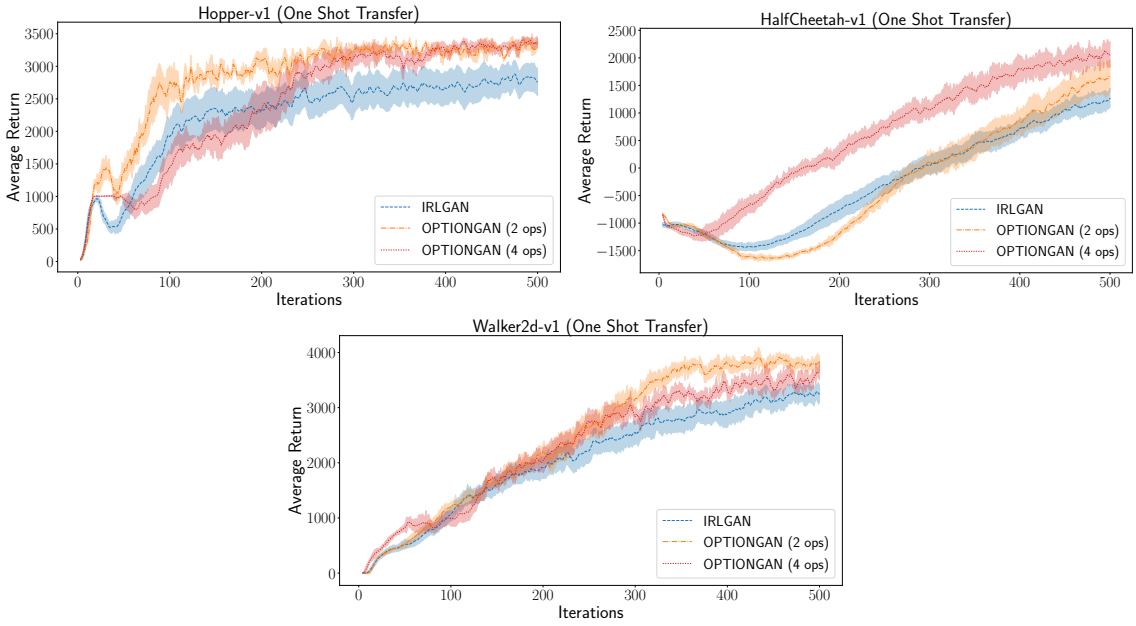

Figure B.1: Experiments from 40 expert demonstrations from varied gravity environments without any demonstrations on the novice training environment. Average returns from expert demonstrations across all mixed environments: 3657.71 (Hopper-v1), 4181.97 (HalfCheetah-v1), 4218.12 (Walker2d-v1).

IRLGAN

We use a Gaussian Multilayer Perceptron policy as in (Schulman et al., 2015) with two 64 unit hidden layers and tanh hidden layer activations. The output of the network gives the Gaussian mean and the standard deviation is modeled by a single learned variable as in (Schulman et al., 2015). Similarly for our discriminator network, we use the same architecture with a sigmoid output, tanh hidden layer activations, and a learning rate of

B.3 Experimental Setup and Hyperparameters

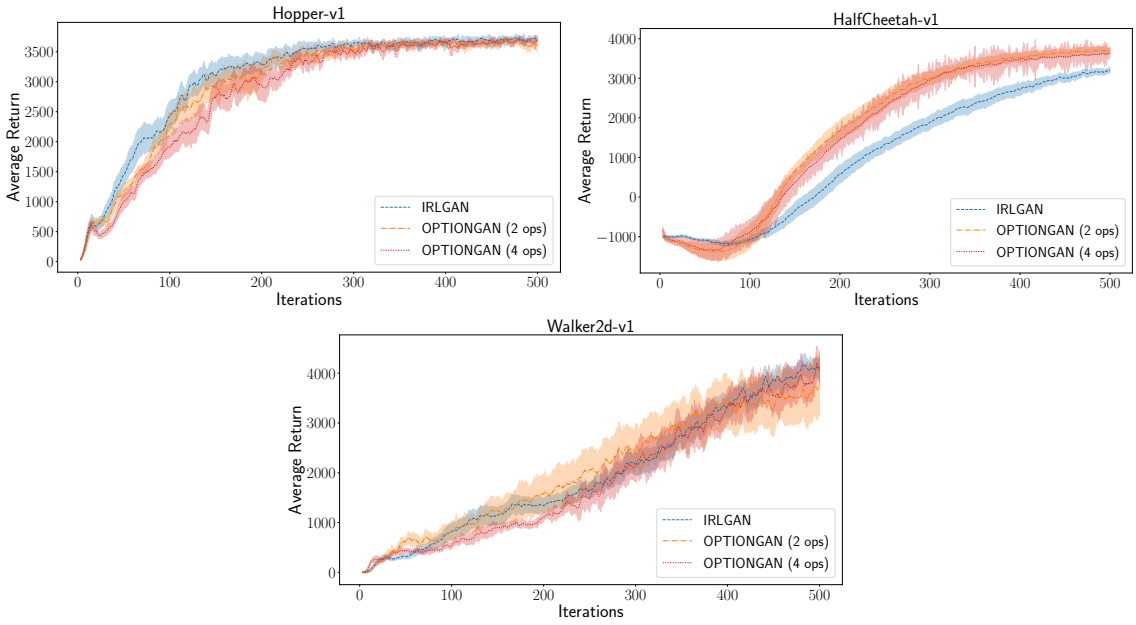

Figure B.2: Simple locomotion task learning curves. The True Average Return provided by the expert demonstrations are: 3778.82 (Hopper-v1), 4156.94 (HalfCheetah-v1), 5528.51 (Walker2d-v1). Error bars indicate standard error of True Average Return across 10 trial runs.

.

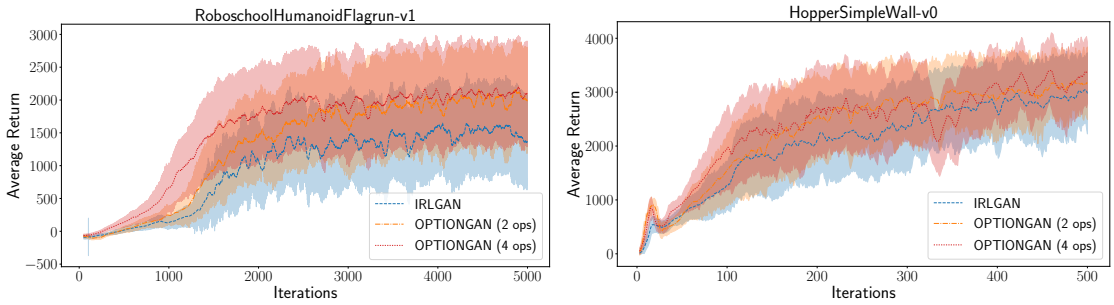

Figure B.3: Evaluation on RoboschoolHumanoidFlagrun-v1 environment (Schulman et al., 2017). Average expert rollout returns: 2822.13.

$1 \cdot 10^{-3}$ for the discriminator. We do not use entropy regularization or $l2$ regularization as it resulted in worse performance. For every policy update we perform 3 discriminator updates as we found the policy optimization step is able to handle this and results in faster learning.

OPTIONGAN

Aligning with the IRLGAN networks, we make use of a Gaussian Multilayer Perceptron policy as in (Schulman et al., 2015) with 2 hidden layers of 64 units with tanh hidden layer activations for our shared hidden layers. These hidden layers connect to $||\Omega||$ options depending on the experiment (2 or 4). In this case the output of the network gives the Gaussian mean for each option and the standard deviation is modeled by a single learned variable per option. The policy-over-options is also modeled by a neural network of size $(64, 64)$ – two hidden layers of size 64 each – with tanh activations and a softmax output corresponding to the number of options. For our discriminator, we use the same architecture with tanh hidden layer activations, a sigmoid output and $||\Omega||$ outputs, one for each option. We use the policy over options to create a specialized mixture of experts model with a specialized loss function which converges to options. We use a learning rate of 10^{-3} for the discriminator. Same as in IRLGAN, we do not make use of entropy regularization or $l2$ regularization as we found either regularizers to hurt performance. Instead we use scaling factors for the regularization terms included in the loss: $\lambda_b = 10.0, \lambda_e = 10.0, \lambda_v = 1.0$ for the 2 options case and $\lambda_b = 0.01, \lambda_e = 10.0, \lambda_v = 1.0$ for the 4 options case. Again, we perform 3 discriminator updates per policy update

B.3.2 RoboschoolHumanoidFlagrun-v1

For Roboschool experiments we use proximal policy optimization (PPO) with a clipping objective (Schulman et al., 2017) (clipping parameter set to $\epsilon = 0.02$). We perform 5 Adam (Kingma and Ba, 2015) policy updates on the PPO clipping objective with a learning rate of 10^{-3}. The value function and advantage estimation parameters from previous experiments are maintained while our network architecture sizes are increased to $(128, 128)$ and use ReLU activations instead of tanh.

B.4 Reward Decomposition over Expert Demonstrations

We show that the trained policy-over-options network shows some intrinsic structure over the expert demonstrations.

B.4 Reward Decomposition over Expert Demonstrations

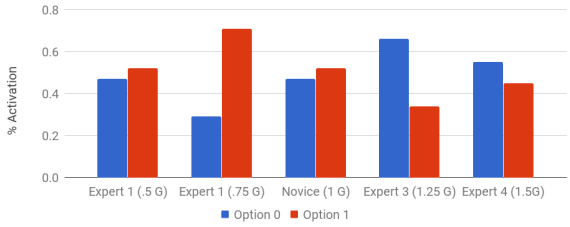

Figure B.4: Probability distribution of π_Ω over options on expert demonstrations. Inherent structure is found in the demonstrations.

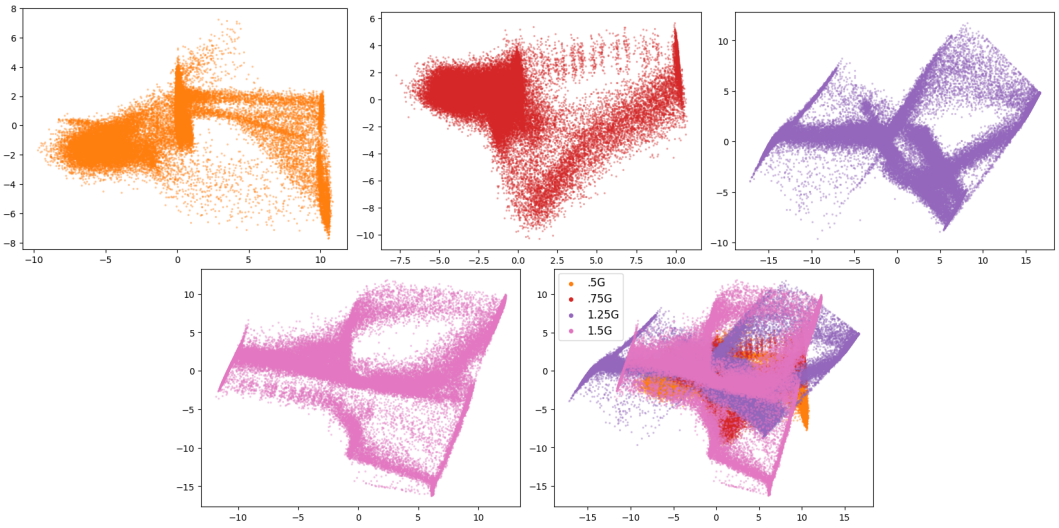

Figure B.5: State distribution of expert demonstrations projected onto a 2D plane to pair with Figure 5.5. Axes correspond to projected state dimensions from the original 6-D state space using SVD.

In Figure B.4 are shown the activation of the gating function across expert rollouts after training. We see that the underlying division in expert demonstrations is learned by the policy-over-options, which indicates that our method for training the policy-over-options induces it to learn a latent structure to the expert demonstrations and thus can benefit in the transfer case since each option inherently specializes to be used in different environments. We find that options specialized more clearly over the experts with environments closest to the normal gravity environment, while the others use an even mixture of options. This is due to the fact that the mixing specialized options are able to cover the state space of

the non-specialized options as we can observe from the state distribution of the expert demonstrations shown in Figure B.5.