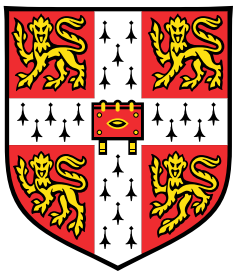
Sample-Efficient Deep Reinforcement Learning for Continuous Control



Shixiang Gu

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Jesus College

July 2019

I would like to dedicate this thesis to my loving family …

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Shixiang Gu
July 2019

Acknowledgements

First and foremost, I would like to acknowledge my advisors Richard E. Turner, Zoubin Ghahramani, and Bernhard Schölkopf for their kind and patient support in my rather exploratory PhD endeavor. They have inspired me with theories and ideas from Bayesian machine learning, kernel methods, and causality, and most importantly, they all valued my personal growth as a researcher more than specific research directions and allowed me to explore various promising directions of research. I appreciate Rich for again and again explaining complex machine learning concepts in absolute clarity, and helping me generalize or concretize research ideas. I also thank Zoubin for creating a collaborative, open atmosphere in Cambridge Machine Learning Group, and illuminating how different machine learning methods connect to one another with simple diagrams. My time working with Bernhard was more limited; however, he served as an inspirational figure for me for his endless curiosity as a scientist, as he always thought toward creating new fields of research with most impacts. I received a lot of wisdom and inspiration from all my advisors, and they are researchers I aspire to one day become. If I list one of the most important lessons, that is the art of abstraction. Machine learning research, particularly of deep learning, has recently exploded in terms of the amount of papers. The key technique that helped me navigate this breadth of work is to understand the fundamentals of each method, to abstract away its details, and to position each of them in a unified picture. Cambridge, Max Planck Institute, and Bayesian philosophy of thinking particularly encouraged me to think in these terms, and such ability allowed me to explore practical yet fundamental enough algorithmic improvements in probabilistic machine learning, deep learning, and reinforcement learning through my PhD. The inspiration for this thesis also comes from abstraction, as I aim to abstract, connect and improve model-based, value-based, and policy-based approaches in reinforcement learning.

Besides my PhD advisors, there are many other individuals who have inspired and supported me. I would like to first thank two key mentors who first inspired to pursue PhD and the career as a researcher: Professor Steve Mann and Professor Geoffrey Hinton during my undergraduate time at the University of Toronto. I was exposed to the excitements of pushing the frontier of what know through research, by working with Professor Mann on a project that we later demoed at SIGGRAPH. When I worked with Professor Hinton on

my undergraduate thesis, it was a truly exciting time at his lab. The results on speech and image recognitions were about to revolutionalize these research fields and it was the dawn of deep learning. Beyond his humble victory speech of "there is no turning back" remark, what has inspired me of his life is his commitment to research and education. I thank him greatly for managing times out, sometimes in the evenings or weekends, to meet with me and discuss even the most basics of research. The life of a researcher, despite numerous apparent difficulties and commitments, therefore appeared to me as tremendously rewarding, as I see the two professors continually explore and enable new fields of research through their entire careers.

My acknowledgements cannot end without thanking amazing individuals I encountered and worked with during my internships at Google Brain and DeepMind. I would like to thank Ilya Sutskever and Vincent Vanhoucke for giving me the valuable opportunity to do my first internship at Google Brain right after my first year of PhD. The environment at Google Brain was incredibly attractive: many reknown researchers and faculties frequented at microkitchens; computation resources and data were abundant for fast iterations and quick scaling of research ideas; and research freedom empowered me to work on whatever topics of interest. In addition, I thank Ilya for introducing me to Sergey Levine of the University of California Berkeley and Timothy Lillicrap of DeepMind, two individuals who introduced me to the field of reinforcement learning and robotics and with whom I formed the longest duration of external collaboration. Sergey has not only been tremendouly helpful in positioning, brainstorming, and iterating research ideas, but also been a great educator who valued the growths and successes of all his collaborators. Tim, whose pioneering work in deep RL for continuous control inspired our joint and my first work in RL, committed to difficult long-distance collaborations, with meeting times often during his evening times, and later kindly hosted me for a part-time internship visit at DeepMind. A bulk of my thesis would not have happened if not for the support from these individuals, along with the research freedom from my advisors.

Beyond the names listed above, there are many more individuals I feel thankful toward. I was particularly fortunate to have spent time at the University of Cambridge, Max Planck Institute, Google Brain and DeepMind, where I met such diverse, open-minded, and intelligent individuals who have and continue to inspire me. While it is hard to exhaustively list all of them, in no specific order I would like to thank: Charlie Tang, Jimmy Lei Ba, Chi Jin, Navdeep Jaitly, George Dahl, Chris Maddison, Yujia Li, Nitish Srivastava, Emily Denton, Laurent Charlin, Alex Graves, Alex Krizhevsky, Ajay Agrawal, Brendan Frey, Richard Zemel at the University of Toronto; Adam Ścibior, Mateo Rojas-Carulla, Yingzhen Li, Thang Bui, Nilesh Tripuraneni, Matej Balog, Matthias Bauer, Paul Rubenstein, Chaochao Lu, Alessan-

dro Davide Ialongo, Niki Kilbertus, Robert Pinsler, John Bronskill, Yutian Chen, Hong Ge, Yarin Gal, Rowan McAllister, Amar Shah, Mark van der Wilk, Mark Rowland, Matthew W. Hoffman, Felipe Tobar, James R. Lloyd, David Duvenaud, Jose Miguel Hernandez-Lobato, Carl Edward Rasmussen from the Uiversity of Cambridge; Okan Koc, Dieter Büchler, Monotonobu Kanagawa, Giambattista Parascandolo, Diana Rebmann, Hans Kersting, Maren Mahsereci, Atalanti-Anastasia Mastakouri, Vinay Jayaram, Matthias Hohmann, Michael Hirsch, Sebanstian Trimpe, Philip Hennig, Jan Peters from the Max Planck Insitute for Intelligent Systems, Tübingen; Natasha Jaques, Eric Zhang, Ethan Holly, Laurent Dinh, Stephan Zhang, Ben Poole, Jakob N. Foerster, Thang Luong, Deirdre Quillen, Kelvin Xu, Douglas Eck, Quoc Le, Jascha Sohl-Dickstein, Ian Goodfellow, Sylvain Gelly, Oriol Vinvals, Samy Bengio, Jeff Dean from Google Brain; Nicolas Heess, Andriy Mnih, Tom Erez, Yuval Tassa, Gabriel Dulac-Arnold, Jon Scholz, Shakir Mohammed, David Silver, Raia Hadsell, Shane Legg from DeepMind; Chelsea Finn, Vitchyr Pong, Pulkit Agrawal, Murtaza Dalal, Marvin Zhang, Justin Fu, Murtaza Dalal, Carlos Florensa Campo, Yan (Rocky) Duan, Xi (Peter) Chen, John Schulman, Pieter Abbeel from the University of California Berkeley.

Last but not least, I would like to thank my mother Yueqi Zhou for providing me with the best environments to learn and succeed. On important matters in life and career, she valued listening to my opinions, shared her thoughts and wisdom patiently, and discussed with me many hours until we agree on the fundamentals. I thank her sincerely for her continual and selfless support.

Abstract

Reinforcement learning (RL) is a powerful, generic approach to discovering optimal policies in complex sequential decision-making problems. Recently, with flexible function approximators such as neural networks, RL has greatly expanded its realm of applications, from playing computer games with pixel inputs, to mastering the game of Go, to learning parkour movements by simulated humanoids. However, the common RL approaches are known to be sample intensive, making them difficult to be applied to real-world problems such as robotics. This thesis makes several contributions toward developing RL algorithms for learning in the wild, where sample-efficiency and stability are critical. The key contributions include Normalized Advantage Functions (NAF), extending Q-learning for continuous action problems; Interpolated Policy Gradient (IPG), unifying prior policy gradient algorithm variants through theoretical analyses on bias and variance; and Temporal Difference Models (TDM), interpreting a parameterized Q-function as a generalized dynamics model for novel temporally abstracted model-based planning. Importantly, this thesis highlights that these algorithms can be seen as bridging gaps between branches of RL – model-based with model-free, and on-policy with off-policy. The proposed algorithms not only achieve substantial improvements over the prior approaches, but also provide novel perspectives on how to mix different branches of RL effectively to gain the best of both worlds. NAF has subsequently been shown to be able to train two 7-DoF robot arms to open doors using only 2.5 hours of real-world experience, making it one of the first demonstrations of deep RL approaches on real robots.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

a	Action
p	State transition probability; initial state probability
p_t^π	Marginal state distribution at time t for policy π
Q^π	State-action value function (Q-function) of policy π
Q^*	Optimal state-action value function (optimal Q-function)
R	Cumulative rewards
r	Reward function
s	State
V^π	State value function of policy π
V^*	Optimal state value function

Greek Symbols

γ	Discount factor
π	Policy function
ρ^π	γ-discounted, unnormalized state-visitation frequency of policy π
$\tilde{\rho}^\pi$	Non-discounted, unnormalized state-visitation frequency of policy π
τ	Trajectory consisting of a sequence of states s and actions a

Acronyms / Abbreviations

AC Actor-Critic

DDPG Deep Deterministic Policy Gradient

DNN Deep Neural Network

DQN Deep Q-Network

ES Evolutionary Strategy

GAE Generalized Advantage Estimation

IPG Interpolated Policy Gradient

LSPI Least-Squares Policy Iteration

LSTD Least-Squares Temporal Difference

MDP Markov Decision Process

NAF Normalized Advantage Functions

PG Policy Gradient

POMDP Partially-Observable Markov Decision Process

PPO Proximal Policy Optimization

RL Reinforcement Learning

SGD Stochastic Gradient Descent

SVG Stochastic Value Gradient

TD Temporal Difference

TRPO Trust-Region Policy Optimization

Chapter 1

Introduction

1.1 Reinforcement Learning and Sample Efficiency

We live in the world of sequential data and constant decision-making. The beauty of time, sequence, and recursion is that simple functions, e.g. laws of physics, can enable rich complex phenomena, e.g. the earth and us, to emerge as a consequence. We, and other living organisms capable of decision-making, can choose to predictively alter the course of the future by acting in the environment. This ability to achieve desired goals through interation is essential for survival as individuals or species. Richness of goals and how predictably we can accomplish them have been proposed as key metrics for defining a universal measure of intelligence (Legg and Hutter, 2007; Salge et al., 2014). How we can most effectively change the passive dynamics of the world to achieve what we want, is at the core of sequential decision-making and is the fundamental problem studied in the field of reinforcement learning (RL)

Reinforcement learning (RL) studies a series of approaches for solving arbitrary goal-directed sequential decision-making problems with only high-level reward signals and no supervision. It has recently been extended to utilize large neural network policies and value functions, and has been successful in solving a range of difficult problems, such as playing computer games from pixels, the game of Go, and continuous control of humanoids in simulation (Heess et al., 2017; Lillicrap et al., 2016; Mnih et al., 2015; Schulman et al., 2016; Silver et al., 2016). The use of deep neural networks (DNNs) to parameterize functions which are then learned from data minimizes the need for manual feature and policy engineering, and allows learning end-to-end policies mapping from high-dimensional inputs, such as images, directly to actions. However, such expressive parametrization also introduces a number of practical problems. Deep reinforcement learning algorithms tend to be sensitive to hyperparameter settings, often requiring extensive hyperparameter sweeps to find good

values. Poor hyperparameter settings tend to produce unstable or non-convergent learning. Deep RL algorithms also tend to exhibit high sample complexity, often to the point of being impractical to run on some real physical systems. In this thesis, we propose scalable RL methods with improved sample efficiency that connect and bridge gaps between branches of RL: model-based with model-free, and on-policy with off-policy.

1.2 Thesis Outline

- Chapter 2 provides an overview on the basics of model-based, value-based, and policy-based approaches to reinforcement learning, highlighting their differences in terms of learning signals and target functions to estimate.
- Chapter 3 discusses normalized advantage functions (NAF) and LQR-based acceleration of Q-learning for continuous control. The content is adapted from:
  - Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, Sergey Levine. "Continuous Deep Q-Learning with Model-based Acceleration". ICML 2016. (Gu et al., 2016b)
  - Shixiang Gu*, Ethan Holly*, Timothy Lillicrap, Sergey Levine. "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates". ICRA 2017. (Gu et al., 2017a)
- Chapter 4 discusses interpolated policy gradient (IPG), a framework for unifying off-policy actor-critic and on-policy Monte Carlo policy gradient and providing theoretical bounds on the bias induced through imperfect critic and off-policy sampling. The content is adapted from:
  - Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E. Turner, Sergey Levine. "Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic". ICLR 2017. (Gu et al., 2017b)
  - Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E. Turner, Bernhard Schölkopf, Sergey Levine. "Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning". (Gu et al., 2017c)
- Chapter 5 discusses temporal difference models (TDM) and shows that an off-policy value-based method with vectorized parameterization, goal-conditioned rewards, and

relabeling trick (Andrychowicz et al., 2017) can generalize model-based methods to achieve substantial improvements in sample efficiency. The content is adapted from:

  - Vitchyr Pong*, Shixiang Gu*, Murtaza Dalal, Sergey Levine. "Temporal Difference Models: Model-Free Deep RL for Model-Based Control". ICLR 2018. (Pong et al., 2018)

Each of the papers mentioned above is through multi-author collaboration. However, in every case I was the (joint) first author. As such, I contributed the most in formulating the research, implementing the code, writing the paper, and running the experiments. I have also published the following papers whilst carrying out my PhD, but these have not been included in this thesis:

- Nilesh Tripuraneni*, Shixiang Gu*, Hong Ge, Zoubin Ghahramani. "Particle Gibbs for Infinite Hidden Markov Models". NIPS 2015. (Tripuraneni et al., 2015)
- Shixiang Gu, Zoubin Ghahramani, Richard E. Turner. "Neural Adaptive Sequential Monte Carlo". NIPS 2015. (Gu et al., 2015)
- Shixiang Gu, Sergey Levine, Ilya Sutskever, Andriy Mnih. "MuProp: Unbiased Backpropagation for Stochastic Neural Networks". ICLR 2016. (Gu et al., 2016a)
- Eric Jang, Shixiang Gu, Ben Poole. "Categorical Reparametrization with Gumble-Softmax". ICLR 2017. (Jang et al., 2017)
- Natasha Jaques, Shixiang Gu, Dzmitry Bahdanau, Jose Miguel Hernndez Lobato, Richard E. Turner, Douglas Eck. "Sequence Tutor: Conservative fine-tuning of sequence generation models with KL-control". ICML 2017. (Jaques et al., 2017)
- Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, Sergey Levine. "Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning". ICLR 2018. (Eysenbach et al., 2018)
- George Tucker, Surya Bhupatiraju, Shixiang Gu, Richard E. Turner, Zoubin Ghahramani, Sergey Levine. "The Mirage of Action-Dependent Baselines in Reinforcement Learning". ICML 2018. (Tucker et al., 2018)
- Ofir Nachum, Shixiang Gu, Honglak Lee, Sergey Levine. "Data-Efficient Hierarchical Reinforcement Learning". NIPS 2018. (Nachum et al., 2018)

Chapter 2

Reinforcement Learning Algorithms

This chapter provides an overview of the basic approaches in reinforcement learning. We first introduce the standard objective, functions, and mathematical notations in RL used throughout this thesis, and then move on to detail classes of RL algorithms based on two groupings: (1) model-based and model-free, and (2) on-policy and off-policy. The emphasis is put on the simplest formulations, which are often most compatible with scalable neural network function approximations. This also enables us to abstract many approaches in RL to the most basic forms, and illuminate their advantages, disadvantages and relations. Beyond the materials covered here, RL has a rich set of algorithms and theoretical analyses. We will refer readers to the excellent books by Sutton and Barto (1998) and Szepesvári (2010) for more comprehensive overviews on the history and development of RL, and the survey by Deisenroth et al. (2013) for a practical understanding into challenges that arise when RL is applied to real-world applications such as robotics.

2.1 Reinforcement Learning

The basic RL formulation involves an agent interacting with a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, \gamma, P, r)$, where $\mathcal{S}$ denotes the fully-observed state space of the environment and the agent, $\mathcal{A}$ denotes the action space, γ denotes the discount factor, $P = \{p(s_0), p(s_{t+1}|s_t, a_t)\}$ denotes the initial state distribution and transition dynamics of the environment, and $r(s,a)$ denotes the reward function. At time t, an agent in state $s_t \in \mathcal{S}$ takes an action $a_t \in \mathcal{A}$ according to its policy $\pi(a_t|s_t)$, the state transitions to s_{t+1} according to P, and the agent receives a reward $r_t = r(s_t, a_t)$. The objective is to learn the optimal policy π^*

to maximize the γ-discounted cumulative future return,

$$J(\pi) = \mathbb{E}_{s_0 \sim p(s_0), a_0 \sim \pi(a_0|s_0), s_1 \sim p(s_1|s_0,a_0), a_1 \sim \pi(a_1|s_1), \cdots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \tag{2.1}$$

$$= \mathbb{E}_{s_0, a_0, s_1, \cdots \sim P, \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], \tag{2.2}$$

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{2.3}$$

where we assume infinite planning horizon. The difficulty of RL is that we do not know the dynamics P of the MDP and we can only sample transitions from it, usually in an episodic manner without control over initial state distributions. In certain cases, we also assume that we do not know the reward function $r(s,a)$ and only observe its sampled values. For real-world applications such as robotics, it is crucial to minimize the number of samples required for learning, i.e. maximize the sample efficiency of the algorithm.

The main goal is to learn the optimal policy π^*, which can be done by directly optimizing a parameterized π with respect to the above objective, known as *policy-based* approaches. Alternative approaches estimate other functions from the experience data, from which π^* can be efficiently derived. Two such approaches are *value-based* and *model-based*. Value-based approaches learn the Q-function $Q^\pi(s,a)$, or state-action value function, which summarizes the future return of taking action a from state s and following policy π afterwards:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \cdots \sim P, \pi | s_t, a_t} \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'}) \right]. \tag{2.4}$$

The estimated Q-function can then be used to improve the policy. In particular, if the Q-function Q^* for the optimal policy π^* is learned, the optimal policy is directly given by $\pi^*(a_t|s_t) = \delta(a_t = \arg\max_a Q^*(s_t,a))$.

Model-based approaches, on the other hand, estimate the dynamics $\hat{p}(s_{t+1}|s_t,a_t) \approx p(s_{t+1}|s_t,a_t)$ and the reward function $\hat{r}(s,a) \approx r(s,a)$ through fitting approximators $\hat{p}(s_{t+1}|s_t,a_t)$ and $\hat{r}(s,a)$ from observed transitions $\{s_t, a_t, r_t, s_{t+1}\}$, and then use the learned models to approximately solve for the optimal policy as below,

$$\pi^* \approx \arg\max_{\pi} \hat{J}(\pi) = \arg\max_{\pi} \mathbb{E}_{s_0, a_0, s_1, \cdots \sim \hat{P}, \pi} \left[\sum_{t=0}^{\infty} \gamma^t \hat{r}(s_t, a_t) \right]. \tag{2.5}$$

Importantly, it is also trivial to use the learned model to evaluate an approximation to the state-action value function of any policy π as,

$$Q^{\pi}(s_t, a_t) \approx \mathbb{E}_{s_{t+1}, a_{t+1}, \cdots \sim \hat{P}, \pi | s_t, a_t} \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} \hat{r}(s_{t'}, a_{t'}) \right]. \tag{2.6}$$

The quality of approximations to π^* and Q^{π} depends strongly on the quality of the learned dynamics $\hat{P}$ and reward function $\hat{r}$.

2.2 Model-based and Model-free Algorithms

This section describes three approaches in RL based on being model-based or model-free, where the model-free includes value-based and policy-based algorithms. For the clarify of presentation, we explicitly separate model-based from model-free in the discussion here; however, as we analyze in Chapter 5, there exists subtle connections between model-based and model-free (particularly value-based).

2.2.1 Model-based Algorithms

Model-based algorithms, more commonly known as optimal control in the control theory literatures, are a set of algorithms that utilize a learned or available approximate model of an MDP to solve for the optimal policy without relying on real environment samples. While classic control often only involves planning with an available model of the robot or the control system with minimal model learning such as online system identification, model-based algorithms in RL literatures often involve both *model learning* as well as *model planning*. A diverse set of parameterizations of the model have been explored, including locally linear dynamics as in linear-quadratic regulator (LQR) systems (Levine et al., 2016; Li and Todorov, 2004; Todorov and Li, 2005), feed-forward and recurrent neural networks (Nguyen and Widrow, 1990; Schmidhuber, 1990, 1991; Wahlström et al., 2015; Watter et al., 2015; Werbos, 1989), and non-parametric probabilistic models such as Gaussian processes (Deisenroth and Rasmussen, 2011). While subtle differences exist in their model learning objectives, simple maximium likelihood estimation (MLE) is most frequently used. Equivalently, they minimize the inclusive Kullback–Leibler (KL) divergence between $p(s_{t+1}|s_t, a_t)$ and a family of variational distribution $q(s_{t+1}|s_t, a_t)$ based on transition

samples,

$$\hat{p} = \arg\min_q \mathbb{E}_{s_t,a_t\sim P,\beta} \left[D_{\mathrm{KL}}(p(\cdot|s_t,a_t), q(\cdot|s_t,a_t)) \right] \tag{2.7}$$

$$= \arg\max_q \mathbb{E}_{s_t,a_t,s_{t+1}\sim P,\beta} \left[\log q(s_{t+1}|s_t,a_t) \right]. \tag{2.8}$$

The notation β means that that samples can come from any behavior policy and do not need to come from on-policy rollouts based on π. This makes the model-based approach generally seen as an *off-policy* algorithm, capable of utilizing all samples collected during learning to make policy improvement. Furthermore, model-based RL can use the rich learning signals available in predicting high-dimensional observation s_{t+1}, e.g. an image, which is substantially more than predicting scalar rewards or cumulative returns. These properties make model-based algorithms very sample-efficient when a good, generalizable model can be learned from samples (Deisenroth and Rasmussen, 2011; Kurutach et al., 2018; Nagabandi et al., 2017). However, when learning a good model is difficult, e.g. dynamics are discontinuous and observations are high dimensional, model-based algorithms often exhibit worse asymptotic performances (with infinite samples) than policy-based or value-based approaches that are considered *model-free* (Gu et al., 2016b; Pong et al., 2018).

The choice of parameterization and learning objective for the model affects the model-based policy performance, but so does the choice of planning algorithm. Given a fitted model of dynamics $\hat{p}$, the derivation of the optimal policy is just another RL problem, with the following additional conditions: (1) samples can be queried infinitely, (2) states can be reset to any states during rollouts, and (3) dynamics may be differentiable. Therefore, new algorithms utilizing these additional properties, along with any standard policy-based or value-based algorithms can be used to derive the policy. In fact, recent results that have shown more success with model-based RL using neural networks have utilized dynamics-gradient-free policy derivation from the model (Ha and Schmidhuber, 2018; Kurutach et al., 2018; Nagabandi et al., 2017). Below are a list of planning algorithm examples to derive the optimal policy from a model:

1. Closed-form solve, e.g. linear-quadratic regulator (LQR), iterative LQG (iLQG) (Levine et al., 2016; Li and Todorov, 2004; Todorov and Li, 2005; Watter et al., 2015)

2. Gradient-based action optimization, e.g. model-predictive control (MPC) with gradient-based action updates

3. Gradient-free action optimization, e.g. MPC with gradient-free action updates (Chebotar et al., 2017b; Nagabandi et al., 2017; Theodorou et al., 2010) and Monte Carlo tree search

4. Gradient-based policy optimization, e.g. PILCO (Deisenroth and Rasmussen, 2011)
5. Gradient-free policy optimization, e.g. TRPO or evolutionary strategies with model-based policy evaluation (Ha and Schmidhuber, 2018; Kurutach et al., 2018)
6. Q-learning, e.g. Dyna-Q algorithms (Gu et al., 2016b; Sutton, 1990)
7. Actor-critic method, e.g. stochastic value gradients (SVG) (Heess et al., 2015a)

Taken together, the model parametrization, model learning objective, and planning algorithm offer a wide range of design choices for model-based algorithms. While a wide gap on asymptotic performance against model-free algorithms exists at the moment, the hope is that further research can yield substantially better results for model-based in the future.

Iterative LQG

The iterative Linear-Quadratic Gaussian (iLQG) is a popular model-based algorithm that optimizes trajectories by iteratively constructing locally optimal linear feedback controllers under a local linearization of the dynamics $p(s_{t+1}|s_t,a_t) = \mathcal{N}(s_{t+1}; f_{st}s_t + f_{at}a_t, F_t)$ and a quadratic expansion of the rewards $r(s_t,a_t)$ (Tassa et al., 2012). Under linear dynamics and quadratic rewards, the action-value function $Q(s_t,a_t)$ and value function $V(s_t)$ are locally quadratic and can be computed by dynamics programming.[1]

$$\begin{aligned} Q_{sa,sat} &= r_{sa,sat} + f_{sat}^T V_{s,st+1} f_{sat} \\ Q_{sat} &= r_{sat} + f_{sat}^T V_{s,st+1} \\ V_{s,st} &= Q_{s,st} - Q_{a,st}^T Q_{a,at}^{-1} Q_{a,st} \\ V_{st} &= Q_{st} - Q_{a,st}^T Q_{a,at}^{-1} Q_{at} \\ Q_{s,sT} &= V_{s,sT} = r_{s,sT} \end{aligned} \tag{2.9}$$

The time-varying linear feedback controller $g(s_t) = \hat{a}_t + k_t + K_t(s_t - \hat{s}_t)$ maximizes the locally quadratic Q, where $k_t = -Q_{a,at}^{-1} Q_{at}$, $K_t = -Q_{a,at}^{-1} Q_{a,st}$, and $\hat{s}_t, \hat{a}_t$ denote states and actions of the current trajectory around which the partial derivatives are computed. Employing the maximum entropy objective (Levine and Koltun, 2013), we can also construct a linear-Gaussian controller, where c is a scalar to adjust for arbitrary scaling of the reward magnitudes,

$$\pi_t^{iLQG}(a_t|s_t) = \mathcal{N}(a_t; \hat{a}_t + k_t + K_t(s_t - \hat{s}_t), -cQ_{a,at}^{-1}) \tag{2.10}$$

[1] While standard iLQG notation denotes Q, V as discounted sum of costs, we denote them as sum of rewards to make them consistent with the rest of the thesis

When the dynamics are not known, a particularly effective way to use iLQG is to combine it with learned time-varying linear models $\hat{p}(s_{t+1}|s_t, a_t)$. In this variant of the algorithm, trajectories are sampled from the controller in Eq. 2.10 and used to fit time-varying linear dynamics with linear regression. These dynamics are then used with iLQG to obtain a new controller, typically using a KL-divergence constraint to enforce a trust region, so that the new controller does not deviate too much from the region in which the samples were generated (Levine and Abbeel, 2014).

2.2.2 Value-based Algorithms

Value-based approaches are model-free algorithms that estimate value function $V(s)$ or state-action value function $Q(s,a)$ in order to efficiently solve for the optimal policy. The core of most value-based approaches is to find functions that satisfy Bellman equations, i.e. their recursive dynamic programming definitions. While value-based algorithms are very broad in scope (Sutton and Barto, 1998; Szepesvári, 2010), we will focus on methods that estimate Q^π or Q^*, since these are methods that can directly utilize off-policy samples for learning. Specifically, we can re-write Q^π definition in Eq. 2.4 recursively as,

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \cdots \sim P, \pi | s_t, a_t} \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'}) \right] \tag{2.11}$$

$$= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p(\cdot|s_t, a_t), a_{t+1} \sim \pi(\cdot|s_{t+1})} \left[Q^\pi(s_{t+1}, a_{t+1}) \right]. \tag{2.12}$$

This suggests minimizing the following surrogate objective for learning a parameterized state-action value function Q_w to approximate Q^π,

$$L(w) = \mathbb{E}_{s_t, a_t, s_{t+1} \sim P, \beta} \left[\left(Q_w(s_t, a_t) - y_t \right)^2 \right] \tag{2.13}$$

$$y_t = r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi(\cdot|s_{t+1})} \left[Q_w(s_{t+1}, a_{t+1}) \right]. \tag{2.14}$$

Such method that optimizes for Bellman consistency is also known as temporal difference (TD) learning (Tesauro, 1995), a method for approximate dynamic programming. In this example, it is evaluating Q^π, i.e. performing policy evaluation; however, a different Bellman equation for Q^* leads to the renowned Q-learning algorithm (Watkins and Dayan, 1992). Notice that for these temporal difference learning objectives, we could directly use samples from any behavior policy β as similarly done in the model-based objective of Eq. 2.8. The capability for simple off-policy learning enable value-based methods to be more sample efficient than policy-based approaches which are generally only effective on-policy; however, since it optimizes for a surrogate Bellman error objective, its convergence is harder to

analyze and guarantee, particularly with nonlinear function approximation and off-policy sampling (Bhatnagar et al., 2009; Sutton et al., 2009a,b; Tsitsiklis and Van Roy, 1996; Watkins and Dayan, 1992). With the mainstream application of neural network function approximations in value-based approaches (Gu et al., 2016b; Lillicrap et al., 2016; Mnih et al., 2015), a series of techniques have been introduced to combat such instability, such as replay buffer (Lin, 1992; Mnih et al., 2015; Schaul et al., 2015b), exploration heuristics (Gal and Ghahramani, 2016; Osband et al., 2016), target networks (Lillicrap et al., 2016; Mnih et al., 2015), advantage-value decomposition (Gu et al., 2016b; Wang et al., 2016), conservative target value estimators (Fujimoto et al., 2018; Hasselt, 2010; Van Hasselt et al., 2015), mixing multi-step returns (Munos et al., 2016), and soft value iteration (Haarnoja et al., 2017; Rawlik et al., 2013). We will not go through these details and instead focus on their basic forms to highlight how they relate to model-based and policy-based in later sections.

Q-Learning

Q-learning (Watkins and Dayan, 1992) is one of the most popular value-based algorithms and makes use of the Bellman optimality equation defined for Q^*,

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \cdots \sim P, \pi^* | s_t, a_t} \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'}) \right] \tag{2.15}$$

$$= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p(\cdot | s_t, a_t)} \left[\max_a Q^*(s_{t+1}, a) \right]. \tag{2.16}$$

A practical implementation for deep learning optimizes a parameterized Q-function Q_w, a deep Q-network (DQN) (Mnih et al., 2015), using stochastic gradient descent (SGD) on the following objective,

$$L(w) = \mathbb{E}_{s_t, a_t, s_{t+1} \sim P, \beta} \left[(Q_w(s_t, a_t) - y_t)^2 \right] \tag{2.17}$$

$$y_t = r(s_t, a_t) + \gamma \max_a Q'(s_{t+1}, a), \tag{2.18}$$

where Q' denotes the target network which syncs with the main Q_w at some fixed intervals (Mnih et al., 2015) or smoothly tracks Q_w (Lillicrap et al., 2016). β represents sampling from replay buffer (Lin, 1992; Mnih et al., 2015), whose policy is effectively a mixture of all exploration policies, which usually adapt with changing Q_w. As noted in Section 2.2.2, many techniques have been devised to stablize Q-learning with neural network function approximations.

For discrete action spaces, Eq. 2.17 can be optimized straightforwardly; however, for continuous action space, the $\max_a$ operator makes the target value computation difficult.

Actor-critic methods (Sutton et al., 1999a) are thus typically proposed for value-based approaches in continuous action spaces. In Chapter 3, we discuss how deep Q-learning can be directly extended to continuous action space by adopting a specific parameterization of Q-functions.

2.2.3 Policy-based Algorithms

Policy-based (PG) approaches, and in particular policy gradient methods, are model-free algorithms and considered the most direct way to find the optimal policy, since they do not involve surrogate objectives as in model-based or value-based approaches. They directly apply gradient ascent with respect to the RL objective $J(\pi)$ in Eq. 2.2. Specifically, for a parameterized stochastic policy π_θ, the gradient of $J(\pi) = J(\theta)$ with respect to θ is given by the following, where $\tau_{t+1} = \{s_{t+1:\infty}, a_{t+1:\infty}\}$ denotes a partial trajectory of states and actions, $\hat{Q}(s_t, a_t, \tau_{t+1}) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'})$ denotes γ-discounted cumulative rewards, $p_t^\pi(s_t)$ is the marginal state distribution at time t for policy π, and $\rho^\pi(s) = \sum_{t=0}^{\infty} \gamma^t p_t^\pi(s_t = s)$ is the unnormalized γ-discounted state visitation frequency,

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{s_0,a_0,s_1,\cdots\sim P,\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t,a_t)\right] \tag{2.19}$$

$$= \mathbb{E}_{s_0,a_0,s_1,\cdots\sim P,\pi_\theta}\left[\left(\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t=0}^{\infty} \gamma^t r(s_t,a_t)\right)\right] \tag{2.20}$$

$$= \mathbb{E}_{s_0,a_0,s_1,\cdots\sim P,\pi_\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{t'=t}^{\infty} \gamma^{t'} r(s_{t'},a_{t'})\right)\right] \tag{2.21}$$

$$= \mathbb{E}_{s_0,a_0,s_1,\cdots\sim P,\pi_\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \gamma^t \hat{Q}(s_t,a_t,\tau_{t+1})\right] \tag{2.22}$$

$$= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_t\sim p_t^\pi(\cdot),a_t\sim\pi_\theta(\cdot|s_t),\tau_{t+1}\sim P,\pi_\theta|s_t,a_t}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{Q}(s_t,a_t,\tau_{t+1})\right] \tag{2.23}$$

$$= \mathbb{E}_{s_t\sim\rho^\pi(\cdot),a_t\sim\pi_\theta(\cdot|s_t),\tau_{t+1}\sim P,\pi_\theta|s_t,a_t}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{Q}(s_t,a_t,\tau_{t+1})\right] \tag{2.24}$$

$$= \mathbb{E}_{s_t\sim\rho^\pi(\cdot),a_t\sim\pi_\theta(\cdot|s_t)}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) Q^\pi(s_t,a_t)\right] \tag{2.25}$$

$$= \mathbb{E}_{s_t\sim\rho^\pi(\cdot)}\left[\nabla_\theta \mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)} Q^\pi(s_t,a_t)\right]. \tag{2.26}$$

The last line of derivation is based on the definition of Q^π. A subtle detail to note is that in practical algorithms, we do not use state samples s from $\rho^\pi(s)$, but instead use normalized undiscounted state visitation frequency $\bar{\rho}^\pi(s) \propto \sum_{t=0}^{\infty} p_t^\pi(s_t = s)$. Thomas (2014) discusses that this induces bias in the gradient estimation; however, this bias is essential for good,

practical performance. In the rest of this thesis, we only discuss additional biases on top of the practical implementation bias, and also interchangeably use these two definitions of state visitation distribution with ρ^π. Note that to make the original discounted definition of ρ^π normalized, we simply multiply by $1-\gamma$.

2.3 On-Policy and Off-Policy Algorithms

An RL algorithm is considered *on-policy* if it mainly uses transition samples from the current policy π to improve the policy at each iteration. By contrast, an *off-policy* algorithm can reuse samples from any policy (past policies or arbitrary behavior policies) to make effective policy improvements on the same batch of data. In this section, we use the policy gradient derivation in Section 2.2.3 as a starting point and introduce its on-policy and off-policy variants.

2.3.1 On-Policy Likelihood Ratio Policy Gradient

Policy gradients derived in Eq. 2.19 are on-policy, since their estimations involve expectation over transitions from policy π. In particular, we call the gradient estimator in Eq. 2.24 *Monte Carlo policy gradient estimator* or *likelihood ratio policy gradient estimator*, since it relies on Monte Carlo estimate of the Q-function $\hat{Q}(s,a,\tau)$ and a likelihood ratio or score function gradient estimator. It is the basis of popular on-policy policy gradient algorithms such as REINFORCE (Williams, 1992), trust-region policy optimization (TRPO) (Schulman et al., 2015a), asynchronous advantage actor-critic (A3C) (Mnih et al., 2016), proximal policy optimization (PPO) (Schulman et al., 2017). In practice, a number of techniques are required to make this estimator lower variance, such as baselines (Weaver and Tao, 2001; Williams, 1992) and generalized advantage estimation (Schulman et al., 2016). State-dependent baselines (Mnih and Gregor, 2014; Schulman et al., 2015a; Williams, 1992) are particularly appealing since they do not introduce bias (Williams, 1992) and can be trained easily with neural function approximation. Specifically, the state-dependent baseline is applied as,

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t\sim\rho^\pi(\cdot),a_t\sim\pi_\theta(\cdot|s_t),\tau_{t+1}\sim P,\pi_\theta|s_t,a_t}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\hat{Q}(s_t,a_t,\tau_{t+1}) - V_\phi(s_t)\right)\right], \tag{2.27}$$

where V_ϕ is learned to approximate V^π. Empirically, this simple unbiased technique leads to substantial variance reduction (Tucker et al., 2018; Weaver and Tao, 2001; Williams, 1992). While V^π is not the baseline that can optimally reduce the variance (Weaver and Tao, 2001),

its estimation is generally simpler and is a preferred choice in practical Monte Carlo policy gradient algorithms (Mnih et al., 2016; Schulman et al., 2015a, 2017).

The most appealing aspect of on-policy Monte Carlo policy gradient methods is their simplicity and stability. Because they do not rely on surrogate objectives as optimized in value-based or model-based approaches and use unbiased Monte Carlo estimate $\hat{Q}(s,a,\tau)$, their convergence properties are easier to analyze and optimization algorithms with provable policy improvement guarantees have been devised (Gu et al., 2017c; Kakade, 2001; Schulman et al., 2015a). The high-variance nature of the Monte Carlo estimation can be alleviated by taking large batch of samples per policy gradient update, or trading it off with slight bias in gradient estimation that usually does not cause performance degradation in practice (Gu et al., 2017c; Mnih et al., 2016; Schulman et al., 2016). The downside of these Monte Carlo methods is that they often cannot utilize off-policy samples effectively without introducing non-trivial bias, since naíve importance correction does not scale well with action and temporal dimensions (Levine and Koltun, 2013; Thomas and Brunskill, 2016). This makes the Monte Carlo policy gradient algorithms very data intensive. However, these algorithms directly learn π^* and the approximation of V^π is only used for variance reduction, so they can generally use much simpler function approximations, such as smaller neural networks or linear function approximation, than in value-based or model-based approaches (Duan et al., 2016; Gu et al., 2017c; Lillicrap et al., 2016; Rajeswaran et al., 2017). Such simplicity may sometimes make Monte Carlo policy gradient still a preferred method of choice.

2.3.2 Off-Policy Expected Actor-Critic

To arrive at an off-policy policy gradient algorithm, we again start from the derivation in Eq. 2.19. In contrast to Eq. 2.24, we call the gradient estimator in Eq. 2.26 *deterministic policy gradient estimator* (Gu et al., 2017b,c), *expected policy gradient estimator* (Asadi et al., 2017; Ciosek and Whiteson, 2018; Degris et al., 2012), or *all-action policy gradient estimator* (Sutton, 2000), since it uses a critic $Q^\pi(s,a)$ to compute analytic gradient integrated over immediate action a. Importantly, the critic Q^π can be estimated using off-policy temporal difference learning. Such off-policy policy evaluation provides the foundation for deriving off-policy actor-critic algorithms.

Actor-critic methods (Sutton et al., 1999a) optimize a value function Q^π or V^π, the *critic*, and a policy function π, the *actor*, iteratively. They are model-free methods and considered an hybrid approach between value-based and policy-based. The core of the idea is captured in policy iteration algorithm (Howard, 1964), which uses policy evaluation to estimate Q^π or V^π and uses policy improvement to improve the policy π locally with respect to the action-value evaluation. Classically, actor-critic algorithms are solved with analytic

updates, such as least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003; Lagoudakis et al., 2002), which iterates between least-squares temporal difference (LSTD) (Bradtke and Barto, 1996) solve of $Q^{\pi^n}(s,a)$ given policy π^n at iteration n, and analytic update of the actor $\pi^{n+1}(a|s) = \delta(a = \arg\max_{a'} Q^{\pi^n}(s,a'))$. Similarly, with neural network function approximations, the algorithm alternates between the following two minimization objectives with respect to Q_w and a parameterized policy π_θ,

$$L(w) = \mathbb{E}_{s_t,a_t,s_{t+1}\sim P,\beta}\left[\left(Q_w(s_t,a_t) - y_t\right)^2\right] \tag{2.28}$$

$$y_t = r(s_t,a_t) + \gamma \mathbb{E}_{a\sim\pi_\theta(\cdot|s_{t+1})} Q'(s_{t+1},a) \tag{2.29}$$

$$L(\theta) = -\mathbb{E}_{s_t\sim P,\beta}\left[\mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)} Q_w(s_t,a_t)\right] \tag{2.30}$$

Since optimizing each objective is expensive, practical algorithms often perform a small number of SGD updates per objective for iteratively learning Q_w and π_θ. In the limiting case with a deterministic policy $\pi_\theta(a|s) = \delta(a = \mu_\theta(s))$, we get deterministic policy gradient (DPG) algorithm (Lever, 2014; Lillicrap et al., 2016). If we use a stochastic policy (Gu et al., 2017c; Haarnoja et al., 2018; Heess et al., 2015a), the gradient for the negative policy loss is given below, where we use ρ^β to denote state visitation frequency under policy β,

$$-\nabla_\theta L(\theta) = \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_\theta \mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)} Q_w(s_t,a_t)\right] \tag{2.31}$$

$$= \mathbb{E}_{s_t\sim\rho^\beta(\cdot),a_t\sim\pi_\theta(\cdot|s_t)}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) Q_w(s_t,a_t)\right]. \tag{2.32}$$

The second line is unnecessary in most cases, since if the action is discrete, analytic integration with respect to action is usually tractable, and if the action is continuous and the policy a simple distribution such as Gaussian, reparameterization trick (Kingma and Welling, 2014) can be used to compute a lower variance estimator. However, we explicit write this using the score function estimator form to illuminate the closeness of actor-critic gradient with Monte Carlo policy gradient in the next section.

As we contrast Eq. 2.26 with Eq. 2.31, we observe that the biases in off-policy actor-critic gradient only come from two sources: approximation of Q^π through Q_w and off-policy state sampling using ρ^β instead of ρ^π. In Chapter 4, we show that we can derive theoretical bounds on policy gradient biases in actor-critic methods based on those two factors, and empirically demonstrate that by changing design choices in off-policy actor-critic algorithms, we can better model the stable learning behavior of on-policy Monte Carlo policy gradient algorithms with significantly improved sample efficiency.

An intriguing property of such stochastic gradient actor-critic algorithm is that by simply changing the SGD update ratio of the two objectives, it can interpolate between Q-learning

and policy gradient, discussed in Section 2.2.2 and Chapter 2.2.3 respectively. Specifically, if we have many actor updates per critic update, the actor is effectively amortizing $\arg\max_a Q^\pi(s,a)$ at each iteration and the learning dynamics is similar to that of Q-learning. If we have many critic updates per actor update, the critic is doing full policy evaluation to estimate Q^π, and the actor is updated using a gradient estimate resembling that of classic policy gradient. We will discuss such connection more in the next chapter on policy gradient. In practice, these algorithms often use simple one-to-one update ratio (Lillicrap et al., 2016), making the analysis of learning dynamics difficult. In Chapter 4, we discuss how a variant of our interpolated policy gradient (IPG) algorithm, which does 1-to-5000 actor-to-critic updates, performs much more stably than prior methods, possibly due to its closer resemblance to the stable policy gradient algorithm.

Chapter 3

Continuous Deep Q-Learning with Model-based Acceleration

In this chapter, we derive a variant of Q-learning (Watkins and Dayan, 1992) that can be used in continuous domains. Model-free reinforcement learning in domains with continuous actions is typically handled with policy search methods (Peters et al., 2010; Peters and Schaal, 2006). Integrating value function estimation into these techniques results in actor-critic algorithms (Hafner and Riedmiller, 2011; Lillicrap et al., 2016; Schulman et al., 2015a), which combine the benefits of policy search and value function estimation, but at the cost of training two separate function approximators with respect to different optimization objectives. Our proposed Q-learning algorithm for continuous domains, which we call normalized advantage functions (NAF), avoids the need for a second actor or policy function, resulting in a simpler algorithm. The simpler optimization objective and the choice of value function parameterization result in an algorithm that is substantially more sample-efficient when used with large neural network function approximators on a range of continuous control domains.

Beyond deriving an improved model-free deep reinforcement learning algorithm, we also seek to incorporate elements of model-based RL to accelerate learning, without giving up the strengths of model-free methods. One approach is for off-policy algorithms such as Q-learning to incorporate off-policy experience produced by a model-based planner. However, while this solution is a natural one, our empirical evaluation shows that it is ineffective at accelerating learning. As we discuss in our evaluation, this is due in part to the nature of value function estimation algorithms, which must experience both good and bad state transitions to accurately model the value function landscape. We propose an alternative approach to incorporating learned models into our continuous-action Q-learning algorithm based on *imagination rollouts*: on-policy samples generated under the learned model, analogous to the Dyna-Q method (Sutton, 1990). We show that this is extremely effective when the learned

dynamics model perfectly matches the true one, but degrades dramatically with imperfect learned models. However, we demonstrate that iteratively fitting local linear models (Li and Todorov, 2004) to the latest batch of on-policy or off-policy rollouts provides sufficient *local* accuracy to achieve substantial improvement using short imagination rollouts in the vicinity of the real-world samples.

Lastly, we include a demonstration of an asynchronous variant of our NAF algorithm running across a cluster of robots. We demonstrate that, contrary to commonly held assumptions about sample inefficiency of model-free algorithms, off-policy deep Q-function based algorithms such as NAF can achieve training times that are suitable for real robotic systems. Our real world experiments show that our approach can be used to learn a door opening skill from scratch under 2.5 hours with 2 robot arms, using only general-purpose neural network representations and without any human demonstrations. To the best of our knowledge, this is the first demonstration of autonomous door opening that does not use human-provided examples for initialization.

3.1 Normalized Advantage Functions

Q-learning (Watkins and Dayan, 1992) requires evaluating $\arg\max_a Q(s,a)$ in order to compute the update target and derive the optimal policy, as shown in Eq. 3.1. When action a is continuous and Q is a complex function approximation such as a neural network, this optimization is generally expensive or intractable. A naïve approach to get around this problem is to discretize the continuous action space; however, this approach scales poorly with the action dimension since using B bins per dimension for N-dimensional action space creates B^N discrete actions (Lillicrap et al., 2016).

$$\min_\theta \mathbb{E}_{s_t,a_t,s_{t+1}\sim\beta}[(Q_\theta(s_t,a_t)-y_t)^2], \quad y_t = r_t + \gamma \max_a Q'(s_{t+1},a) \tag{3.1}$$

$$\mu_\theta(s_t) = \arg\max_a Q_\theta(s_t,a) \tag{3.2}$$

We propose a simple method to enable Q-learning in continuous action spaces with deep neural networks, which we refer to as normalized advantage functions (NAF). The idea behind normalized advantage functions is to represent the Q-function $Q(s_t,a_t)$ in Q-learning in such a way that its maximum with respect to a_t, $\arg\max_{a_t} Q(s_t,a_t)$, can be determined easily and analytically during the Q-learning update. While a number of representations are possible that allow for analytic maximization, the one we use in our implementation is based on a neural network that separately outputs a value function term $V(s)$ and an advantage term

$A(s,a)$, which is parameterized as a quadratic function of nonlinear features of the state:

$$\begin{aligned} Q(s,a|\theta^Q) &= A(s,a|\theta^A) + V(s|\theta^V) \\ A(s,a|\theta^A) &= -\frac{1}{2}(a-\mu(s|\theta^\mu))^T P(s|\theta^P)(a-\mu(s|\theta^\mu)) \end{aligned} \tag{3.3}$$

$P(s|\theta^P)$ is a state-dependent, positive-definite square matrix, which is parametrized by $P(s|\theta^P) = L(s|\theta^P)L(s|\theta^P)^T$, where $L(s|\theta^P)$ is a lower-triangular matrix whose entries come from a linear output layer of a neural network, with the diagonal terms exponentiated. While this representation is more restrictive than a general neural network function, since the Q-function is quadratic in a, the action that maximizes the Q-function is always given by $\mu(s|\theta^\mu)$. We use this representation with a deep Q-learning algorithm analogous to Mnih et al. (2015), using target networks and a replay buffers as described by Lillicrap et al. (2016). NAF, given by Algorithm 1, is considerably simpler than DDPG.

Algorithm 1 Continuous Q-Learning with NAF

Randomly initialize normalized Q network $Q(s,a|\theta^Q)$.
Initialize target network Q' with weight $\theta^{Q'} \leftarrow \theta^Q$.
Initialize replay buffer $R \leftarrow \emptyset$.
for episode=1,M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $s_1 \sim p(s_1)$
  for t=1,T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$
    Execute a_t and observe r_t and s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    for iteration=1,I **do**
      Sample a random minibatch of m transitions from R
      Set $y_i = r_i + \gamma V'(s_{i+1}|\theta^{Q'})$
      Update θ^Q by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i,a_i|\theta^Q))^2$
      Update the target network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    end for
  end for
end for

Decomposing Q into an advantage term A and a state-value term V was suggested by Baird III (1993); Harmon and Baird III (1996), and was recently explored by Wang et al. (2016) for discrete action problems. Normalized action-value functions have also been proposed by Rawlik et al. (2013) in the context of an alternative temporal difference learning algorithm. However, our method is the first to combine such representations with deep neural networks into an algorithm that can be used to learn policies for a range of challenging

continuous control tasks. In general, A does not need to be quadratic, and exploring other parametric forms such as multimodal distributions is an interesting avenue for future work.

3.1.1 Locally-Invariant Exploration for Normalized Advantage Functions

Exploration is an essential component of reinforcement learning algorithms. The simplest and most common type of exploration involves randomizing the actions according to some distribution, either by taking random actions with some probability (Mnih et al., 2015), or adding Gaussian noise in continuous action spaces (Schulman et al., 2015a). However, choosing the magnitude of the random exploration noise can be difficult, particularly in high-dimensional domains where different action dimensions require very different exploration scales. Furthermore, independent (spherical) Gaussian noise may be inappropriate for tasks where the optimal behavior requires correlation between action dimensions, as for example in the case of the swimming snake described in our experiments, which must coordinate the motion of different body joints to produce a synchronized undulating gait.

The NAF provides us with a simple and natural avenue to obtain an adaptive exploration strategy, analogously to Boltzmann exploration. The idea is to use the matrix in the quadratic component of the advantage function as the precision for a Gaussian action distribution. This naturally causes the policy to become more deterministic along directions where the advantage function varies steeply, and more random along directions where it is flat (Ciosek and Whiteson, 2018; Haarnoja et al., 2017; Heess et al., 2012). The corresponding policy is given by

$$\begin{aligned} \pi(a|s) &= \exp^{Q(s,a|\theta^Q)} / \int \exp^{Q(s,a|\theta^Q)} da \\ &= \mathcal{N}(\mu(s|\theta^\mu), cP(s|\theta^P)^{-1}). \end{aligned} \tag{3.4}$$

Previous work also noted that generating Gaussian exploration noise independently for each time step was not well-suited for many continuous control tasks, particularly simulated robotic tasks where the actions correspond to torques or velocities (Lillicrap et al., 2016). The intuition is that, as the length of the time-step decreases, temporally independent Gaussian exploration will cancel out between time steps. Instead, prior work proposed to sample noise from an Ornstein-Uhlenbech (OU) process to generate a temporally correlated noise sequence (Lillicrap et al., 2016). We adopt the same approach in our work, but sample the innovations for the OU process from the Gaussian distribution in Equation 3.4. Lastly, we note that the overall scale of $P(s|\theta^P)$ could vary significantly through the learning, and depends on the

magnitude of the cost, which introduces an undesirable additional degree of freedom. We therefore use a heuristic adaptive-scaling trick to stabilize the noise magnitudes.

3.2 Accelerating Model-free Learning with Model-based Rollouts

While NAF provides some advantages over actor-critic model-free RL methods in continuous domains, we can improve their data efficiency substantially under some additional assumptions by exploiting learned models. We will show that incorporating a particular type of learned model into Q-learning with NAFs significantly improves sample efficiency, while still allowing the final policy to be finetuned with model-free learning to achieve good performance without the limitations of imperfect models.

3.2.1 Model-Guided Exploration

One natural approach to incorporating a learned model into an off-policy algorithm such as Q-learning is to use the learned model to generate good exploratory behaviors using planning or trajectory optimization. To evaluate this idea, we utilize the iLQG algorithm (Tassa et al., 2012) to generate good trajectories under the model, and then mix these trajectories together with on-policy experience by appending them to the replay buffer. Note that all experiences are still collected from the real environment, except the exploration policy includes an iLQG component from model-based planning. Interestingly, we show in our evaluation that, even when such exploration policy is close to optimal, e.g. planning under the perfect model, the improvement obtained from this approach is often quite small, and varies significantly across domains and choices of exploration noise. The intuition behind this result is that off-policy iLQG exploration is too different from the learned policy, and Q-learning must consider alternatives in order to ascertain the optimality of a given action. That is, it's not enough to simply show the algorithm *good* actions, it must also experience bad actions to understand which actions are better and which are worse.

3.2.2 Imagination Rollouts

As discussed in the previous section, incorporating off-policy exploration from good, narrow distributions, such as those induced by iLQG, often does not result in significant improvement for Q-learning. These results suggest that Q-learning, which learns a policy based on minimizing temporal differences, inherently requires noisy on-policy actions to succeed. In

real-world domains such as robots and autonomous vehicles, this can be undesirable for two reasons: first, it suggests that large amounts of on-policy experience are required in addition to good off-policy samples, and second, it implies that the policy must be allowed to make "its own mistakes" during training, which might involve taking undesirable or dangerous actions that can damage real-world hardware.

One way to avoid these problems while still allowing for a large amount of on-policy exploration is to generate synthetic on-policy trajectories under a learned model. Adding these synthetic samples, which we refer to as *imagination rollouts*, to the replay buffer effectively augments the amount of experience available for Q-learning. The particular approach we use is to perform rollouts in the real world using a mixture of planned iLQG trajectories and on-policy trajectories, with various mixing coefficients evaluated in our experiments, and then generate additional synthetic on-policy rollouts using the learned model from each state visited along the real-world rollouts. This approach can be viewed as a variant of the Dyna-Q algorithm (Sutton, 1990). However, while Dyna-Q has primarily been used with small and discrete systems, we show that using iteratively refitted linear models allows us to extend the approach to deep reinforcement learning on a range of continuous control domains. In some scenarios, we can even generate all or most of the real rollouts using off-policy iLQG controllers, which is desirable in safety-critic domains where poorly trained policies might take dangerous actions. The algorithm is given as Algorithm 2, and is an extension on Algorithm 1 combining model-based RL.

Imagination rollouts can suffer from severe bias when the learned model is inaccurate. For example, we found it very difficult to train nonlinear neural network models for the dynamics that would actually improve the efficiency of Q-learning when used for imagination rollouts. As discussed in the following section, we found that using iteratively refitted time-varying linear dynamics produced substantially better results. In either case, we would still like to preserve the generality and optimality of model-free RL while deriving the benefits of model-based learning. To that end, we observe that most of the benefit of model-based learning is derived in the early stages of the learning process, when the policy induced by the neural network Q-function is poor. As the Q-function becomes more accurate, on-policy behavior tends to outperform model-based controllers. We therefore propose to switch off imagination rollouts after a given number of iterations.[1] In this framework, the imagination rollouts can be thought of as an inexpensive way to pretrain the Q-function, such that fine-tuning using real world experience can quickly converge to an optimal solution.

[1] In future work, it would be interesting to select this iteration adaptively based on the expected relative performance of the Q-function policy and model-based planning.

Algorithm 2 Imagination Rollouts with Fitted Dynamics and Optional iLQG Exploration

Randomly initialize normalized Q network $Q(s,a|\theta^Q)$.
Initialize target network Q' with weight $\theta^{Q'} \leftarrow \theta^Q$.
Initialize replay buffer $R \leftarrow \emptyset$ and fictional buffer $R_f \leftarrow \emptyset$.
Initialize additional buffers $B \leftarrow \emptyset, B_{old} \leftarrow \emptyset$ with size nT.
Initialize fitted dynamics model $\mathcal{M} \leftarrow \emptyset$.
for $episode = 1, M$ **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  Select $\mu'(s_t,t)$ from $\{\mu(s_t|\theta^\mu), \pi_t^{iLQG}(a_t|s_t)\}$ with probabilities $\{p, 1-p\}$
  for $t = 1, T$ **do**
    Select action $a_t = \mu'(s_t,t) + \mathcal{N}_t$
    Execute a_t and observe r_t and s_{t+1}
    Store transition $(s_t, a_t, r_t, s_{t+1}, t)$ in R and B
    if mod $(episode \cdot T + t, m) = 0$ and $\mathcal{M} \neq \emptyset$ **then**
      Sample m $(s_i, a_i, r_i, s_{i+1}, i)$ from B_{old}
      Use $\mathcal{M}$ and $\mu(s|\theta^\mu)$ with noise $\mathcal{N}'$ to simulate l steps from each sampled state s_i
      Store all fictional transitions in R_f
    end if
    Sample a random minibatch of m transitions $I \cdot l$ times from R_f and I times from R, and update $\theta^Q, \theta^{Q'}$ as in Algorithm 1 per minibatch.
  end for
  if B_f is full **then**
    $\mathcal{M} \leftarrow$ FitLocalLinearDynamics(B_f) (see Section 3.2.3)
    $\pi^{iLQG} \leftarrow$ iLQG_OneStep($B_f, \mathcal{M}$) (see Section 2.2.1)
    $B_{old} \leftarrow B_f, B_f \leftarrow \emptyset$
  end if
end for

3.2.3 Fitting the Dynamics Model

In order to obtain good imagination rollouts and improve the efficiency of Q-learning, we needed to use an effective and data-efficient model learning algorithm. While prior methods propose a variety of model classes, including neural networks (Heess et al., 2015a), Gaussian processes (Deisenroth and Rasmussen, 2011), and locally-weighted regression (Atkeson et al., 1997), we found that we could obtain good results by using iteratively refitted time-varying linear models, as proposed by Levine and Abbeel (2014). In this approach, instead of learning a good global model for all states and actions, we aim only to obtain a good local model around the latest set of samples. This approach requires a few additional assumptions: namely, it requires the initial state to be either deterministic or low-variance Gaussian, and it requires the states and actions to all be continuous. To handle domains with more varied initial states, we can use a mixture of Gaussian initial states with separate time-varying linear models for each one. The model itself is given by $p_t(s_{t+1}|s_t,a_t) = \mathcal{N}(\mathbf{F}_t[s_t;a_t]+\mathbf{f}_t,\mathbf{N}_t)$. Every n episodes, we refit the parameters $\mathbf{F}_t$, $\mathbf{f}_t$, and $\mathbf{N}_t$ by fitting a Gaussian distribution at each time step to the vectors $[s_t^i;a_t^i;s_{t+1}^i]$, where i indicates the sample index, and conditioning this Gaussian on $[s_t;a_t]$ to obtain the parameters of the linear-Gaussian dynamics at that step. We use $n = 5$ in our experiments. Although this approach introduces additional assumptions beyond the standard model-free RL setting, we show in our evaluation that it produces impressive gains in sample efficiency on tasks where it can be applied.

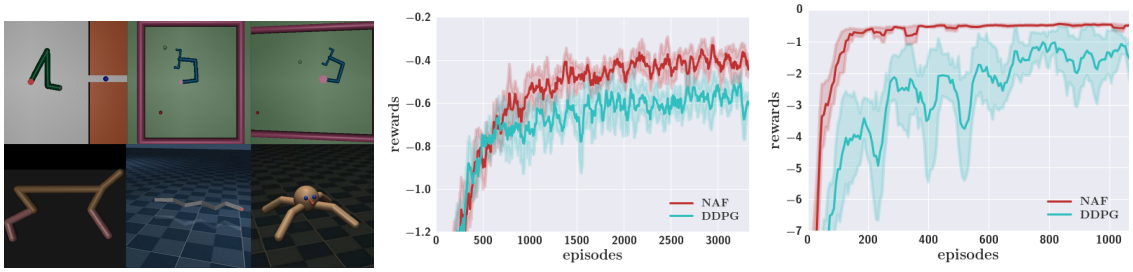
3.3 Experiments in Simulation

We evaluated our approach on a set of simulated robotic tasks using the MuJoCo simulator (Todorov et al., 2012). The tasks were based on the benchmarks described by Lillicrap et al. (2016), and are summarized in Table 3.1 and visualized in Figure 3.1a. Although we attempted to replicate the tasks in previous work as closely as possible, discrepancies in the simulator parameters and the contact model produced results that deviate slightly from those reported in prior work. In all experiments, the input to the policy consisted of the state of the system, defined in terms of joint angles and root link positions. Angles were often converted to sine and cosine encoding.

For both our method and the prior DDPG (Lillicrap et al., 2016) algorithm in the comparisons, we used neural networks with two layers of 200 rectified linear units (ReLU) to produce each of the output parameters – the Q-function and policy in DDPG, and the value function V, the advantage matrix L, and the mean μ for NAF. Since Q-learning was done with a replay buffer, we applied the Q-learning update 5 times per each step of experience

Domain	Description	Domain	Description
Cartpole	The classic cart-pole swing-up task. Agent must balance a pole attached to a cart by applying forces to the cart alone. The pole starts each episode hanging upside-down.	Reacher	Agent is required to move a 3-DOF arm from random starting locations to random target positions.
Peg	Agent is required to insert the tip of a 3-DOF arm from locally-perturbed starting locations to a fixed hole.	Gripper	Agent must use an arm with gripper appendage to grasp an object and manuver the object to a fixed target.
GripperM	Agent must use an arm with gripper attached to a moveable platform to grasp an object and move it to a fixed target.	Canada2d	Agent is required to use an arm with hockey-stick like appendage to hit a ball initialzed to a random start location to a random target location.
Cheetah	Agent should move forward as quickly as possible with a cheetah- like body that is constrained to the plane.	Swimmer6	Agent should swim in snake-like manner toward the fixed target using six joints, starting from random poses.
Ant	The four-legged ant should move toward the fixed target from a fixed starting position and posture.	Walker2d	Agent should move forward as quickly as possible with a bipedal walker constrained to the plane without falling down or pitching the torso too far forward or backward.

Table 3.1 List of domains. All the domains except ant are 2D.



(a) Example task domains.

(b) NAF and DDPG on multi-target reacher.

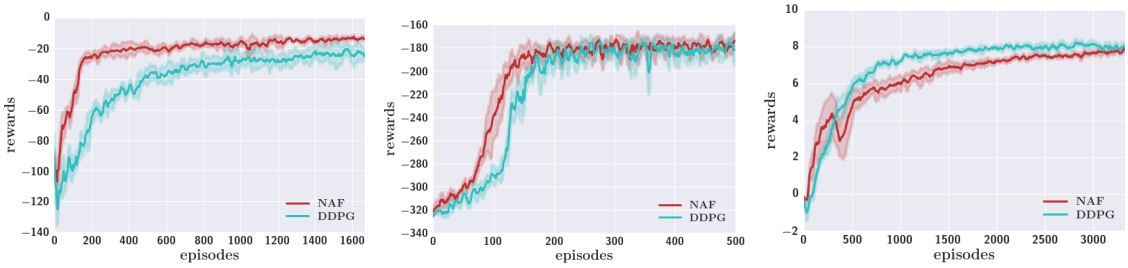
(c) NAF and DDPG on peg insertion.

Fig. 3.1 (a) Task domains: top row from left (manipulation tasks: peg, gripper, mobile gripper), bottom row from left (locomotion tasks: cheetah, swimmer6, ant). (b,c) NAF vs DDPG results on three-joint reacher and peg insertion. On reacher, the DDPG policy continuously fluctuates the tip around the target, while NAF stabilizes well at the target.

to accelerate learning ($I = 5$). To ensure a fair comparison, DDPG also updates both the Q-function and policy parameters 5 times per step.

3.3.1 Normalized Advantage Functions

In this section, we compare NAF and DDPG on 10 representative domains from Lillicrap et al. (2016), with three additional domains: a four-legged 3D ant, a six-joint 2D swimmer, and a



(a) NAF significantly outperforms DDPG on moving gripper.

(b) NAF converges faster than DDPG on swimmer6.

(c) NAF converges faster than DDPG on cheetah initially, but asymptoptically performs worse.

Fig. 3.2 NAF vs DDPG on three domains.

2D peg. We found the most sensitive hyperparameters to be presence or absence of batch normalization, base learning rate for Adam (Kingma and Ba, 2014) $\in \{1e^{-4}, 1e^{-3}, 1e^{-2}\}$, and exploration noise scale $\in \{0.1, 0.3, 1.0\}$. We report the best performance for each domain in Table 3.2. We were unable to achieve good results with the method of Rawlik et al. (2013) on our domains, likely due to the complexity of high-dimensional neural network function approximators.

Figure 3.1b and Figure 3.1c show the performances on the three-joint reacher, peg insertion, and a gripper with mobile base. While the numerical gap in reacher may be small, qualitatively there is also a very noticeable difference between NAF and DDPG. DDPG converges to a solution where the deterministic policy causes the tip to fluctuate continuously around the target, and does not reach it precisely. NAF, on the other hand, learns a smooth policy that makes the tip slow down and stabilize at the target. This difference is more noticeable in peg insertion and moving gripper, as shown by the much faster convergence rate to the optimal solution. Precision is very important in many real-world robotic tasks, and these result suggest that NAF may be preferred in such domains.

On locomotion tasks, the performance of the two methods is relatively similar. On the six-joint swimmer task and four-legged ant, NAF slightly outperforms DDPG in terms of the convergence speed; however, DDPG is faster on cheetah and finds a better policy on walker2d. The loss in performance of NAF can potentially be explained by downside of the mode-seeking behavior, where it is hard to explore other modes once the quadratic advantage function finds a good one. Choosing a parametric form that is more expressive than a quadratic could be used to address this limitation in future work.

Figures 3.2a, 3.2b, and 3.2c provide additional results on the comparison experiments between DDPG and NAF. NAF generally outperforms DDPG. In certain tasks that require

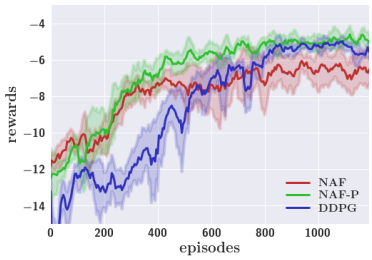


Fig. 3.3 NAF with exploration noise generated using the precision term (NAF-P) slightly outperforms the best DDPG result. Precision term is not used until step 50,000.

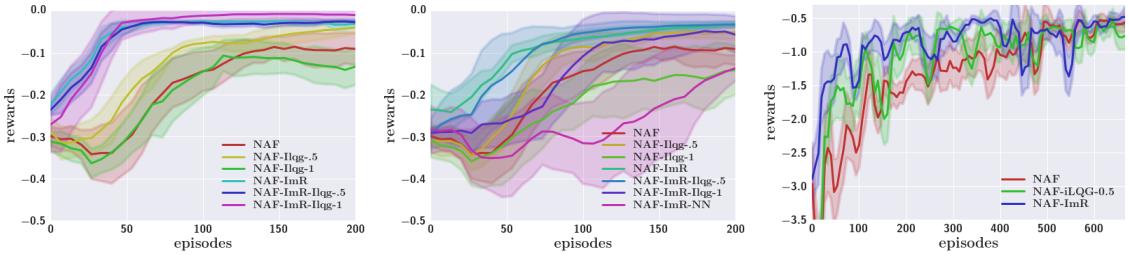
precision, such as peg insertion, the difference is very noticeable. However, there are also few cases where NAF underperforms DDPG. The most consistent of such cases is cheetah. While both DDPG and NAF enable cheetah to run decent distances, it is often observed that the cheetah movements learned in NAF are little less natural than those from DDPG. We speculate such behaviors come from a mode-seeking behavior of NAF, and exploring other parametric forms of NAF, such as multi-modal variants, is a promising avenue for future work.

Using the learned precision as the noise covariance for exploration allowed for convergence to a better policy on the "canada2d" task, which requires using an arm to strike a puck toward a target, as shown in Figure 3.3, but did not make a significant difference on the other domains.

The results on all of the domains are summarized in Table 3.2. Overall, NAF outperformed DDPG on the majority of tasks, particularly manipulation tasks that require precision and suffer less from the lack of multimodal Q-functions. This makes this approach particularly promising for efficient learning of real-world robotic tasks.

Domains	-	DDPG	episodes	NAF	episodes
Cartpole	-2.1	-0.601	420	-0.604	**190**
Reacher	-2.3	-0.509	1370	**-0.331**	**1260**
Peg	-11	-0.950	690	**-0.438**	**130**
Gripper	-29	1.03	2420	**1.81**	**1920**
GripperM	-90	-20.2	1350	**-12.4**	**730**
Canada2d	-12	-4.64	1040	**-4.21**	900
Cheetah	-0.3	**8.23**	**1590**	7.91	2390
Swimmer6	-325	-174	220	**-172**	**190**
Ant	-4.8	-2.54	2450	-2.58	**1350**
Walker2d	0.3	**2.96**	**850**	1.85	1530

Table 3.2 Best test rewards of DDPG and NAF policies, and the episodes it requires to reach within 5% of the best value. "-" denotes scores by a random agent.



(a) NAF on single-target reacher.

(b) NAF on single-target reacher.

(c) NAF on single-target gripper.

Fig. 3.4 Results on NAF with iLQG-guided exploration and imagination rollouts (a) using true dynamics (b,c) using fitted dynamics. “ImR" denotes using the imagination rollout with $l = 10$ steps on the reacher and $l = 5$ steps on the gripper. “iLQG-x" indicates mixing x fraction of iLQG episodes. Fitted dynamics uses time-varying linear models with sample size $n = 5$, except “-NN" which fits a neural network to global dynamics.

3.3.2 Evaluating Best-Case Model-Based Improvement with True Models

In order to determine how best to incorporate model-based components to accelerate model-free Q-learning, we tested several approaches using the ground truth dynamics, to control for challenges due to model fitting. We evaluated both of the methods discussed in Section 3.2: the use of model-based planning to generate good off-policy rollouts in the real world, and the use of the model to generate on-policy synthetic rollouts.

Figure 3.4a shows the effect of mixing off-policy iLQG experience and imagination rollouts on the three-joint reacher. It is noticeable that mixing the good off-policy experience does not significantly improve data-efficiency, while imagination rollouts always improve data-efficiency or final performance significantly. In the context of Q-learning, this result is not entirely surprising: Q learning must experience both good and bad actions in order to determine which actions are preferred, while the good model-based rollouts are so far removed from the policy in the early stages of learning that they provide little useful information. Figure 3.4a also evaluates two different variants of the imagination rollouts approach, where the rollouts in the real world are performed either using the learned policy, or using model-based planning with iLQG. In the case of this task, the iLQG rollouts achieve slightly better results, since the on-policy imagination rollouts sampled around these off-policy states provide Q-learning with additional information about alternative action not taken by the iLQG planner. In general, we did not find that off-policy rollouts were consistently better than on-policy rollouts across all tasks, but they did consistently produce good results. Performing

off-policy rollouts with iLQG may be desirable in real-world domains, where a partially learned policy might take undesirable or dangerous actions.

Domains	-	0.5	ImR	ImR,0.5	ImR,1
Reacher	-0.488	-0.449	-0.448	**-0.426**	-0.548
episodes	740	670	450	430	**90**
Canada2d	-6.23	-6.23	-5.89	**-5.88**	-12.0
episodes	1970	1580	570	**140**	210
Cheetah	7.00	7.10	**7.36**	7.29	6.43
episodes	580	1080	590	740	**390**

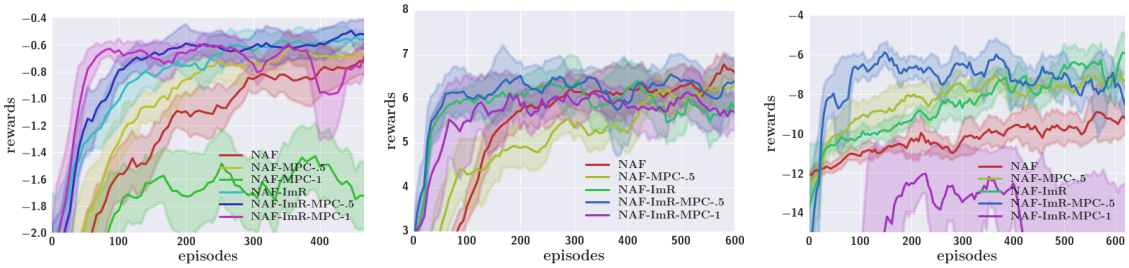
Table 3.3 Best-case model-based acceleration with true dynamics models. Best test rewards of NAF policies (first row), and the episodes it required to reach 5% of the best value (second row). “0.5" and “1" correspond to the fraction of MPC episodes. “ImR" means using imagination rollout with rollout length $l = 10$ for reacher, canada2d, and $l = 5$ for cheetah.

In this work, iLQG with true dynamics is used to generate guided exploration trajectories. While iLQG works for simple manipulation tasks with small number of initial states, it does not work well for random target reacher or complex locomotion tasks such as cheetah. We therefore run iLQG with replanning for the experiments reported in Figures 3.5c, 3.5b, and 3.5a, and Table 3.3. It is important to note that for those experiments, the hyper-parameters were fixed (batch normalization is on, learning rate is 10^{-3}, and exploration size is 0.3), and thus the results differ slightly from the experiments in the previous section.

In cheetah and other complex locomotion tasks, MPC policy is usually sub-optimal, and thus poor performance of mixing MPC experience in Figure 3.5b is expected. On the other hand, MPC policy works reasonably in hard manipulation tasks such as canada2d, and there is significant gain from mixing MPC experience as Figure 3.5c shows. However, the most consistent gain comes from using imagination rollouts in all three domains. In particular, Figure 3.5c shows that in canada2d, MPC experiences gives very good trajectories, i.e. those that hit balls in roughly the right directions, and doing rollouts can generate more of this useful experience, enabling canada2d to learn very quickly. While with true dynamics having the imagination experience directly means more experience and such result may be trivial, it is still important to see the benefits of rollouts which only explore up to $l = 10$ steps away from the real experience, as reported here. This is an interesting result, since this means the dynamics model only needs to be accurate around the data trajectories and this significantly lessens the requirement on fitted models.

3.3.3 Guided Imagination Rollouts with Fitted Dynamics

In this section, we evaluated the performance of imagination rollouts with learned dynamics. As seen in Figure 3.4b, we found that fitting time-varying linear models following the



(a) NAF on multi-target reacher. Insignificant gain from mixing MPC experience. Significant gain from imagination rollouts.

(b) NAF on cheetah. Great speeds up with imagination rollouts, no gain from mixing MPC experiences.

(c) NAF on canada2d. Very significant speed-ups from mixing MPC experiences, both with or without the rollouts.

Fig. 3.5 NAF on multi-target reacher, cheetah, and canada2d, with model-based acceleration using true dynamics: “ImR" denotes using the imagination rollout, $l = 10$ steps. “MPC-x" indicates mixing x fraction of MPC episodes.

imagination rollout algorithm is substantially better than fitting neural network dynamics models for the tasks we considered. There is a fundamental tension between efficient learning and expressive models like neural nets. We cannot hope to learn useful neural network models with a small number of samples for complex tasks, which makes it difficult to acquire a good model with fewer samples than are necessary to acquire a good policy. While the model is trained with supervised learning, which is typically more sample efficient, it often needs to represent a more complex function (e.g. rigid body physics). However, having such expressive models is more crucial as we move to improve model accuracy. Figure 3.4b presents results that compare fitted neural network models with the true dynamics when combined with imagination rollouts. These results indicate that the learned neural network models negate the benefits of imagination rollouts on our domains.

To evaluate imagination rollouts with fitted time-varying linear dynamics, we chose single-target variants of two of the manipulation tasks: the reacher and the gripper task. The results are shown in Figure 3.4b and 3.4c. We found that imagination rollouts of length 5 to 10 were sufficient for these tasks to achieve significant improvement over the fully model-free variant of NAF.

Adding imagination rollouts in these domains provided 2-5 factors of improvement in data efficiency. In order to retain the benefit of model-free learning and allow the policy to continue improving once it exceeds the quality possible under the learned model, we switch off the imagination rollouts after 130 episodes (20,000 steps) on the gripper domain. This produces a small transient drop in the performance of the policy, but the results quickly improve again. Switching off the imagination rollouts also ensures that Q-learning does not

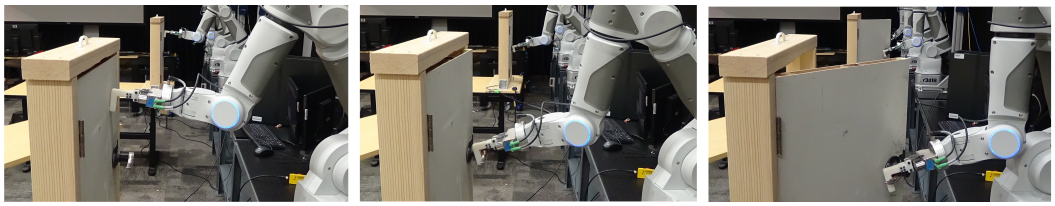


Fig. 3.6 Two robots learning to open doors using asynchronous NAF. The final policy learned with two workers could achieve a 100% success rate on the task across 20 consecutive trials.

diverge after it reaches good values, as were often observed in the gripper. This suggests that imagination rollouts, in contrast to off-policy exploration discussed in the previous section, is an effective method for bootstrapping model-free deep RL.

It should be noted that, although time-varying linear models combined with imagination rollouts provide a substantial boost in sample efficiency, this improvement is provided at some cost in generality, since effective fitting of time-varying linear models requires relatively small initial state distributions. With more complex initial state distributions, we might cluster the trajectories and fit multiple models to account for different modes. Extending the benefits of time-varying linear models to less restrictive settings is a promising direction and build on prior work (Fu et al., 2015; Levine et al., 2016). That said, our results show that imagination rollouts are a very promising approach to accelerating model-free learning when combined with the right kind of dynamics model.

3.4 Experiments on Real-World Robots

The real-world experiments are conducted with the 7-DoF arm shown in Figure 3.6. For parallel data collection and training on multiple robot arms, we used an asynchronous variant of NAF. The algorithm implementation and experimental details are discussed in Gu et al. (2017a).

3.4.1 Random Target Reaching

In this experiment, we demonstrate the benefits from parallel training with real robots using the simple reaching task. We set up the same reaching experiment in the real world across up to four robots. Robots execute policies at 20 Hz, while the training thread simply updates the network continuously at approximately 100 Hz.

Figure 3.7 confirms that 2 or 4 workers significantly improves learning speed over 1 worker, though the gains on this simple task are not substantial past 2 workers. Importantly, when the training thread is not synchronized with the data collection thread and the data

Fig. 3.7 The 7-DoF arm random target reaching with asynchronous NAF on real robots. Note that 1 worker suffers in both learning speed and final policy performance.

collection is too slow, it may not just slow down learning but also hurt the final policy performance, as observed in the 1-worker case. Further discrepancies from the simulation may also be explained by physical discrepancies among different robots.

3.4.2 Door Opening

The previous section describes a real-world evaluation of asynchronous NAF and demonstrates that learning can be accelerated by using multiple workers. In this section, we describe a more complex door opening task. Door opening presents a practical application of robotic learning that involves complex and discontinuous contact dynamics. Previous work has demonstrated learning of door opening policies using example demonstration provided by a human expert (Kalakrishnan et al., 2011). In this work, we demonstrate that we can learn policies for pulling open a door from scratch using asynchronous NAF. The entire task required approximately 2.5 hours to learn with two workers learning simultaneously, and the final policy achieves 100% success rate evaluated across 20 consecutive trials. An illustration of this task is shown in Figure 3.6, and the supplementary video in Gu et al. (2017a) shows different stages in the learning process, as well as the final learned policy.

Figure 3.8 illustrates the difference in the learning process between one and two workers, where the horizontal axis shows the number of parameter updates. 100,000 updates correspond to approximately half an hour, with some delays incurred due to periodic policy evaluation, which is only used for measuring the reward for the plot. One worker required significantly more than 4 hours to achieve 100% success rate, while two workers achieved

Fig. 3.8 Learning curves for real-world door opening. Learning with two workers significantly outperforms the single worker, and achieves a 100% success rate in under 500,000 update steps, corresponding to about 2.5 hours of real time.

the same success rate in 2.5 hours. Qualitatively, the learning process goes through a set of stages as the robots learn the task, as illustrated by learning curves in Figure 3.8, where the plateau near reward=0 corresponds to placing the hook near the handle, but not pulling the door open. In the first stage, the robots are unable to reach the handle, and explore the free space to determine an effective policy for reaching. Once the robots begin to contact the handle sporadically, they will occasionally pull on the handle by accident, but require additional training to be able to reach the handle consistently; this corresponds to the plateau in the learning curves. At this point, it becomes much easier for the robots to pull open the door, and a successful policy emerges. The final policy learned by the two workers was able to open the door every time, including in the presence of exploration noise.

3.5 Discussion

We explored several methods for improving the sample efficiency of model-free deep reinforcement learning. We first propose a method for applying standard Q-learning methods to high-dimensional, continuous domains, using the normalized advantage function (NAF) representation. This allows us to simplify the more standard actor-critic style algorithms, while preserving the benefits of nonlinear value function approximation, and allows us to employ a simple and effective adaptive exploration method. We show that, in comparison to recently proposed deep actor-critic algorithms, our method tends to learn faster and acquires more accurate policies. We further explore how model-free RL can be accelerated

by incorporating learned models, without sacrificing the optimality of the policy in the face of imperfect model learning. We show that, although Q-learning can incorporate off-policy experience, learning primarily from off-policy exploration (via model-based planning) only rarely improves the overall sample efficiency of the algorithm. We postulate that this caused by the need to observe both successful and unsuccessful actions, in order to obtain an accurate estimate of the Q-function. We demonstrate that an alternative method based on synthetic on-policy rollouts achieves substantially improved sample complexity, but only when the model learning algorithm is chosen carefully. We demonstrate that training neural network models does not provide substantive improvement in our domains, but simple iteratively refitted time-varying linear models do provide substantial improvement on domains where they can be applied.

We further presented an asynchronous variant of NAF can be used to learn complex robotic manipulation skills from scratch on real physical robotic manipulators. We demonstrate that our approach can learn a complex door opening task with only a few hours of training, and our simulated results demonstrate that training times decrease with more learners. Our technical contribution consists of a novel asynchronous version of the normalized advantage functions (NAF) deep reinforcement learning algorithm, as well as a number of practical extensions to enable safe and efficient deep reinforcement learning on physical systems, and our experiments confirm the benefits of nonlinear deep neural network policies over simpler shallow representations for complex robotic manipulation tasks.

Chapter 4

Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation

One of the simplest ways to learn a neural network policy is to collect a batch of behavior wherein the policy is used to act in the world, and then compute and apply a policy gradient update from this data. This is referred to as on-policy learning because all of the updates are made using data that was collected from the trajectory distribution induced by the current policy of the agent. It is straightforward to compute unbiased on-policy gradients, and practical on-policy gradient algorithms tend to be stable and relatively easy to use. A major drawback of such methods is that they tend to be data inefficient, because their gradient estimators have high variances.Off-policy algorithms based on Q-learning and actor-critic learning (Sutton et al., 1999a) have also proven to be an effective approach to deep RL such as in (Mnih et al., 2015) and (Lillicrap et al., 2016). Such methods reuse samples by storing them in a memory replay buffer and train a value function or Q-function with off-policy updates. This improves data efficiency, but often at a cost in stability and ease of use (Duan et al., 2016; Henderson et al., 2018). In essence, off-policy algorithms make better use of available data, but often at the cost of biased or high-variance estimates (Dudík et al., 2011; Jiang and Li, 2016) of the policy gradient that can lead to performance crashes.

We propose Interpolated Policy Gradient (IPG), a parameterized family of policy gradient methods that interpolate between on-policy and off-policy learning. Such methods are in general biased, but we show that the bias can be bounded. We show that a number of recent methods (Gu et al., 2017b; O'Donoghue et al., 2017; Wang et al., 2017) can be viewed as special cases of this more general family. Furthermore, our empirical results show that

β	ν	CV	Examples
-	0	No	REINFORCE (Williams, 1992),TRPO (Schulman et al., 2015b)
π	0	Yes	Q-Prop (Gu et al., 2017b)
-	1	-	DDPG (Lillicrap et al., 2016; Silver et al., 2014),SVG(0) (Heess et al., 2015a)

Table 4.1 Prior policy gradient method objectives as special cases of IPG.

in most cases, a mix of policy gradient and actor-critic updates achieves the best results, demonstrating the value of considering interpolated policy gradients.

4.1 Interpolated Policy Gradient

Interpolated policy gradient (IPG), mixes likelihood ratio gradient with $\hat{Q}(s,a,\tau)$, which provides unbiased but high-variance gradient estimation, and expected gradient (Asadi et al., 2017; Ciosek and Whiteson, 2018; Degris et al., 2012; Gu et al., 2017b,c; Sutton, 2000) through an off-policy fitted critic $Q_w(s,a)$, which provides low-variance but biased gradients. IPG directly interpolates the two terms from Eq. 2.24 and 2.31:

$$\begin{aligned}\nabla_\theta J(\theta) \approx\ & (1-\nu)\mathbb{E}_{s_t\sim\rho^\pi(\cdot),a_t\sim\pi(\cdot|s_t),\tau_{t+1}\sim P,\pi}[\nabla_\theta \log\pi_\theta(a_t|s_t)\hat{Q}(s_t,a_t,\tau_{t+1})] \\ & + \nu\mathbb{E}_{s_t\sim\rho^\beta(\cdot)}[\nabla_\theta \mathbb{E}_{a\sim\pi(\cdot|s_t)}[Q_w(s_t,a)]] \\ =\ & (1-\nu)\mathbb{E}_\pi[\nabla_\theta \log\pi_\theta(a_t|s_t)\hat{Q}(s_t,a_t,\tau_{t+1})] + \nu\mathbb{E}_\beta[\nabla_\theta \bar{Q}^\pi_w(s_t)],\end{aligned} \tag{4.1}$$

where $0\le\nu\le 1$ and $\bar{Q}^\pi_w(s_t)=\mathbb{E}_{a\sim\pi(\cdot|s_t)}[Q_w(s_t,a)]$, and we use E_π and E_β to shorten the notations for on-policy and off-policy sampling of transitions for gradient computation. For the simplicity of presentation, we do not include state-dependent baselines for likelihood ratio terms, but they can be used without affecting the derivation. This gradient estimator is biased from two sources: off-policy state sampling ρ^β, and inaccuracies in the critic Q_w. However, as we show in Section 4.2, we can bound the biases for all the cases, and in some cases, the algorithm still guarantees monotonic convergence as in Kakade and Langford (2002); Schulman et al. (2015b).

4.1.1 Control Variates for Interpolated Policy Gradient

While IPG includes ν to trade off bias and variance directly, it contains a likelihood ratio gradient term, for which we can introduce a control variate (CV) (Ross, 2006) to further

reduce the estimator variance. The expression for the IPG with control variates is below,

$$\nabla_\theta J(\theta) \approx (1-\nu)\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{Q}(s_t,a_t,\tau_{t+1})] + \nu\mathbb{E}_\beta[\nabla_\theta \bar{Q}^\pi_w(s_t)] \tag{4.2}$$

$$= (1-\nu)\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a_t|s_t)(\hat{Q}(s_t,a_t,\tau_{t+1}) - Q_w(s_t,a_t))] \tag{4.3}$$

$$+ (1-\nu)\mathbb{E}_\pi[\nabla_\theta \bar{Q}^\pi_w(s_t)] + \nu\mathbb{E}_\beta[\nabla_\theta \bar{Q}^\pi_w(s_t)] \tag{4.4}$$

$$\approx (1-\nu)\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a_t|s_t)(\hat{Q}(s_t,a_t,\tau_{t+1}) - Q_w(s_t,a_t))] + \mathbb{E}_\beta[\nabla_\theta \bar{Q}^\pi_w(s_t)]. \tag{4.5}$$

The first approximation indicates the biased approximation from IPG, while the second approximation indicates replacing the ρ^π in the control variate correction term with ρ^β and merging with the last term. The second approximation is a design decision and introduces additional bias when $\beta \neq \pi$ but it helps simplify the expression to be analyzed more easily, and the additional benefit from the variance reduction from the control variate could still outweigh this extra bias. The biases are analyzed in Section 4.2. The likelihood ratio gradient term is now proportional to the residual in on- and off-policy Q-value or advantage estimates $\hat{Q}(s_t,a_t,\tau_{t+1}) - Q_w(s_t,a_t) = \hat{A}(s_t,a_t,\tau_{t+1}) - A_w(s_t,a_t)$, and therefore, we call this term *residual likelihood ratio gradient*. Intuitively, if the off-policy critic estimate is accurate, this term has a low magnitude and the overall variance of the estimator is reduced.

4.1.2 Relationship to Prior Policy Gradient and Actor-Critic Methods

Crucially, IPG allows interpolating a rich list of prior deep policy gradient methods using only three parameters: β, ν, and the use of the control variate (CV). The connection is summarized in Table 4.1 and the algorithm is presented in Algorithm 3. Importantly, a wide range of prior work has only explored limiting cases of the spectrum, e.g. $\nu = 0, 1$, with or without the control variate. Our work provides a thorough theoretical analysis of the biases, and in some cases performance guarantees, for each of the method in this spectrum and empirically demonstrates often the best performing algorithms are in the midst of the spectrum.

4.1.3 $\nu = 1$: Actor-Critic methods

Before presenting our theoretical analysis, an important special case to discuss is $\nu = 1$, which corresponds to an expected actor-critic method. Several advantages of this special case include that the policy can be deterministic and the learning can be done effectively off-policy, as it does not have to estimate the Monte Carlo critic $\hat{Q}$ through rolling out on-policy trajectories or doing high-variance importance correction on off-policy trajectories (Dudík et al., 2011; Jiang and Li, 2016; Munos et al., 2016). Prior work such as DDPG (Lillicrap

Algorithm 3 Interpolated Policy Gradient

β, ν, useCV Initialize w for critic Q_w, θ for stochastic policy π_θ, and replay buffer $\mathcal{R} \leftarrow \emptyset$. **repeat**
Roll-out π_θ for E episodes, T time steps each, to collect a batch of data $\mathcal{B} = \{s,a,r\}_{1:T,1:E}$ to $\mathcal{R}$
Fit Q_w using $\mathcal{R}$ and π_θ, and fit baseline $V_\phi(s_t)$ using $\mathcal{B}$
Compute Monte Carlo advantage estimate $\hat{A}_{t,e}$ using $\mathcal{B}$ and V_ϕ
if useCV **then**
  Compute critic-based advantage estimate $\bar{A}_{t,e}$ using $\mathcal{B}$, Q_w and π_θ
  Compute and center the learning signals $l_{t,e} = \hat{A}_{t,e} - \bar{A}_{t,e}$ and set $b = 1$
else
  Center the learning signals $l_{t,e} = \hat{A}_{t,e}$ and set $b = \nu$
end if
Multiply $l_{t,e}$ by $(1-\nu)$
Sample $\mathcal{D} = s_{1:M}$ from $\mathcal{R}$ and/or $\mathcal{B}$ based on β
Compute $\nabla_\theta J(\theta) \approx \frac{1}{ET}\sum_e \sum_t \nabla_\theta \log \pi_\theta(a_{t,e}|s_{t,e}) l_{t,e} + \frac{b}{M}\sum_m \nabla_\theta \bar{Q}^\pi_w(s_m)$
Update policy π_θ using $\nabla_\theta J(\theta)$
until π_θ converges.

et al., 2016) and related Q-learning methods have proposed aggressive off-policy exploration strategy to exploit these properties of the algorithm. In this work, we compare alternatives such as using on-policy exploration and stochastic policy with classical DDPG algorithm designs, and show that in some domains the off-policy exploration can significantly deteriorate the performance. Theoretically, we confirm this empirical observation by showing that the bias from off-policy sampling in β increases monotonically with the total variation or KL divergence between β and π. Both the empirical and theoretical results indicate that well-designed actor-critic methods with an on-policy exploration strategy could be a more reliable alternative than with an off-policy exploration.

4.2 Theoretical Analysis

In this section, we present a theoretical analysis of the bias in the interpolated policy gradient. This is crucial, since understanding the biases of the methods can improve our intuition about its performance and make it easier to design new algorithms in the future. Because IPG includes many prior methods as special cases, our analysis also applies to those methods and other intermediate cases. We first analyze a special case and derive results for general IPG. All proofs are in the Appendix.

4.2.1 $\beta \neq \pi$, $\nu = 0$: Policy Gradient with Control Variate and Off-Policy Sampling

This section provides an analysis of the special case of IPG with $\beta \neq \pi$, $\nu = 0$, and the control variate. Plugging in to Eq. 4.5, we get an expression similar to Q-Prop (Gu et al., 2017b),

$$\nabla_\theta J(\theta) \approx \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a_t|s_t)(\hat{Q}(s_t,a_t,\tau_{t+1}) - Q_w(s_t,a_t))] + \mathbb{E}_\beta[\nabla_\theta \bar{Q}_w^\pi(s_t)] \tag{4.6}$$

$$= \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a_t|s_t)(\hat{A}(s_t,a_t,\tau_{t+1}) - A_w^\pi(s_t,a_t))] + \mathbb{E}_\beta[\nabla_\theta \bar{Q}_w^\pi(s_t)], \tag{4.7}$$

except that it also supports utilizing off-policy data for updating the policy. We use $A_w^\pi(s,a) = Q_w(s,a) - \mathbb{E}_{a\sim\pi(\cdot|s_t)}[Q_w(s_t,a)]$ to denote the advantage estimate from a learned Q-function $Q_w \approx Q^\pi$, and $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ to later denote the advantage function of policy π. To analyze the bias for this gradient expression, we first introduce $\tilde{J}(\pi,\tilde{\pi})$, a local approximation to $J(\pi)$, which has been used in prior theoretical work (Kakade and Langford, 2002; Schulman et al., 2015b). The derivation and the bias from this approximation are discussed in the proof for Theorem 1 in the Appendix.

$$J(\pi) = J(\tilde{\pi}) + \mathbb{E}_{\rho^\pi,\pi}[A^{\tilde{\pi}}(s_t,a_t)] \approx J(\tilde{\pi}) + \mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)] = \tilde{J}(\pi,\tilde{\pi}). \tag{4.8}$$

Note that $J(\pi) = \tilde{J}(\pi,\tilde{\pi}=\pi)$ and $\nabla_\pi J(\pi) = \nabla_\pi \tilde{J}(\pi,\tilde{\pi}=\pi)$. In practice, $\tilde{\pi}$ corresponds to policy π_k at iteration k and π corresponds next policy π_{k+1} after parameter update. Thus, this approximation is often sufficiently good. Next, we write the approximate objective for Eq. 4.6,

$$\begin{aligned} \tilde{J}^{\beta,\nu=0,CV}(\pi,\tilde{\pi}) &\triangleq J(\tilde{\pi}) + \mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t) - A_w^{\tilde{\pi}}(s_t,a_t)] + \mathbb{E}_{\rho^\beta}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)] \approx \tilde{J}(\pi,\tilde{\pi}) \\ \bar{A}_w^{\pi,\tilde{\pi}}(s_t) &= \mathbb{E}_\pi[A_w^{\tilde{\pi}}(s_t,\cdot)] = \mathbb{E}_\pi[Q_w(s_t,\cdot)] - \mathbb{E}_{\tilde{\pi}}[Q_w(s_t,\cdot)]. \end{aligned} \tag{4.9}$$

Note that $\tilde{J}^{\beta,\nu=0}(\pi,\tilde{\pi}=\pi) = \tilde{J}(\pi,\tilde{\pi}=\pi) = J(\pi)$, and $\nabla_\pi \tilde{J}^{\beta,\nu=0}(\pi,\tilde{\pi}=\pi)$ equals Eq. 4.6. We can bound the absolute error between $\tilde{J}^{\beta,\nu=0,CV}(\pi,\tilde{\pi})$ and $J(\pi)$ by the following theorem, where $D_{\mathrm{KL}}^{\max}(\pi_i,\pi_j) = \max_s D_{\mathrm{KL}}(\pi_i(\cdot|s),\pi_j(\cdot|s))$ is the maximum KL divergence between π_i,π_j.

Theorem 1 *If* $\varepsilon = \max_s |\bar{A}_w^{\pi,\tilde{\pi}}(s)|$, $\zeta = \max_s |\bar{A}^{\pi,\tilde{\pi}}(s)|$, *where*

$$\bar{A}_w^{\pi,\tilde{\pi}}(s_t) = \mathbb{E}_\pi[A_w^{\tilde{\pi}}(s_t,\cdot)] = \mathbb{E}_\pi[Q_w(s_t,\cdot)] - \mathbb{E}_{\tilde{\pi}}[Q_w(s_t,\cdot)]$$

$$\bar{A}^{\pi,\tilde{\pi}}(s_t) = \mathbb{E}_\pi[A^{\tilde{\pi}}(s_t,\cdot)] = \mathbb{E}_\pi[Q^{\tilde{\pi}}(s_t,\cdot)] - \mathbb{E}_{\tilde{\pi}}[Q^{\tilde{\pi}}(s_t,\cdot)]$$

then

$$\left\| J(\pi) - \tilde{J}^{\beta,\nu=0,CV}(\pi,\tilde{\pi}) \right\|_1 \leq 2\frac{\gamma}{(1-\gamma)^2}\left(\varepsilon\sqrt{D_{KL}^{\max}(\tilde{\pi},\beta)} + \zeta\sqrt{D_{KL}^{\max}(\pi,\tilde{\pi})}\right)$$

Theorem 1 contains two terms: the second term confirms $\tilde{J}^{\beta,\nu=0,CV}$ is a local approximation around π and deviates from $J(\pi)$ as $\tilde{\pi}$ deviates, and the first term bounds the bias from off-policy sampling using the KL divergence between the policies $\tilde{\pi}$ and β. This means that the algorithm fits well with policy gradient methods which constrain the KL divergence per policy update, such as covariant policy gradient (Bagnell and Schneider, 2003), natural policy gradient (Kakade and Langford, 2002), REPS (Peters et al., 2010), and trust-region policy optimization (TRPO) (Schulman et al., 2015b).

4.2.2 Monotonic Policy Improvement Guarantee

Some forms of on-policy policy gradient methods have theoretical guarantees on monotonic convergence Kakade and Langford (2002); Schulman et al. (2015b). Such guarantees often correspond to stable empirical performance on challenging problems, even when some of the constraints are relaxed in practice (Duan et al., 2016; Gu et al., 2017b; Schulman et al., 2015b). We can show that a variant of IPG allows off-policy sampling while still guaranteeing monotonic convergence. The algorithm and the proof are provided in the appendix. This algorithm is usually impractical to implement; however, IPG with trust-region updates when $\beta \neq \pi, \nu = 1, \text{CV} = true$ approximates this monotonic algorithm, similar to how TRPO is an approximation to the theoretically monotonic algorithm proposed by Schulman et al. (2015b).

4.2.3 General Bounds on the Interpolated Policy Gradient

We can establish bias bounds for the general IPG algorithm, with and without the control variate, using Theorem 2. The additional term that contributes to the bias in the general case is δ, which represents the error between the advantage estimated by the off-policy critic and the true A^π values.

Theorem 2 *If* $\delta = \max_{s,a} |A^{\tilde{\pi}}(s,a) - A^{\tilde{\pi}}_w(s,a)|$, $\varepsilon = \max_s |\bar{A}^{\pi,\tilde{\pi}}_w(s)|$, $\zeta = \max_s |\bar{A}^{\pi,\tilde{\pi}}(s)|$,

$$\tilde{J}^{\beta,\nu}(\pi,\tilde{\pi}) \triangleq J(\tilde{\pi}) + (1-\nu)\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[\hat{A}^{\tilde{\pi}}] + \nu\mathbb{E}_{\rho^{\beta}}[\bar{A}^{\pi,\tilde{\pi}}_w]$$

$$\tilde{J}^{\beta,\nu,CV}(\pi,\tilde{\pi}) \triangleq J(\tilde{\pi}) + (1-\nu)\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[\hat{A}^{\tilde{\pi}} - A^{\tilde{\pi}}_w] + \mathbb{E}_{\rho^{\beta}}[\bar{A}^{\pi,\tilde{\pi}}_w]$$

$$\textit{then}, \left\| J(\pi) - \tilde{J}^{\beta,\nu}(\pi,\tilde{\pi}) \right\|_1 \leq \frac{\nu\delta}{1-\gamma} + 2\frac{\gamma}{(1-\gamma)^2}\left(\nu\varepsilon\sqrt{D^{\max}_{KL}(\tilde{\pi},\beta)} + \zeta\sqrt{D^{\max}_{KL}(\pi,\tilde{\pi})}\right)$$

$$\left\| J(\pi) - \tilde{J}^{\beta,\nu,CV}(\pi,\tilde{\pi}) \right\|_1 \leq \frac{\nu\delta}{1-\gamma} + 2\frac{\gamma}{(1-\gamma)^2}\left(\varepsilon\sqrt{D^{\max}_{KL}(\tilde{\pi},\beta)} + \zeta\sqrt{D^{\max}_{KL}(\pi,\tilde{\pi})}\right)$$

This bound shows that the bias from directly mixing the expected policy gradient through ν comes from two terms: how well the critic Q_w is approximating Q^{π}, and how close the off-policy sampling policy is to the actor policy. We also show that the bias introduced is proportional to ν while the variance of the high variance likelihood ratio gradient term is proportional to $(1-\nu)^2$, so ν allows directly trading off bias and variance. Theorem 2 fully bounds bias in the full spectrum of IPG methods; this enables us to analyze how biases arise and interact and help us design better algorithms.

4.3 Related Work

An overarching aim of this chapter is to help unify on-policy and off-policy policy gradient algorithms into a single conceptual framework. Q-Prop (Gu et al., 2017b), PGQ (O'Donoghue et al., 2017), and ACER (Wang et al., 2017) are all recent works that combine on-policy with off-policy learning. IPG with $0 < \nu < 1$ and without the control variate relates closely to PGQ and ACER, but differ in the details. PGQ mixes in the Q-learning Bellman error objective, and ACER mixes parameter update steps rather than directly mixing gradients. And both PGQ and ACER come with numerous additional design details that make fair comparisons with methods like TRPO and Q-Prop difficult. We instead focus on the three minimal variables of IPG and explore their settings in relation to the closely related TRPO and Q-Prop methods, in order to theoretically and empirically understand in which situations we might expect gains from mixing on- and off-policy gradients.

Asides from these more recent works, the use of off-policy samples with policy gradients has been a popular direction of research (Degris et al., 2012; Jie and Abbeel, 2010; Levine and Koltun, 2013; Peshkin and Shelton, 2002). Most of these methods rely on variants of importance sampling (IS) to correct for bias. The use of importance sampling ensures unbiased estimates, but at the cost of considerable variance, as quantified by the ESS measure used by Jie and Abbeel (2010). Ignoring importance weights produces bias but, as shown in

our analysis and prior work (Wawrzyński, 2009), this bias can be bounded. Therefore, our IPG estimators have higher bias as the sampling distribution deviates from the policy, while IS methods have higher variance. Among these importance sampling methods, Levine and Koltun (2013) evaluates on tasks that are the most similar to our work, but the focus is on using importance sampling to include demonstrations, rather than to speed up learning from scratch.

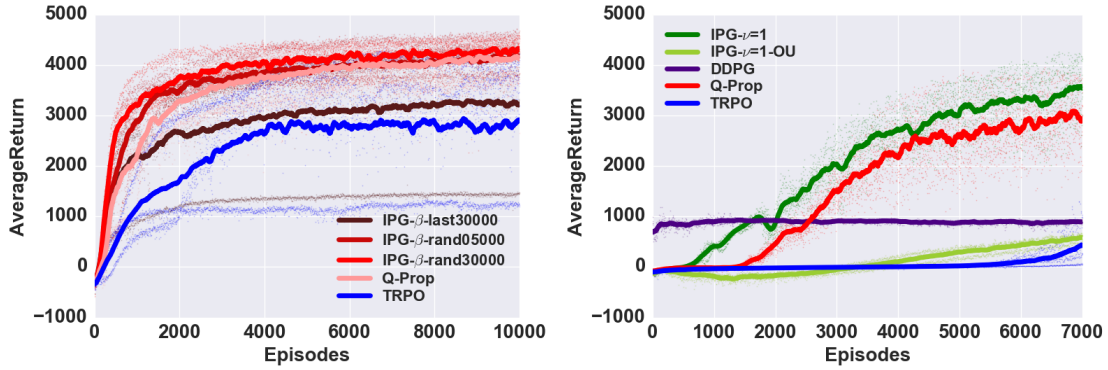
Lastly, there are many methods that combine on- and off-policy data for policy evaluation (Mahmood et al., 2014; Munos et al., 2016; Precup, 2000), mostly through variants of importance sampling. Combining our methods with more sophisticated policy evaluation methods will likely lead to further improvements, as done in (Degris et al., 2012). A more detailed analysis of the effect of importance sampling on bias and variance is left to future work, where some of the relevant work includes Jiang and Li (2016); Jie and Abbeel (2010); Mahmood et al. (2014); Precup (2000); Thomas and Brunskill (2016).

4.4 Experiments

In this section, we empirically show that the three parameters of IPG can interpolate different behaviors and often achieve superior performance versus prior methods that are limiting cases of this approach. Crucially, all methods share the same algorithmic structure as Algorithm 3, and we hold the rest of the experimental details fixed. All experiments were performed on MuJoCo domains in OpenAI Gym (Brockman et al., 2016; Todorov et al., 2012), with results presented for the average over three seeds. Additional experimental details are provided in the Appendix.

4.4.1 $\beta \neq \pi$, $\nu = 0$, with the control variate

We evaluate the performance of the special case of IPG discussed in Section 4.2.1. This case is of particular interest, since we can derive monotonic convergence results for a variant of this method under certain conditions, despite the presence of off-policy updates. Figure 4.1a shows the performance on the HalfCheetah-v1 domain, when the policy update batch size is 5000 transitions (i.e. 5 episodes). "last" and "rand" indicate if β samples from the most recent transitions or uniformly from the experience replay. "last05000" would be equivalent to Q-Prop given $\nu = 0$. Comparing "IPG-β-rand05000" and "Q-Prop" curves, we observe that by drawing the same number of samples randomly from the replay buffer for estimating the critic gradient, instead of using the on-policy samples, we get faster convergence. If we sample batches of size 30000 from the replay buffer, the performance further improves.



(a) IPG with $\nu = 0$ and the control variate.

(b) IPG with $\nu = 1$.

Fig. 4.1 (a) IPG-$\nu = 0$ vs Q-Prop on HalfCheetah-v1, with batch size 5000. IPG-β-rand30000, which uses 30000 random samples from the replay as samples from β, outperforms Q-Prop in terms of learning speed. (b) IPG-ν=1 vs other algorithms on Ant-v1. In this domain, on-policy IPG-ν=1 with on-policy exploration significantly outperforms DDPG and IPG-ν=1-OU, which use a heuristic OU (Ornstein–Uhlenbeck) process noise exploration strategy, and marginally outperforms Q-Prop.

However, as seen in the "IPG-β-last30000" curve, if we instead use the 30000 most recent samples, the performance degrades. One possible explanation for this is that, while using random samples from the replay increases the bound on the bias according to Theorem 1, it also decorrelates the samples within the batch, providing more stable gradients. This is the original motivation for experience replay in the DQN method (Mnih et al., 2015), and we have shown that such decorrelated off-policy samples can similarly produce gains for policy gradient algorithms. See Table 4.2 for results on other domains.

The results for this variant of IPG demonstrate that random sampling from the replay provides further improvement on top of Q-Prop. Note that these replay buffer samples are different from standard off-policy samples in DDPG or DQN algorithms, which often use aggressive heuristic exploration strategies. The samples used by IPG are sampled from prior policies that follow a conservative trust-region update, resulting in greater regularity but less exploration. In the next section, we show that in some cases, ensuring that the off-policy samples are not *too* off-policy is essential for good performance.

4.4.2 $\beta = \pi, \nu = 1$

In this section, we empirically evaluate another special case of IPG, where $\beta = \pi$, indicating on-policy sampling, and $\nu = 1$, which reduces to a trust-region, on-policy variant of a

expected actor-critic method. Although this algorithm performs actor-critic updates, the use of a trust region makes it more similar to TRPO or Q-Prop than DDPG.

Results for all domains are shown in Table 4.2. Figure 4.1b shows the learning curves on Ant-v1. Although IPG-ν=1 methods can be off-policy, the policy is updated every 5000 samples to keep it consistent with other IPG methods, while DDPG updates the policy on every step in the environment and makes other design choices (Lillicrap et al., 2016). We see that, in this domain, standard DDPG becomes stuck with a mean reward of 1000, while IPG-ν=1 improves monotonically, achieving a significantly better result. To investigate why this large discrepancy arises, we also ran IPG-ν=1 with the same OU process exploration noise as DDPG, and observed large degradation in performance. This provides empirical support for Theorem 2. It is illuminating to contrast this result with the previous experiment, where the off-policy samples did not adversely alter the results. In the previous experiments, the samples came from Gaussian policies updated with trust-regions. The difference between π and β was therefore approximately bounded by the trust-regions. In the experiment with Brownian noise, the behaving policy uses temporally correlated noise, with potentially unbounded KL-divergence from the learned Gaussian policy. In this case, the off-policy samples result in excessive bias, wiping out the variance reduction benefits of off-policy sampling. In general, we observed that for the harder Ant-v1 and Walker-v1 domains, on-policy exploration is more effective, even when doing off-policy state sampling from a replay buffer. This results suggests the following lesson for designing off-policy actor-critic methods: for domains where exploration is difficult, it may be more effective to use on-policy exploration with bounded policy updates than to design heuristic exploration rules such as the OU process noise, due to the resulting reduction in bias.

4.4.3 General Cases of Interpolated Policy Gradient

Table 4.2 shows the results for experiments where we compare IPG methods with varying values of ν; additional results are provided in the Appendix. $\beta \neq \pi$ indicates that the method uses off-policy samples from the replay buffer, with the same batch size as the on-policy batch for fair comparison. We ran sweeps over $\nu = \{0.2, 0.4, 0.6, 0.8\}$ and found that $\nu = 0.2$ consistently produce better performance than Q-Prop, TRPO or prior actor-critic methods. As an example, Figure 4.2 demonstrates that $\nu = 0.2$ outperforms the Q-Prop baseline in the challenging Humanoid-v1 environment. Importantly, we compared all methods with the same algorithm designs (exploration, policy, etc.), since Q-Prop and TRPO are IPG-ν=0 with and without the control variate. IPG-ν=1 is a novel variant of the actor-critic method that differs from DDPG (Lillicrap et al., 2016) and SVG(0) (Heess et al., 2015a) due to the use of a trust region. The results in Table 4.2 suggest that, in most cases, the best performing

	HalfCheetah-v1		Ant-v1		Walker-v1		Humanoid-v1	
	$\beta = \pi$	$\beta \neq \pi$	$\beta = \pi$	$\beta \neq \pi$	$\beta = \pi$	$\beta \neq \pi$	$\beta = \pi$	$\beta \neq \pi$
IPG-ν=0.2	3356	3458	**4237**	**4415**	**3047**	1932	1231	920
IPG-cv-ν=0.2	**4216**	4023	**3943**	**3421**	1896	1411	**1651**	**1613**
IPG-ν=1	2962	**4767**	**3469**	**3780**	2704	805	**1571**	**1530**
Q-Prop	4178	**4182**	3374	**3479**	2832	1692	1423	**1519**
TRPO	2889	N.A.	1520	N.A.	1487	N.A.	615	N.A.

Table 4.2 Comparisons on all domains with mini-batch size 10000 for Humanoid and 5000 otherwise. We compare the maximum of average test rewards in the first 10000 episodes (Humanoid requires more steps to fully converge; see the Appendix for learning curves). Results outperforming Q-Prop (or IPG-cv-ν=0 with $\beta = \pi$) are boldface. The two columns show results with on-policy and off-policy samples for estimating the expected policy gradient.

algorithm is one that interpolates between the policy-gradient and actor-critic variants, with intermediate values of ν.

4.5 Discussion

In this section, we introduced interpolated policy gradient methods, a family of policy gradient algorithms that allow mixing off-policy learning with on-policy learning while satisfying performance bounds. This family of algorithms unifies and interpolates on-policy likelihood ratio policy gradient and off-policy expected policy gradient, and includes a number of prior works as approximate limiting cases. Empirical results confirm that, in many cases, interpolated gradients have improved sample-efficiency and stability over the prior state-of-the-art methods, and the theoretical results provide intuition for analyzing the cases in which the different methods perform well or poorly. Our hope is that this detailed analysis of interpolated gradient methods can not only provide for more effective algorithms in practice, but also give useful insight for future algorithm design.

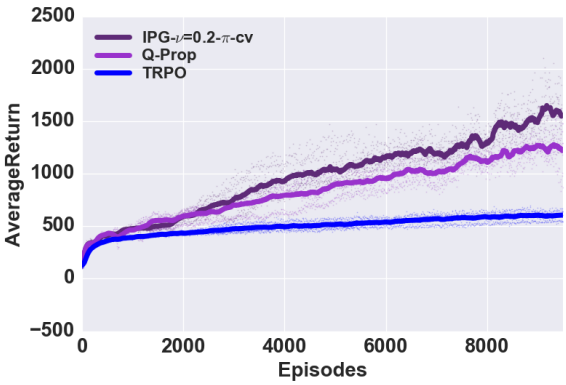


Fig. 4.2 IPG-$\nu = 0.2$-π-CV vs Q-Prop and TRPO on Humanoid-v1 with batch size 10000 in the first 10000 episodes. IPG-$\nu = 0.2$-π-CV, with a small difference of $\nu = 0.2$ multiplier, out-performs Q-Prop. All these methods have stable, monotonic policy improvement. The experiment is cut at 10000 episodes due to heavy compute requirement of Q-Prop and IPG methods, mostly from fitting the off-policy critic.

Chapter 5

Temporal Difference Models: Model-Free Deep RL for Model-Based Control

Reinforcement learning (RL) algorithms provide a formalism for autonomous learning of complex behaviors. When combined with rich function approximators such as deep neural networks, RL can provide impressive results on tasks ranging from playing games (Mnih et al., 2015; Silver et al., 2016), to flying and driving (Lillicrap et al., 2016; Zhang et al., 2016), to controlling robotic arms (Gu et al., 2017a; Levine et al., 2016). However, these deep RL algorithms often require a large amount of experience to arrive at an effective solution, which can severely limit their application to real-world problems where this experience might need to be gathered directly on a real physical system. Part of the reason for this is that direct, model-free RL learns only from the reward: experience that receives no reward provides minimal supervision to the learner.

In contrast, model-based RL algorithms obtain a large amount of supervision from every sample, since they can use each sample to better learn how to predict the system dynamics – that is, to learn the "physics" of the problem. Once the dynamics are learned, near-optimal behavior can in principle be obtained by planning through these dynamics. Model-based algorithms tend to be substantially more efficient (Deisenroth et al., 2013; Nagabandi et al., 2017), but often at the cost of larger asymptotic bias: when the dynamics cannot be learned perfectly, as is the case for most complex problems, the final policy can be highly suboptimal. Therefore, conventional wisdom holds that model-free methods are less efficient but achieve the best asymptotic performance, while model-based methods are more efficient but do not produce policies that are as optimal.

Can we devise methods that retain the efficiency of model-based learning while still achieving the asymptotic performance of model-free learning? This is the question that we study in this chapter. The search for methods that combine the best of model-based and model-free learning has been ongoing for decades, with techniques such as synthetic experience generation (Sutton, 1990), partial model-based backpropagation (Heess et al., 2015b; Nguyen and Widrow, 1990), and layering model-free learning on the residuals of model-based estimation (Chebotar et al., 2017a) being a few examples. However, a direct connection between model-free and model-based RL has remained elusive. By effectively bridging the gap between model-free and model-based RL, we should be able to smoothly transition from learning models to learning policies, obtaining rich supervision from every sample to quickly gain a moderate level of proficiency, while still converging to an unbiased solution.

To arrive at a method that combines the strengths of model-free and model-based RL, we study a variant of goal-conditioned value functions (Andrychowicz et al., 2017; Schaul et al., 2015a; Sutton et al., 2011). Goal-conditioned value functions learn to predict the value function for every possible goal state. That is, they answer the following question: what is the expected reward for reaching a particular state, given that the agent is attempting (as optimally as possible) to reach it? The particular choice of reward function determines what such a method actually does, but rewards based on distances to a goal hint at a connection to model-based learning: if we can predict how easy it is to reach any state from any current state, we must have some kind of understanding of the underlying "physics." In this work, we show that we can develop a method for learning variable-horizon goal-conditioned value functions where, for a specific choice of reward and horizon, the value function corresponds directly to a model, while for larger horizons, it more closely resembles model-free approaches. Extension toward more model-free learning is thus achieved by acquiring "multi-step models" that can be used to plan over progressively coarser temporal resolutions, eventually arriving at a fully model-free formulation.

The principle contribution of our work is a new RL algorithm that makes use of this connection between model-based and model-free learning to learn a specific type of goal-conditioned value function, which we call a temporal difference model (TDM). This value function can be learned very efficiently, with sample complexities that are competitive with model-based RL, and can then be used with an MPC-like method to accomplish desired tasks. Our empirical experiments demonstrate that this method achieves substantially better sample complexity than fully model-free learning on a range of challenging continuous control tasks, while outperforming purely model-based methods in terms of final performance. Furthermore,

the connection that our method elucidates between model-based and model-free learning may lead to a range of interesting future methods.

5.1 Preliminaries

In this section, we introduce the reinforcement learning (RL) formalism, temporal difference Q-learning methods, model-based RL methods, and goal-conditioned value functions. We will build on these components to develop temporal difference models (TDMs) in the next section. RL deals with decision making problems that consist of a state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $P(s' \mid s,a)$, and an initial state distribution p_0. The goal of the learner is encapsulated by a reward function $r(s,a,s')$. Typically, long or infinite horizon tasks also employ a discount factor γ, and the standard objective is to find a policy $\pi(a \mid s)$ that maximizes the expected discounted sum of rewards, $\mathbb{E}_\pi[\sum_t \gamma^t r(s_t,a_t,s_{t+1})]$, where $s_0 \sim p_0$, $a_t \sim \pi(a_t|s_t)$, and $s_{t+1} \sim P(s' \mid s,a)$.

Q-functions. We will focus on RL algorithms that learn a Q-function. The Q-function represents the expected total (discounted) reward that can be obtained by the optimal policy after taking action a_t in state s_t, and can be defined recursively as following:

$$Q(s_t,a_t) = \mathbb{E}_{p(s_{t+1}|s_t,a_t)}[r(s_t,a_t,s_{t+1}) + \gamma \max_a Q(s_{t+1},a)]. \tag{5.1}$$

The optimal policy can then recovered according to $\pi(a_t|s_t) = \delta(a_t = \arg\max_a Q(s_t,a))$. Deep Q-learning algorithms (Mnih et al., 2015; Riedmiller, 2005; Watkins and Dayan, 1992) learn the Q-function via an off-policy stochastic gradient descent algorithm, estimating the expectation in the above equation with samples collected from the environment and computing its gradient. Q-learning methods can use transition tuples (s_t,a_t,s_{t+1},r_t) collected from any exploration policy, which generally makes them more efficient than direct policy search, though still less efficient than purely model-based methods.

Model-based RL and optimal control. Model-based RL takes a different approach to maximize the expected reward. In model-based RL, the aim is to train a model of the form $f(s_t,a_t)$ to predict the next state s_{t+1}. Once trained, this model can be used to choose actions, either by backpropagating reward gradients into a policy, or planning directly through the model. In the latter case, a particularly effective method for employing a learned model is model-predictive control (MPC), where a new action plan is generated at each time step, and the first action of that plan is executed, before replanning begins from scratch. MPC can be

formalized as the following optimization problem:

$$a_t = \arg\max_{a_{t:t+T}} \sum_{i=t}^{t+T} r(s_i, a_i) \text{ where } s_{i+1} = f(s_i, a_i) \; \forall \; i \in \{t, ..., t+T-1\}. \tag{5.2}$$

We can also write the dynamics constraint in the above equation in terms of an implicit dynamics, according to

$$a_t = \arg\max_{a_{t:t+T}, s_{t+1:t+T}} \sum_{i=t}^{t+T} r(s_i, a_i) \text{ such that } C(s_i, a_i, s_{i+1}) = 0 \; \forall \; i \in \{t, ..., t+T-1\}, \tag{5.3}$$

where $C(s_i, a_i, s_{i+1}) = 0$ if and only if $s_{i+1} = f(s_i, a_i)$. This implicit version will be important in understanding the connection between model-based and model-free RL.

Goal-conditioned value functions. Q-functions trained for a specific reward are specific to the corresponding task, and learning a new task requires optimizing an entirely new Q-function. Goal-conditioned value functions address this limitation by conditioning the Q-value on some task description vector $s_g \in \mathcal{G}$ in a goal space $\mathcal{G}$. This goal vector induces a parameterized reward $r(s_t, a_t, s_{t+1}, s_g)$, which in turn gives rise to parameterized Q-functions of the form $Q(s, a, s_g)$. A number of goal-conditioned value function methods have been proposed in the literature, such as universal value functions (Schaul et al., 2015a) and Horde (Sutton et al., 2011). When the goal corresponds to an entire state, such goal-conditioned value functions usually predict how well an agent can reach a particular state, *when it is trying to reach it*. The knowledge contained in such a value function is intriguingly close to a model: knowing how well you can reach any state is closely related to understanding the physics of the environment. With Q-learning, these value functions can be learned for any goal s_g using the same off-policy (s_t, a_t, s_{t+1}) tuples. Relabeling previously visited states with the reward for any goal leads to a natural data augmentation strategy, since each tuple can be replicated many times for many different goals without additional data collection. Andrychowicz et al. (2017) used this property to produce an effective curriculum for solving multi-goal task with delayed rewards. As we discuss below, relabeling past experience with different goals enables goal-conditioned value functions to learn much more quickly from the same amount of data.

5.2 Temporal Difference Model Learning

In this section, we introduce a type of goal-conditioned value functions called temporal difference models (TDMs) that provide a direct connection to model-based RL. We will first motivate this connection by relating the model-based MPC optimizations in Equations (5.2) and (5.3) to goal-conditioned value functions, and then present our temporal difference model derivation, which extends this connection from a purely model-based setting into one that becomes increasingly model-free.

5.2.1 From Goal-Conditioned Value Functions to Models

Let us consider the choice of reward function for the goal conditioned value function. Although a variety of options have been explored in the literature (Andrychowicz et al., 2017; Schaul et al., 2015a; Sutton et al., 2011), a particularly intriguing connection to model-based RL emerges if we set $\mathcal{G} = \mathcal{S}$, such that $g \in \mathcal{G}$ corresponds to a *goal state* $s_g \in \mathcal{S}$, and we consider distance-based reward functions r_d of the following form:

$$r_d(s_t, a_t, s_{t+1}, s_g) = -D(s_{t+1}, s_g),$$

where $D(s_{t+1}, s_g)$ is a distance, such as the Euclidean distance $D(s_{t+1}, s_g) = \|s_{t+1} - s_g\|_2$. This choice of reward function seems arbitrary at first glance; however, it is a sensible function as it connects closely with the implicit dynamics function discussed in Eq. 5.3: reward is maximum at 0 only if the goal satisfies the dynamics of the MDP. Specifically, if $\gamma = 0$, we have $Q(s_t, a_t, s_g) = -D(s_{t+1}, s_g)$ at convergence of Q-learning, which means that $Q(s_t, a_t, s_g) = 0$ implies that $s_{t+1} = s_g$. Plug this Q-function into the model-based planning optimization in Equation (5.3), denoting the task control reward as r_c, such that the solution to

$$a_t = \underset{a_{t:t+T}, s_{t+1:t+T}}{\arg\max} \sum_{i=t}^{t+T} r_c(s_i, a_i) \text{ such that } Q(s_i, a_i, s_{i+1}) = 0 \; \forall \; i \in \{t, ..., t+T-1\} \tag{5.4}$$

yields a model-based plan. We have now derived a precise connection between model-free and model-based RL, in that model-free learning of goal-conditioned value functions can be used to directly produce an implicit model that can be used with MPC-based planning. However, this connection by itself is not very useful: the resulting implicit model is fully model-based, and does not provide any kind of long-horizon capability. In the next section, we show how to extend this connection into the long-horizon setting by introducing the temporal difference model (TDM).

5.2.2 Long-Horizon Learning with Temporal Difference Models

If we consider the case where $\gamma > 0$, the optimization in Equation (5.4) no longer corresponds to any optimal control method. In fact, when $\gamma = 0$, Q-values have well-defined units: units of distance between states. For $\gamma > 0$, no such interpretation is possible. The key insight in temporal difference models is to introduce a different mechanism for aggregating long-horizon rewards. Instead of evaluating Q-values as discounted sums of rewards, we introduce an additional input τ, which represents the planning horizon, and define the Q-learning recursion as

$$Q(s_t, a_t, s_g, \tau) = \mathbb{E}_{p(s_{t+1}|s_t,a_t)}[-D(s_{t+1}, s_g)1[\tau = 0] + \max_a Q(s_{t+1}, a, s_g, \tau - 1)1[\tau \neq 0]]. \tag{5.5}$$

The Q-function uses a reward of $-D(s_{t+1}, s_g)$ when $\tau = 0$ (at which point the episode terminates), and decrements τ by one at every other step. Since this is still a well-defined Q-learning recursion, it can be optimized with off-policy data and, just as with goal-conditioned value functions, we can resample new goals s_g and new horizons τ for each tuple (s_t, a_t, s_{t+1}), even ones that were not actually used when the data was collected. In this way, the TDM can be trained very efficiently, since every tuple provides supervision for every possible goal and every possible horizon.

The intuitive interpretation of the TDM is that it tells us how close the agent will get to a given goal state s_g after τ time steps, *when it is attempting to reach that state in τ steps*. Alternatively, TDMs can be interpreted as Q-values in a finite-horizon MDP, where the horizon is determined by τ. For the case where $\tau = 0$, TDMs effectively learn a model, allowing TDMs to be incorporated into a variety of planning and optimal control schemes at test time as in Equation (5.4). Thus, we can view TDM learning as an interpolation between model-based and model-free learning, where $\tau = 0$ corresponds to the single-step prediction made in model-based learning and $\tau > 0$ corresponds to the long-term prediction made by typical Q-functions. While the correspondence to models is not the same for $\tau > 0$, if we only care about the reward at every K step, then we can recover a correspondence by replacing Equation (5.4) with

$$\begin{aligned} a_t = & \operatorname*{arg\,max}_{a_{t:K:t+T}, s_{t+K:K:t+T} i = t, t+K, \ldots, t+T} \sum r_c(s_i, a_i) \\ & \text{such that } Q(s_i, a_i, s_{i+K}, K-1) = 0 \; \forall \, i \in \{t, t+K, \ldots, t+T-K\}, \end{aligned} \tag{5.6}$$

where we only optimize over every K^{th} state and action, i.e. $\{t : K : t+T\} = \{t, t+K, t+2K, \ldots, t+T\}$. As the TDM becomes effective for longer horizons, we can increase K until

$K = T$, and plan over only a single effective time step:

$$a_t = \underset{a_t, a_{t+T}, s_{t+T}}{\arg\max}\ r_c(s_{t+T}, a_{t+T}) \text{ such that } Q(s_t, a_t, s_{t+T}, T-1) = 0. \tag{5.7}$$

This formulation does result in some loss of generality, since we no longer optimize the reward at the intermediate steps. This limits the multi-step formulation to terminal reward problems, but does allow us to accommodate arbitrary reward functions on the terminal state s_{t+T}, which still describes a broad range of practically relevant tasks. Another limitation is that the current formulation assumes deterministic MDPs to define explicit and implicit dynamics models, and an extension to stochastic MDPs which then have to consider a probabilistic variant of goal satisfiability is an interesting future direction. In the next section, we describe how TDMs can be implemented and used in practice for continuous state and action spaces.

5.3 Training and Using Temporal Difference Models

The TDM can be trained with any off-policy Q-learning algorithm, such as DQN (Mnih et al., 2015), DDPG (Lillicrap et al., 2016), NAF (Gu et al., 2016b), and SDQN (Metz et al., 2017). During off-policy Q-learning, TDMs can benefit from arbitrary relabeling of the goal states g and the horizon τ, given the same (s_t, a_t, s_{t+1}) tuples from the behavioral policy as done in (Andrychowicz et al., 2017). This relabeling enables simultaneous, data-efficient learning of short-horizon and long-horizon behaviors for arbitrary goal states, unlike previously proposed goal-conditioned value functions that only learn for a single time scale, typically determined by a discount factor (Andrychowicz et al., 2017; Schaul et al., 2015a). In this section, we describe the design decisions needed to make practical TDM algorithm.

5.3.1 Reward Function Specification

Q-learning typically optimizes scalar rewards, but TDMs enable us to increase the amount of supervision available to the Q-function by using a vector-valued reward. Specifically, if the distance $D(s, s_g)$ factors additively over the dimensions, we can train a vector-valued Q-function that predicts per-dimension distance, with the reward function for dimension j given by $-D_j(s_j, s_{g,j})$. We use the ℓ_1 norm in our implementation, which corresponds to absolute value reward $-|s_j - s_{g,j}|$. The resulting vector-valued Q-function can learn distances along each dimension separately, providing it with more supervision from each training

point. Empirically, we found that this modifications provides a substantial boost in sample efficiency.

We can optionally make an improvement to TDMs if we know that the task reward r_c depends only on some subset of the state or, more generally, state features. In that case, we can train the TDM to predict distances along only those dimensions or features that are used by r_c, which in practice can substantially simplify the corresponding prediction problem. In our experiments, we illustrate this property by training TDMs for pushing tasks that predict distances from an end-effector and pushed object, without accounting for internal joints of the arm, and similarly for a locomotion task.

5.3.2 Policy Extraction with TDMs

While the TDM optimal control formulation Equation (5.7) drastically reduces the number of states and actions to be optimized for long-term planning, it requires solving a constrained optimization problem, which is more computationally expensive than unconstrained problems. We can remove the need for a constrained optimization through a specific architectural decision in the design of the function approximator for $Q(s,a,s_g,\tau)$. We define the Q-function as $Q(s,a,s_g,\tau) = -\|f(s,a,s_g,\tau) - g\|$, where $f(s,a,s_g,\tau)$ outputs a state vector. By training the TDM with a standard Q-learning method, $f(s,a,s_g,\tau)$ is trained to explicitly predict the state that will be reached by a policy attempting to reach s_g in τ steps. This model can then be used to choose the action with fully explicit MPC as below, which also allows straightforward derivation of a multi-step version as in Equation (5.6).

$$a_t = \underset{a_t, a_{t+T}, s_{t+T}}{\arg\max}\ r_c(f(s_t, a_t, s_{t+T}, T-1), a_{t+T}) \tag{5.8}$$

In the case where the task is to reach a goal state s_g, a simpler approach to extract a policy is to use the TDM directly:

$$a_t = \underset{a}{\arg\max}\, Q(s_t, a, s_g, \tau) \tag{5.9}$$

In our experiments, we use Equations (5.8) and (5.9) to extract a policy.

5.3.3 Algorithm Summary

The algorithm is summarized as Algorithm 4. A crucial difference from prior goal-conditioned value function methods (Andrychowicz et al., 2017; Schaul et al., 2015a) is that our algorithm can be used to act according to an arbitrary terminal reward function r_c, both during exploration and at test time. Like other off-policy algorithms (Lillicrap

Algorithm 4 Temporal Difference Model Learning

Require: Task reward function $r_c(s,a)$, parameterized TDM $Q_w(s,a,s_g,\tau)$, replay buffer $\mathcal{B}$
for $n = 0,...,N-1$ episodes **do**
  $s_0 \sim p(s_0)$
  for $t = 0,...,T-1$ time steps **do**
    $a_t^* = \text{MPC}(r_c, s_t, Q_w, T-t)$ {Eq. 5.6, Eq. 5.7, Eq. 5.8, or Eq. 5.9}
    $a_t = \text{AddNoise}(a_t^*)$ {Noisy exploration}
    $s_{t+1} \sim p(s_t, a_t)$, and store $\{s_t, a_t, s_{t+1}\}$ in the replay buffer $\mathcal{B}$ {Step environment}
    for $i = 0, I-1$ iterations **do**
      Sample M transitions $\{s_m, a_m, s'_m\}$ from the replay $\mathcal{B}$.
      Relabel time horizons and goal states $\tau_m, s_{g,m}$ {Section B.1.1}
      $y_m = -\|s'_m - s_{g,m}\| 1[\tau_m = 0] + \max_a Q'(s'_m, a, s_{g,m}, \tau_m - 1) 1[\tau_m \neq 0]$
      $L(w) = \sum_m (Q_w(s_m, a_m, s_{g,m}, \tau_m) - y_m)^2 / M$ {Compute the loss}
      Minimize$(w, L(w))$ {Optimize}
    end for
  end for
end for

et al., 2016; Mnih et al., 2015), it consists of exploration and Q-function fitting. Noise is injected for exploration, and Q-function fitting uses standard Q-learning techniques, with target networks Q' and experience replay (Lillicrap et al., 2016; Mnih et al., 2015). If we view the Q-function fitting as model fitting, the algorithm also resembles iterative model-based RL, which alternates between collecting data using the learned dynamics model for planning (Deisenroth and Rasmussen, 2011) and fitting the model. Since we focus on continuous tasks, we use DDPG (Lillicrap et al., 2016), though any Q-learning method could be used.

The computation cost of the algorithm is mostly determined by the number of updates to fit the Q-function per transition, I. In general, TDMs can benefit from substantially larger I than classic model-free methods such as DDPG due to relabeling increasing the amount of supervision signals. In real-world applications such as robotics where we care most of the sample efficiency (Gu et al., 2017a), the learning is often bottlenecked by the data collection rather than the computation, and therefore large I values are usually not a significant problem and can continuously benefit from the acceleration in computation.

5.4 Related Work

Combining model-based and model-free reinforcement learning techniques is a well-studied problem, though no single solution has demonstrated all of the benefits of model-based and model-free learning. Some methods first learn a model and use this model to simulate experience (Gu et al., 2016b; Sutton, 1990) or compute better gradients for model-free updates (Heess et al., 2015b; Nguyen and Widrow, 1990). Other methods use model-free

algorithms to correct for the local errors made by the model (Bansal et al., 2017; Chebotar et al., 2017a). While these prior methods focused on combining different model-based and model-free RL techniques, our method proposes an equivalence between these two branches of RL through a specific generalization of goal-conditioned value function. As a result, our approach achieves much better sample efficiency in practice on a variety of challenging reinforcement learning tasks than model-free alternatives, while exceeding the performance of purely model-based approaches.

We are not the first to study the connection between model-free and model-based methods, with Boyan (1999) and Parr et al. (2008) being two notable examples. Boyan (1999) shows that one can extract a model from a value function when using a tabular representation of the transition function. Parr et al. (2008) shows that, for linear function approximators, the model-free and model-based RL approaches produce the same value function at convergence. Our contribution differs substantially from these: we are not aiming to show that model-free RL with a reward function corresponding to a particular goal performs similarly to model-based RL at convergence, but rather how we can achieve sample complexity comparable to model-based RL while retaining the favorable asymptotic performance of model-free RL in complex tasks with function approximation.

A central component of our method is to train goal-conditioned value functions. Many variants of goal-conditioned value functions have been proposed in the literature Dosovitskiy and Koltun (2016); Foster and Dayan (2002); Schaul et al. (2015a); Sutton et al. (2011). Critically, unlike the works on contextual policies (Caruana, 1998; Da Silva et al., 2012; Kober and Peters, 2012) which require on-policy trajectories with each new goal, the value function approaches such as Horde (Sutton et al., 2011) and UVFA (Schaul et al., 2015a) can reuse off-policy data to learn rich contextual value functions using the same data.

TDMs condition on a policy trying to reach a goal and must predict τ steps into the future. This type of prediction is similar to the prediction made by prior work on multi-step models (Mishra et al., 2017; Venkatraman et al., 2016): predict the state after τ actions. An important difference is that multi-step models do not condition on a policy reaching a goal, and so they require optimizing over a sequence of actions, making the input space grow linearly with the planning horizon.

A particularly related UVFA extension is HER Andrychowicz et al. (2017); Kaelbling (1993). Both HER and our method retroactively relabel past experience with goal states that are different from the goal aimed for during data collection. However, unlike our method, the standard UVFA in HER uses a single temporal scale when learning, and does not explicitly provide for a connection between model-based and model-free learning. The practical result of these differences is that our approach empirically achieves better sample complexity on a

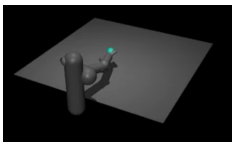
wide range of complex continuous control tasks than HER, while the theoretical connection between model-based and model-free learning sheds light on the underlying reasons for the efficiency of UVFA methods such as HER and our algorithm.

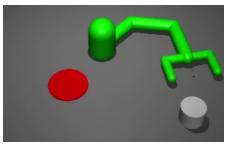
TDM can also be seen as an instance of hierarchical RL algorithm (Bacon et al., 2017; Dayan and Hinton, 1993; Dietterich, 2000; Kaelbling, 1993; Parr and Russell, 1998; Sutton et al., 1999b) for temporally-extended planning. In constrast with end-to-end model-free approaches explored in Bacon et al. (2017), TDM learns a goal-conditioned low-level policy using off-policy RL, and uses model-based planning for the high-level policy with respect to the task reward.

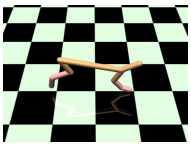
Lastly, our motivation is shared by other lines of work besides goal-conditioned value functions that try enhancing supervision signals for model-free RL (Bellemare et al., 2017; Jaderberg et al., 2017; Silver et al., 2017). Predictron (Silver et al., 2017) augments classic RL with multi-step reward predictions, while UNREAL (Jaderberg et al., 2017) also augments it with pixel control as a secondary reward objective. These are substantially different methods from our work, but share the motivation to achieve efficient RL by increasing the amount of learning signals from finite data.

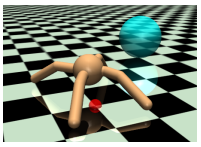
5.5 Experiments

Our experiments examine how the sample efficiency and performance of TDMs compare to both model-based and model-free RL algorithms. We expect to have the efficiency of model-based RL but with less model bias. We also aim to study the importance of several key design decisions in TDMs, and evaluate the algorithm on a real-world robotic platform. For the model-free comparison, we compare to DDPG (Lillicrap et al., 2016), which typically achieves the best sample efficiency on benchmark tasks (Duan et al., 2016); hindsight experience replay (HER), which uses goal-conditioned value functions (Andrychowicz et al., 2017); and DDPG with the same sparse rewards of HER. For the model-based comparison, we compare to the model-based component in Nagabandi et al. (2017), a recent work that reports highly efficient learning with neural network dynamics models. Details of the baseline implementations are in the Appendix. We perform the comparison on five simulated tasks: (1) a 7 DoF arm reaching various random end-effector targets, (2) an arm pushing a puck to a target location, (3) a planar cheetah attempting to reach a goal velocity (either forward or backward), (4) a quadrupedal ant attempting to reach a goal position, and (5) an ant attempting to reach a goal position and velocity. The tasks are shown in Figure 5.1 and terminate when either the goal is reached or the time horizon is reached. The pushing task requires long-horizon reasoning to reach and push the puck. The cheetah and ant tasks


(a) 7-DoF Reacher


(b) Pusher


(c) Half Cheetah


(d) Ant


(e) Sawyer Robot

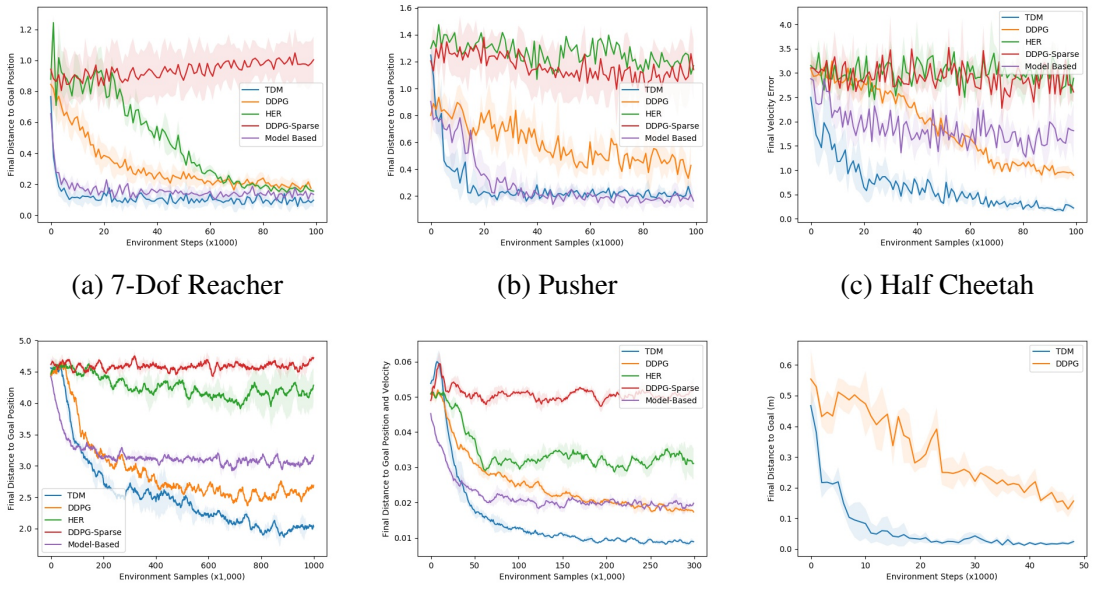
Fig. 5.1 The tasks in our experiments: (a) reaching target locations, (b) pushing a puck to a random target, (c) training the cheetah to run at target velocities, (d) training an ant to run to a target position or a target position and velocity, and (e) reaching target locations (real-world Sawyer robot).

require handling many contact discontinuities which is challenging for model-based methods, with the ant environment having particularly difficult dynamics given the larger state and action space. The ant position and velocity task presents a scenario where reward shaping as in traditional RL methods may not lead to optimal behavior, since one cannot maintain both a desired position and velocity. However, such a task can be very valuable in realistic settings. For example, if we want the ant to jump, we might instruct it to achieve a particular velocity at a particular location. We also tested TDMs on a real-world robot arm reaching end-effector positions, to study its applicability to real-world tasks.

For the simulated and real-world 7-DoF arm, our TDM is trained on all state components. For the pushing task, our TDM is trained on the hand and puck XY-position. For the half cheetah task, our TDM is trained on the velocity of the cheetah. For the ant tasks, our TDM is trained on either the position or the position and velocity for the respective task. Full details are in the Appendix.

5.5.1 TDMs vs Model-Free, Model-Based, and Direct Goal-Conditioned RL

The results are shown in Figure 5.2. When compared to the model-free baselines, the pure model-based method learns learns much faster on all the tasks. However, on the harder cheetah and ant tasks, its final performance is worse due to model bias. TDMs learn as quickly or faster than the model-based method, but also always learn policies that are as good as if not better than the model-free policies. Furthermore, TDMs requires fewer samples than the model-free baselines on ant tasks and drastically fewer samples on the other tasks.



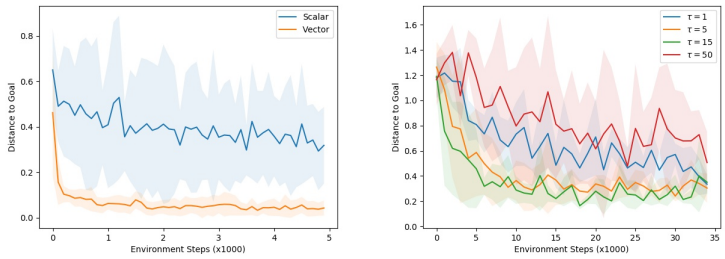
(a) 7-Dof Reacher (b) Pusher (c) Half Cheetah

(d) Ant: Position (e) Ant: Position and Velocity (f) Sawyer Robot (Real-world)

Fig. 5.2 The comparison of TDM with the baseline methods in model-free (DDPG), model-based, and goal-conditioned value functions (HER - Dense) on various tasks. All plots show the final distance to the goal versus 1000 environment steps (not rollouts). The bold line shows the mean across 3 random seeds, and the shaded region show one standard deviation. Our method, which uses model-free learning, is generally more sample-efficient than model-free alternatives including DDPG and HER and improves upon the best model-based performance.

We also see that using HER does not lead to an improvement over DDPG. While we were initially surprised, we realized that a selling point of HER is that it can solve sparse tasks that would otherwise be unsolvable. In this chapter, we were interested in improving the sample efficiency and not the feasibility of model-free reinforcement learning algorithms, and so we focused on tasks that DDPG could already solve. In these sorts of tasks, the advantage of HER over DDPG with a dense reward is not expected. To evaluate HER as a method to solve sparse tasks, we included the DDPG-Sparse baseline and we see that HER significantly outperforms it as expected. In summary, TDMs converge as fast or faster than model-based (which learns faster than the model-free baselines), while learning as good or better final policy than model-free baselines on all tasks.

Lastly, we ran the algorithm on a 7-DoF Sawyer robotic arm to learn a real-world analogue of the reaching task. Figure 5.2f shows that the algorithm outperforms and learns with fewer samples than DDPG, our model-free baseline. These results show that TDM can scale to real-world tasks.



(a) Scalar vs Vectorized TDMs (b) TDMs with different τ_{max}

Fig. 5.3 Ablation experiments for (a) scalar vs. vectorized TDMs on 7-DoF simulated reacher task and (b) different τ_{max} on pusher task. The vectorized variant performs substantially better, while the horizon effectively interpolates between model-based and model-free learning.

5.5.2 Ablation Studies

In this section, we discuss two key design choices for TDMs that provide substantially improved performance. First, Figure 5.3a examines the tradeoffs between the vectorized and scalar rewards. The results show that the vectorized formulation learns substantially faster than the naïve scalar variant. Second, Figure 5.3b compares the learning speed for different horizon values τ_{max}. Performance degrades when the horizon is too low, and learning becomes slower when the horizon is too high. Interestingly, the performance gap between $\tau_{max} = 0$ and $\tau_{max} = \{5, 15\}$ seems to be consistent with the performance gap observed in Figure 5.2b between TDM and model-based learning, suggesting that TDMs are effectively interpolating between model-based and model-free learning.

5.6 Conclusion

In this chapter, we derive a connection between model-based and model-free reinforcement learning, and present a novel RL algorithm that exploits this connection to greatly improve on the sample efficiency of state-of-the-art model-free deep RL algorithms. Our temporal difference models can be viewed both as goal-conditioned value functions and implicit dynamics models, which enables them to be trained efficiently on off-policy data while still minimizing the effects of model bias. As a result, they achieve asymptotic performance that compares favorably with model-free algorithms, but with a sample complexity that is comparable to purely model-based methods.

While the experiments focus primarily on the new RL algorithm, the relationship between model-based and model-free RL explored in this chapter provides a number of avenues for future work. We demonstrated the use of TDMs with a very basic planning approach, but

further exploring how TDMs can be incorporated into powerful constrained optimization methods for model-predictive control or trajectory optimization is an exciting avenue for future work. Another direction for future is to further explore how TDMs can be applied to complex state representations, such as images, where simple distance metrics may no longer be effective. Although direct application of TDMs to these domains is not straightforward, a number of works have studied how to construct metric embeddings of images that could in principle provide viable distance functions. We also note that while the presentation of TDMs have been in the context of deterministic environments, the extension to stochastic environments is straightforward: TDMs would learn to predict the *expected* distance between the future state and a goal state. Finally, the promise of enabling sample-efficient learning with the performance of model-free RL and the efficiency of model-based RL is to enable widespread RL application on real-world systems. Many applications in robotics, autonomous driving and flight, and other control domains could be explored in future work.

Chapter 6

Concluding Remark

This thesis outlined three approaches to bridge model-based with model-free, and on-policy with off-policy reinforcement learning, in order to improve stability and sample-efficiency of the learning algorithms. Normalized advantage functions (NAF) with model-based acceleration in Chapter 3 combined a novel variant of Q-learning with iLQG-based model samples to accelerate convergence of Q-function. Interpolated policy gradient (IPG) in Chapter 4 connected on-policy Monte Carlo policy gradient with off-policy actor-critic, and derived a family of policy gradient methods with theoretical bounds on biases. Temporal difference models (TDM) in Chapter 5 generalized off-policy value-based approaches to include model-based approaches as a special case, and derived a novel temporal-extended variant of optimal control. Importantly, in each of the scenarios, by carefully combining two approaches, we could achieve the best of both worlds and substantially improve upon either of the approaches. We believe that more novel and impactful algorithmic improvements can be further discovered by unifying and interpolating different branches of RL algorithms, and hope our work could be one of the building blocks toward developing the most intelligent – versatile and sample-efficient – learning algorithm.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. *arXiv preprint arXiv:1707.01495*.

Asadi, K., Allen, C., Roderick, M., Mohamed, A.-r., Konidaris, G., Littman, M., and Amazon, B. U. (2017). Mean actor critic. *stat*, 1050:1.

Atkeson, C. G., Moore, A. W., and Schaal, S. (1997). Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer.

Bacon, P.-L., Harb, J., and Precup, D. (2017). The option-critic architecture. In *AAAI*, pages 1726–1734.

Bagnell, J. A. and Schneider, J. (2003). Covariant policy search. IJCAI.

Baird III, L. C. (1993). Advantage updating. Technical report, DTIC Document.

Bansal, S., Calandra, R., Xiao, T., Levine, S., and Tomlin, C. J. (2017). Goal-driven dynamics learning via bayesian optimization. *arXiv preprint arXiv:1703.09260*.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*.

Bhatnagar, S., Precup, D., Silver, D., Sutton, R. S., Maei, H. R., and Szepesvári, C. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*, pages 1204–1212.

Boyan, J. A. (1999). Least-squares temporal difference learning. In *Proceedings of the 16th International Conference on Machine Learning*, pages 49–56.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine learning*, 22(1-3):33–57.

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

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

Chebotar, Y., Hausman, K., Zhang, M., Sukhatme, G., Schaal, S., and Levine, S. (2017a). Combining model-based and model-free updates for trajectory-centric reinforcement learning. *arXiv preprint arXiv:1703.03078*.

Chebotar, Y., Kalakrishnan, M., Yahya, A., Li, A., Schaal, S., and Levine, S. (2017b). Path integral guided policy search. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3381–3388. IEEE.

Ciosek, K. and Whiteson, S. (2018). Expected policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Da Silva, B., Konidaris, G., and Barto, A. (2012). Learning parameterized skills. *arXiv preprint arXiv:1206.6398*.

Dayan, P. and Hinton, G. E. (1993). Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278.

Degris, T., White, M., and Sutton, R. S. (2012). Off-policy actor-critic. *International Conference on Machine Learning (ICML)*.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

Deisenroth, M. P., Neumann, G., Peters, J., et al. (2013). A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142.

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

Dosovitskiy, A. and Koltun, V. (2016). Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. *International Conference on Machine Learning (ICML)*.

Dudík, M., Langford, J., and Li, L. (2011). Doubly robust policy evaluation and learning. *arXiv preprint arXiv:1103.4601*.

Eysenbach, B., Gu, S., Ibarz, J., and Levine, S. (2018). Leave no trace: Learning to reset for safe and autonomous reinforcement learning. *International Conference for Learning Representations*.

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

Fu, J., Levine, S., and Abbeel, P. (2015). One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. *arXiv preprint arXiv:1509.06841*.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

Gu, S., Ghahramani, Z., and Turner, R. E. (2015). Neural adaptive sequential monte carlo. In *Advances in Neural Information Processing Systems*, pages 2629–2637.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017a). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE.

Gu, S., Levine, S., Sutskever, I., and Mnih, A. (2016a). Muprop: Unbiased backpropagation for stochastic neural networks. *International Conference on Learning Representations (ICLR)*.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. (2017b). Q-prop: Sample-efficient policy gradient with an off-policy critic. *International Conference for Learning Representations*.

Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016b). Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838.

Gu, S., Lillicrap, T., Turner, R. E., Ghahramani, Z., Schölkopf, B., and Levine, S. (2017c). Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3849–3858.

Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. *Neural Information Processing Systems (NIPS)*.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1352–1361. JMLR. org.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

Hafner, R. and Riedmiller, M. (2011). Reinforcement learning in feedback control. *Machine learning*, 84(1-2):137–169.

Harmon, M. E. and Baird III, L. C. (1996). Multi-player residual advantage learning with general function approximation. *Wright Laboratory, WL/AACF, Wright-Patterson Air Force Base, OH*, pages 45433–7308.

Hasselt, H. V. (2010). Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621.

Heess, N., Silver, D., and Teh, Y. W. (2012). Actor-critic reinforcement learning with energy-based policies. In *EWRL*, pages 43–58.

Heess, N., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, A., Riedmiller, M., et al. (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. (2015a). Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Tassa, Y., and Erez, T. (2015b). Learning Continuous Control Policies by Stochastic Value Gradients. *arXiv*, pages 1–13.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Howard, R. A. (1964). Dynamic programming and markov processes.

Hunter, D. R. and Lange, K. (2004). A tutorial on mm algorithms. *The American Statistician*, 58(1):30–37.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. *International Conference on Learning Representations*.

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with gumbel-softmax. *International Conference on Learning Representations (ICLR)*.

Jaques, N., Gu, S., Bahdanau, D., Hernández-Lobato, J. M., Turner, R. E., and Eck, D. (2017). Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control. *International Conference on Machine Learning (ICML)*.

Jiang, N. and Li, L. (2016). Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 652–661.

Jie, T. and Abbeel, P. (2010). On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems*, pages 1000–1008.

Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the tenth international conference on machine learning*, volume 951, pages 167–173.

Kahn, G., Zhang, T., Levine, S., and Abbeel, P. (2016). Plato: Policy learning using adaptive trajectory optimization. *arXiv preprint arXiv:1603.00622*.

Kakade, S. (2001). A natural policy gradient. In *NIPS*, volume 14, pages 1531–1538.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 2, pages 267–274.

Kalakrishnan, M., Righetti, L., Pastor, P., and Schaal, S. (2011). Learning force control policies for compliant manipulation. In *International Conference on Intelligent Robots and Systems (IROS)*.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *ICLR*.

Kober, J. and Peters, J. (2012). Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. (2018). Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*.

Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149.

Lagoudakis, M. G., Parr, R., and Littman, M. L. (2002). Least-squares methods in reinforcement learning for control. In *Hellenic conference on artificial intelligence*, pages 249–260. Springer.

Legg, S. and Hutter, M. (2007). Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17(4):391–444.

Lever, G. (2014). Deterministic policy gradient algorithms.

Levine, S. and Abbeel, P. (2014). Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1071–1079.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40.

Levine, S. and Koltun, V. (2013). Guided policy search. In *International Conference on Machine Learning (ICML)*, pages 1–9.

Li, W. and Todorov, E. (2004). Iterative linear quadratic regulator design for nonlinear biological movement systems.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *ICLR*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321.

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

Metz, L., Ibarz, J., Jaitly, N., and Davidson, J. (2017). Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*.

Mishra, N., Abbeel, P., and Mordatch, I. (2017). Prediction and control with temporal segment models.

Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. *International Conference on Machine Learning (ICML)*.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. G. (2016). Safe and efficient off-policy reinforcement learning. *arXiv preprint arXiv:1606.02647*.

Nachum, O., Gu, S., Lee, H., and Levine, S. (2018). Data-efficient hierarchical reinforcement learning. *Neural Information Processing Systems*.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. (2017). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*.

Nguyen, D. H. and Widrow, B. (1990). Neural networks for self-learning control systems. *IEEE Control systems magazine*, 10(3):18–23.

O'Donoghue, B., Munos, R., Kavukcuoglu, K., and Mnih, V. (2017). Pgq: Combining policy gradient and q-learning. *ICLR*.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *International Conference on Machine learning*.

Parr, R. and Russell, S. J. (1998). Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, pages 1043–1049.

Peshkin, L. and Shelton, C. R. (2002). Learning from scarce experience. *arXiv preprint cs/0204043*.

Peters, J., Mülling, K., and Altun, Y. (2010). Relative entropy policy search. In *AAAI*. Atlanta.

Peters, J. and Schaal, S. (2006). Policy gradient methods for robotics. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2219–2225. IEEE.

Pong, V., Gu, S., Dalal, M., and Levine, S. (2018). Temporal difference models: Model-free deep rl for model-based control. *International Conference for Learning Representations*.

Precup, D. (2000). Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80.

Rajeswaran, A., Lowrey, K., Todorov, E. V., and Kakade, S. M. (2017). Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems*, pages 6550–6561.

Rawlik, K., Toussaint, M., and Vijayakumar, S. (2013). On stochastic optimal control and reinforcement learning by approximate inference. *Robotics*, page 353.

Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer.

Ross, S., Gordon, G. J., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, page 6.

Ross, S. M. (2006). Simulation. *Burlington, MA: Elsevier*.

Salge, C., Glackin, C., and Polani, D. (2014). Empowerment–an introduction. In *Guided Self-Organization: Inception*, pages 67–114. Springer.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015a). Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1312–1320.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015b). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Schmidhuber, J. (1990). An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 253–258. IEEE.

Schmidhuber, J. (1991). Reinforcement learning in markovian and non-markovian environments. pages 500–506.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. (2015a). Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. (2015b). Trust region policy optimization. In *ICML*, pages 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. *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*.

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.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., et al. (2017). The predictron: End-to-end learning and planning. *International Conference on Machine Learning*.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *International Conference on Machine Learning*, pages 216–224.

Sutton, R. S. (2000). All-action policy gradient. *Unpublished*.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. (2009a). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000. ACM.

Sutton, R. S., Maei, H. R., and Szepesvári, C. (2009b). A convergent $o(n)$ temporal-difference algorithm for off-policy learning with linear function approximation. In *Advances in neural information processing systems*, pages 1609–1616.

Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999a). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, volume 99, pages 1057–1063.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems.

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

Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103.

Tassa, Y., Erez, T., and Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4906–4913. IEEE.

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

Theodorou, E., Buchli, J., and Schaal, S. (2010). Learning policy improvements with path integrals. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 828–835.

Thomas, P. (2014). Bias in natural actor-critic algorithms. In *ICML*, pages 441–448.

Thomas, P. and Brunskill, E. (2016). Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

Todorov, E. and Li, W. (2005). A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306. IEEE.

Tripuraneni, N., Gu, S., Ge, H., and Ghahramani, Z. (2015). Particle gibbs for infinite hidden markov models. In *Advances in Neural Information Processing Systems*, pages 2395–2403.

Tsitsiklis, J. and Van Roy, B. (1996). An analysis of temporal-difference learning with function approximationtechnical. Technical report, Report LIDS-P-2322). Laboratory for Information and Decision Systems, Massachusetts Institute of Technology.

Tucker, G., Bhupatiraju, S., Gu, S., Turner, R. E., Ghahramani, Z., and Levine, S. (2018). The mirage of action-dependent baselines in reinforcement learning. *International Conference on Machine Learning*.

Van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. *CoRR, abs/1509.06461*.

Venkatraman, A., Capobianco, R., Pinto, L., Hebert, M., Nardi, D., and Bagnell, J. A. (2016). Improved learning of dynamics models for control. In *International Symposium on Experimental Robotics*.

Wahlström, N., Schön, T. B., and Deisenroth, M. P. (2015). From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2017). Sample efficient actor-critic with experience replay. *International Conference on Learning Representations (ICLR)*.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. *International Conference on Machine Learning (ICML)*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2728–2736.

Wawrzyński, P. (2009). Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural Networks*, 22(10):1484–1497.

Weaver, L. and Tao, N. (2001). The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 538–545. Morgan Kaufmann Publishers Inc.

Werbos, P. J. (1989). Neural networks for control and system identification. In *Decision and Control, 1989., Proceedings of the 28th IEEE Conference on*, pages 260–265. IEEE.

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

Zhang, T., Kahn, G., Levine, S., and Abbeel, P. (2016). Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 528–535. IEEE.

Appendix A

Supplementary Materials for Chapter 4

A.1 Proof for Theorem 1

A.1.1 Local approximation objective with bounded bias

In the main paper, we introduced the approximate objective $\tilde{J}(\pi, \tilde{\pi})$ to $J(\pi)$ during our theoretical analysis. In this section, we discuss the motivations behind this choice, referencing the prior work (Kakade and Langford, 2002; Schulman et al., 2015b).

First, the expected return $J(\pi)$ of a policy π can be written as the sum of the expected return $J(\tilde{\pi})$ of another policy $\tilde{\pi}$ and the expected advantage term between the two policies in the equation, where $A^{\tilde{\pi}}(s_t, a_t)$ is the advantage of policy $\tilde{\pi}$,

$$J(\pi) = J(\tilde{\pi}) + \mathbb{E}_{\rho^{\pi}, \pi}[A^{\tilde{\pi}}(s_t, a_t)].$$

For the proof, see Lemma 1 in (Schulman et al., 2015b). This expression is still not tractable to analyze because of the dependency of unnormalized state sampling distribution ρ^{π} on π. Kakade and Langford (2002); Schulman et al. (2015b) thus introduce a local approximation by replacing ρ^{π} with $\rho^{\tilde{\pi}}$,

$$J(\pi) \approx J(\tilde{\pi}) + \mathbb{E}_{\rho^{\tilde{\pi}}, \pi}[A^{\tilde{\pi}}(s_t, a_t)] \triangleq \tilde{J}(\pi, \tilde{\pi}).$$

We can show that $J(\pi) = \tilde{J}(\pi, \tilde{\pi} = \pi)$ and $\nabla_{\pi} J(\pi) = \nabla_{\pi} \tilde{J}(\pi, \tilde{\pi} = \pi)$, meaning that the $J(\pi)$ and $\tilde{J}(\pi, \tilde{\pi})$ match up to the first order terms. Schulman et al. (2015b) then uses this property, in combination with minorization-maximization Hunter and Lange (2004), to derive a monotonic convergence proof for a variant of policy iteration algorithm. To start our proof for Theorem 1, we first derive the following lemma,

Lemma 3 *If* $\zeta = \max_s |\bar{A}^{\pi,\tilde{\pi}}(s)|$, *then*

$$\left\|J(\pi) - \tilde{J}(\pi,\tilde{\pi})\right\|_1 \leq 2\zeta \frac{\gamma}{(1-\gamma)^2} D_{TV}^{\max}(\tilde{\pi},\pi) \leq 2\zeta \frac{\gamma}{(1-\gamma)^2} \sqrt{D_{KL}^{\max}(\tilde{\pi},\pi)}$$

Proof. We define $\rho_t^{\pi}(s_t)$ as the marginal state distribution at time t assuming that the agent follows policy π from initial state $\rho_0(s_t)$ at time $t = 0$. Note that from the definition of ρ^{π}, $\rho^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t \rho_t^{\pi}(s_t = s)$. We can use the following lemma from Kahn et al. (2016), which is adapted from Ross et al. (2011) and Schulman et al. (2015b).

Lemma 4 *(Kahn et al., 2016)*

$$\left\|\rho_t^{\pi} - \rho_t^{\beta}\right\|_1 \leq 2tD_{TV}^{\max}(\pi,\beta) \leq 2t\sqrt{D_{KL}^{\max}(\pi,\beta)} \tag{A.1}$$

Using Lemma 4, the full proof for Lemma 3 is provided below, where $\bar{A}^{\pi,\tilde{\pi}}(s) = \mathbb{E}_{\pi}[A^{\tilde{\pi}}(s_t,a_t)]$ and $A^{\tilde{\pi}}(s_t,a_t)$ is the advantage function of $\tilde{\pi}$,

$$\begin{aligned}
&\left\|J(\pi) - \tilde{J}(\pi,\tilde{\pi})\right\|_1 \\
&= \left\|\mathbb{E}_{\rho^{\tilde{\pi}}}[\bar{A}^{\pi,\tilde{\pi}}(s)] - \mathbb{E}_{\rho^{\pi}}[\bar{A}^{\pi,\tilde{\pi}}(s)]\right\|_1 \\
&\leq \sum_{t=0}^{\infty} \gamma^t \left\|\mathbb{E}_{\rho_t^{\tilde{\pi}}}[\bar{A}^{\pi,\tilde{\pi}}(s)] - \mathbb{E}_{\rho_t^{\pi}}[\bar{A}^{\pi,\tilde{\pi}}(s)]\right\|_1 \\
&\leq \zeta \sum_{t=0}^{\infty} \gamma^t \left\|\rho_t^{\tilde{\pi}} - \rho_t^{\pi}\right\|_1 \\
&\leq 2\zeta \left(\sum_{t=0}^{\infty} \gamma^t t\right) D_{TV}^{\max}(\tilde{\pi},\pi) \\
&= 2\zeta \frac{\gamma}{(1-\gamma)^2} D_{TV}^{\max}(\tilde{\pi},\pi) \\
&\leq 2\zeta \frac{\gamma}{(1-\gamma)^2} \sqrt{D_{\mathrm{KL}}^{\max}(\pi,\tilde{\pi})}.
\end{aligned} \tag{A.2}$$

This lemma is crucial in our theoretical analysis, as it allows us to tractably bound the biases of the full spectrum of local IPG objectives $\tilde{J}^{\beta,\nu,CV}(\pi,\tilde{\pi})$ against $J(\pi)$.

A.1.2 Main proof for Theorem 1

Proof. We first prove the bound for $\left\|\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu=0,CV}(\pi,\tilde{\pi})\right\|_1$. Using Lemma 4, the bound is given below, with a similar derivation process as in Lemma 3.

$$
\begin{aligned}
&\left\|\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu=0,CV}(\pi,\tilde{\pi})\right\|_1 \\
&= \left\|J(\tilde{\pi})+\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)]-J(\tilde{\pi})-\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)-A^{\tilde{\pi}}_w(s_t,a_t)]-\mathbb{E}_{\rho^{\beta}}[\bar{A}^{\pi,\tilde{\pi}}_w(s_t)]\right\|_1 \\
&= \left\|\mathbb{E}_{\rho^{\tilde{\pi}}}[\bar{A}^{\pi,\tilde{\pi}}_w(s)]-\mathbb{E}_{\rho^{\beta}}[\bar{A}^{\pi,\tilde{\pi}}_w(s)]\right\|_1 \\
&\leq \sum_{t=0}^{\infty}\gamma^t\left\|\mathbb{E}_{\rho^{\tilde{\pi}}_t}[\bar{A}^{\pi,\tilde{\pi}}_w(s)]-\mathbb{E}_{\rho^{\beta}_t}[\bar{A}^{\pi,\tilde{\pi}}_w(s)]\right\|_1 \\
&\leq \varepsilon\sum_{t=0}^{\infty}\gamma^t\left\|\rho^{\tilde{\pi}}_t-\rho^{\beta}_t\right\|_1 \\
&\leq 2\varepsilon(\sum_{t=0}^{\infty}\gamma^t t)D^{\max}_{TV}(\tilde{\pi},\beta) \\
&= 2\varepsilon\frac{\gamma}{(1-\gamma)^2}D^{\max}_{TV}(\tilde{\pi},\beta) \\
&\leq 2\varepsilon\frac{\gamma}{(1-\gamma)^2}\sqrt{D^{\max}_{\mathrm{KL}}(\tilde{\pi},\beta)}.
\end{aligned}
\tag{A.3}
$$

Given this bound, we can directly derive the bound for $\left\|\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu=0,CV}(\pi,\tilde{\pi})\right\|_1$ by combining with Lemma 3,

$$
\begin{aligned}
&\left\|J(\pi)-\tilde{J}^{\beta,\nu=0,CV}(\pi,\tilde{\pi})\right\|_1 \\
&\left\|J(\pi)-\tilde{J}(\pi,\tilde{\pi})+\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu=0,CV}(\pi,\tilde{\pi})\right\|_1 \\
&\leq \left\|\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu=0,CV}(\pi,\tilde{\pi})\right\|_1+\left\|J(\pi)-\tilde{J}(\pi,\tilde{\pi})\right\|_1 \\
&\leq 2\frac{\gamma}{(1-\gamma)^2}\left(\varepsilon\sqrt{D^{\max}_{\mathrm{KL}}(\tilde{\pi},\beta)}+\zeta\sqrt{D^{\max}_{\mathrm{KL}}(\pi,\tilde{\pi})}\right)
\end{aligned}
\tag{A.4}
$$

A.2 Algorithm with Monotonic Convergence Property and its Proof

Algorithm 5 is a special case of IPG, $\tilde{J}^{\beta_i,\nu=0,CV}(\pi,\pi_i)$. We can prove that Algorithm 5 guarantees monotonic improvement, even with off-policy sample usage and imperfect critic Q_w or A_w. This is an interesting result, since most of the prior work have shown such property

Algorithm 5 Policy iteration with non-decreasing returns $J(\pi)$ and bounded off-policy sampling

1: Initialize policy π_0, and critic Q_w
2: **repeat**
3: Compute all advantage values $A^{\pi_i}(s,a)$, and choose any off-policy distribution β_i
4: Update critic Q_w using any method (no requirement for performance)
5: Solve the constrained optimization problem:
6: $\pi_{i+1} \leftarrow \arg\max_\pi \tilde{J}^{\beta_i,\nu=0,CV}(\pi,\pi_i) - C\left(\zeta\sqrt{D_{KL}^{\max}(\pi,\pi_i)} + \varepsilon\sqrt{D_{KL}^{\max}(\pi_i,\beta_i)}\right)$
7: subject to $\sum_a \pi(a|s) = 1 \quad \forall s$
8: where $C = \frac{2\gamma}{(1-\gamma)^2}, \zeta = \max_s |\bar{A}^{\pi,\tilde{\pi}}(s)|, \varepsilon = \max_s |\bar{A}_w^{\pi,\tilde{\pi}}(s)|$
9: **until** π_i converges.

only for purely on-policy policy gradient methods Kakade and Langford (2002); Schulman et al. (2015b). We begin by first introducing the following corollary,

Corollary 1

$$J(\pi) \geq M(\pi,\tilde{\pi}) \geq M^{\beta,\nu=0,CV}(\pi,\tilde{\pi}), J(\tilde{\pi}) = M(\tilde{\pi},\tilde{\pi}) = M^{\beta,\nu=0,CV}(\tilde{\pi},\tilde{\pi}) \tag{A.5}$$

where

$$M(\pi,\tilde{\pi}) = \tilde{J}(\pi,\tilde{\pi}) - C\zeta\sqrt{D_{KL}^{\max}(\pi,\tilde{\pi})}$$
$$M^{\beta,\nu=0,CV}(\pi,\tilde{\pi}) = \tilde{J}^{\beta,\nu=0}(\pi,\tilde{\pi}) - C(\zeta\sqrt{D_{KL}^{\max}(\pi,\tilde{\pi})} + \varepsilon\sqrt{D_{KL}^{\max}(\tilde{\pi},\beta)})$$

Proof. It follows from Theorem 1 in the main text and Theorem 1 in Schulman et al. (2015b). $J(\tilde{\pi}) = M^{\beta,\nu=0,CV}(\tilde{\pi},\tilde{\pi})$ since $\zeta = \varepsilon = 0$ when $\pi = \tilde{\pi}$.

Given Corollary 1, we use minorization-maximization (MM) (Hunter and Lange, 2004) to derive Algorithm 2, a policy iteration algorithm that allows using off-policy samples while guaranteeing monotonic improvement on $J(\pi)$. MM suggests that at each iteration, by maximizing the lower bound, or the minorizer, of the objective, the algorithm can guarantee monotonic improvement: $J(\pi_{i+1}) \geq M^{\beta_i,\nu=0,CV}(\pi_{i+1},\pi_i) \geq M^{\beta_i,\nu=0,CV}(\pi_i,\pi_i) = J(\pi_i)$, where $\pi_{i+1} \leftarrow \arg\max_\pi M^{\beta_i,\nu=0,CV}(\pi,\pi_i)$. Importantly, the algorithm guarantees monotonic improvement regardless of the off-policy distribution β_i or the performance of the critic Q_w. This result is a step toward achieving off-policy policy gradient with convergence guarantee of on-policy algorithms.[1]

[1] Schulman et al. (2015b) applies additional bound, $\varepsilon \geq 2\varepsilon'\sqrt{D_{KL}^{\max}(\pi,\tilde{\pi})}$ where $\varepsilon' = \max_{s,a}|A_w^{\tilde{\pi}}(s,a)|$ to remove dependency on π. In our case, we cannot apply such bound on ζ, since then the inequality in Theorem 1 is still satisfied but the equality is violated, and thus the algorithm no longer guarantees monotonic improvement.

We compare our theoretical algorithm with Algorithm 1 in Schulman et al. (2015b), which guarantees monotonic improvement in a general on-policy policy gradient algorithm. The main difference is the additional term, $-C\varepsilon\sqrt{D_{KL}^{\max}(\tilde{\pi},\beta)}$ to the lower bound. $D_{KL}^{\max}(\tilde{\pi},\beta)$ is constant with respect to π, while $\varepsilon = 0$ if $\pi = \tilde{\pi}$ and $\varepsilon \geq 0$ if otherwise. This suggests that as β becomes more off-policy, the gap between the lower bound and the true objective widens, proportionally to $\sqrt{D_{KL}^{\max}(\tilde{\pi},\beta)}$. This may make each majorization step end in a place very close to where it started, i.e. π_{i+1} very close to π_i, and slow down learning. This again suggests a trade-off that comes in as off-policy samples are used.

A.3 Proof for Theorem 2

We follow the same procedure as the proof for Theorem 1, where we first derive bounds between $\tilde{J}(\pi,\tilde{\pi})$ and the other local objectives, and then combine the results with Lemma 3.

To begin the proof, we first derive the bound for the special case where $\nu = 1$. Having $\nu = 1$, we remove the likelihood ratio policy gradient term, and get the following gradient expression,

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\rho^\beta}[\nabla_\theta \bar{Q}_w^\pi(s_t)]. \tag{A.6}$$

This is an off-policy actor-critic algorithm, and is closely connected to DDPG (Lillicrap et al., 2016), except that it does not use target policy network and its use of a stochastic policy enables on-policy exploration, trust-region policy updates, and no heuristic additive exploration noise.

We can introduce the following bound on the local objective $\tilde{J}^{\beta,\nu=1}(\pi,\tilde{\pi})$, whose policy gradient equals A.6 at $\pi = \tilde{\pi}$, similarly to the proof for Theorem 1 in the main text.

Corollary 2 *If* $\delta = \max_{s,a}|A^{\tilde{\pi}}(s,a) - A_w^{\tilde{\pi}}(s,a)|$, $\varepsilon = \max_s |\bar{A}_w^{\pi,\tilde{\pi}}(s)|$, *and*

$$\tilde{J}^{\beta,\nu=1}(\pi,\tilde{\pi}) = J(\tilde{\pi}) + \mathbb{E}_{\rho^\beta}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)] \tag{A.7}$$

then,

$$\left\| \tilde{J}(\pi,\tilde{\pi}) - \tilde{J}^{\beta,\nu=1}(\pi,\tilde{\pi}) \right\|_1 \leq \frac{\delta}{1-\gamma} + 2\varepsilon\frac{\gamma}{(1-\gamma)^2}\sqrt{D_{KL}^{\max}(\tilde{\pi},\beta)} \tag{A.8}$$

Proof. We note that

$$
\begin{aligned}
&\left\|\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu=1}(\pi,\tilde{\pi})\right\|_1 \\
&=\left\|\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)-A_w^{\tilde{\pi}}(s_t,a_t)]+\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu=0}(\pi,\tilde{\pi})\right\|_1 \\
&\leq\left\|\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)-A_w^{\tilde{\pi}}(s_t,a_t)]\right\|_1+\left\|\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu=0}(\pi,\tilde{\pi})\right\|_1 \\
&\leq\sum_{t=0}^{\infty}\gamma^t\left\|\mathbb{E}_{\rho_t^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)-A_w^{\tilde{\pi}}(s_t,a_t)]\right\|_1+\left\|\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu=0}(\pi,\tilde{\pi})\right\|_1 \\
&\leq\delta\sum_{t=0}^{\infty}\gamma^t+\left\|\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu=0}(\pi,\tilde{\pi})\right\|_1 \\
&=\frac{\delta}{1-\gamma}+\left\|\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu=0}(\pi,\tilde{\pi})\right\|_1 \\
&\leq\frac{\delta}{1-\gamma}+2\varepsilon\frac{\gamma}{(1-\gamma)^2}\sqrt{D_{\mathrm{KL}}^{\max}(\tilde{\pi},\beta)},
\end{aligned}
\tag{A.9}
$$

where the proof uses Theorem 1 at the last step.

Given Corollary 2 and Theorem 1, we are ready to prove the two bounds in Theorem 2.

Proof.

$$
\begin{aligned}
&\left\|\tilde{J}(\pi,\tilde{\pi})-\tilde{J}^{\beta,\nu}(\pi,\tilde{\pi})\right\|_1 \\
&=\left\|J(\tilde{\pi})+\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)]-J(\tilde{\pi})-(1-\nu)\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)]-\nu\mathbb{E}_{\rho^{\beta}}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)]\right\|_1 \\
&=\nu\left\|\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)]-\mathbb{E}_{\rho^{\beta}}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)]\right\|_1 \\
&=\nu\left\|\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)]-\mathbb{E}_{\rho^{\pi}}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)]+\mathbb{E}_{\rho^{\pi}}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)]-\mathbb{E}_{\rho^{\beta}}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)]\right\|_1 \\
&\leq\nu\left\|\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)]-\mathbb{E}_{\rho^{\pi}}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)]\right\|_1+\nu\left\|\mathbb{E}_{\rho^{\pi}}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)]-\mathbb{E}_{\rho^{\beta}}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)]\right\|_1 \\
&=\nu\left\|\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)-\bar{A}_w^{\tilde{\pi}}(s_t,a_t)]\right\|_1+\nu\left\|\mathbb{E}_{\rho^{\pi}}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)]-\mathbb{E}_{\rho^{\beta}}[\bar{A}_w^{\pi,\tilde{\pi}}(s_t)]\right\|_1 \\
&\leq\frac{\nu\delta}{1-\gamma}+2\varepsilon\frac{\nu\gamma}{(1-\gamma)^2}\sqrt{D_{\mathrm{KL}}^{\max}(\tilde{\pi},\beta)}
\end{aligned}
\tag{A.10}
$$

$$
\begin{aligned}
&\left\| \tilde{J}(\pi,\tilde{\pi}) - \tilde{J}^{\beta,\nu,CV}(\pi,\tilde{\pi}) \right\|_1 \\
&= \left\| J(\tilde{\pi}) + \mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)] - J(\tilde{\pi}) - (1-\nu)\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t) - \bar{A}^{\tilde{\pi}}_w(s_t,a_t)] - \mathbb{E}_{\rho^{\beta}}[\bar{A}^{\pi,\tilde{\pi}}_w(s_t)] \right\|_1 \\
&= \left\| \nu(\mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)] - \mathbb{E}_{\rho^{\pi}}[\bar{A}^{\pi,\tilde{\pi}}_w(s_t)]) + \mathbb{E}_{\rho^{\pi}}[\bar{A}^{\pi,\tilde{\pi}}_w(s_t)] - \mathbb{E}_{\rho^{\beta}}[\bar{A}^{\pi,\tilde{\pi}}_w(s_t)] \right\|_1 \\
&\leq \nu \left\| \mathbb{E}_{\rho^{\tilde{\pi}},\pi}[A^{\tilde{\pi}}(s_t,a_t)] - \mathbb{E}_{\rho^{\pi}}[\bar{A}^{\pi,\tilde{\pi}}_w(s_t)] \right\|_1 + \left\| \mathbb{E}_{\rho^{\pi}}[\bar{A}^{\pi,\tilde{\pi}}_w(s_t)] - \mathbb{E}_{\rho^{\beta}}[\bar{A}^{\pi,\tilde{\pi}}_w(s_t)] \right\|_1 \\
&\leq \frac{\nu\delta}{1-\gamma} + 2\varepsilon\frac{\gamma}{(1-\gamma)^2}\sqrt{D^{\max}_{\mathrm{KL}}(\tilde{\pi},\beta)}.
\end{aligned}
\tag{A.11}
$$

We combine these bounds with Lemma 3 to conclude the proof.

A.4 Supplementary Experimental Details

A.4.1 Hyperparameters

GAE($\lambda = 0.97$) (Schulman et al., 2016) is used for $\hat{A}$ estimation. Trust-region update in TRPO is used as the policy optimizer (Schulman et al., 2015b). The standard Q-fitting routine from DDPG (Lillicrap et al., 2016) is used for fitting Q_w, where Q_w is trained with batch size 64, using experience replay of size $1e6$, and target network with $\tau = 0.001$. ADAM (Kingma and Ba, 2014) is used as the optimizer for Q_w. Policy network parametrizes a Gaussian policy with $\pi_\theta(a_t|s_t) = \mathcal{N}(\mu_\theta(s_t), \Sigma_\theta)$, where μ_θ is a two-hidden-layer neural network of size $100-50$ and tanh hidden nonlinearity and linear output, and Σ_θ is a diagonal, state-independent variance. For DDPG, the policy network is deterministic and additionally has tanh activation at the output layer. The critic function Q_w is a two-hidden-layer neural network of size $100-100$ with ReLU activation. For the reparameterized control variates, we use Monte Carlo sample size $m = 1$.

The trust-region step size for policy update is fixed to 0.1 for HalfCheetah-v1 and Humanoid-v1, and 0.01 for Ant-v1 and Walker2d-v1, while the learning rate for ADAM in critic update is fixed to $1e^{-4}$ for HalfCheetah-v1, Ant-v1, Humanoid-v1, and $1e^{-3}$ for Walker2d-v1. Those two hyperparameters are found by first running TRPO and DDPG on each domain, and picking the ones that give best performance for each domain. These parameters are fixed throughout the experiment to ensure fair comparisons.

As in the Q-Prop implementation (Gu et al., 2017b), the residual learning signal in the first term is normalized to be zero mean and unit variance. This introduces additional bias to the gradient estimator, but the bias can be theoretically analyzed by substituting the bounds

with new $\nu' = 1 - \frac{1-\nu}{\sigma}$ in the IPG expressions where σ is the empirical standard deviation of the (residual) learning signal.

The plots in the main text present the mean returns as solid lines, scatter plots of all runs in the background to visualize variability. For X-axis, one “episode" corresponds to 1000 transitions, which is the default maximum episode length for all domains in our experiments. Importantly, “Episodes” do not correspond to actual numbers of episodes taken for Ant-v1, Walker-v1, and Humanoid-v1, since these environments have termination conditions.

Appendix B

Supplementary Materials for Chapter 5

B.1 Experiment Details

In this section, we detail the experimental setups used in our results.

B.1.1 Goal State and τ Sampling Strategy

While Q-learning is valid for any value of s_g and τ for each transition tuple (s_t, a_t, s_{t+1}), the way in which these values are sampled during training can affect learning efficiency. Some potential strategies for sampling s_g are: (1) uniformly sample future states along the actual trajectory in the buffer (i.e., for s_t, choose $s_g = s_{t+k}$ for a random $k > 0$) as in (Andrychowicz et al., 2017); (2) uniformly sample goal states from the replay buffer; (3) uniformly sample goals from a uniform range of valid states. We found that the first strategy performed slightly better than the others, though not by much. In our experiments, we use the first strategy. The horizon τ is sampled uniformly at random between 0 and the maximum horizon τ_{max}.

B.1.2 Tuned Hyperparameters

For TDMs, we found the most important hyperparameters to be the reward scale, τ_{max}, and the number of updates per observations, I. As shown in Figure B.1, TDMs can greatly benefit from larger values of I, though eventually there are diminishing returns and potentially impact, mostly likely due to over-fitting. We found that the baselines did not benefit, except for HER which did benefit from larger I values. For all the model-free algorithms (DDPG, DDPG-Sparse, HER, and TDMs), we performed a grid search over the reward scale in the range $\{0.01, 1, 100, 10000\}$ and the number of updates per observations in the range $\{1, 5, 10\}$. For HER, we also tuned the weight given to the policy pre-tanh-activation $\{0, 0.01, 1\}$, which is

Fig. B.1 TDMs with different number of updates per step I on ant target position task. The maximum distance was set to 5 rather than 6 for this experiment, so the numbers should be lower than the ones reported in the paper.

described in Andrychowicz et al. (2017). For TDMs, we also tuned the best $\tau_{\max}$ in the range $\{15, 25, \text{Horizon} - 1\}$.

B.1.3 Model-free setups

In all our experiments, we used DDPG (Lillicrap et al., 2016) as the base off-policy model-free RL algorithm for learning the TDMs $Q(s, a, g, s_\tau)$. Experience replay (Mnih et al., 2015) has size of 1 million transitions, and the soft target networks (Lillicrap et al., 2016) are used with a polyak averaging coefficient of 0.999 for DDPG and TDM and 0.95 for HER and DDPG-Sparse. For HER and DDPG-Sparse, we also added a penalty on the tanh pre-activation, as in Andrychowicz et al. (2017). Learning rates of the critic and the actor are chosen from {1e-4, 1e-3} and {1e-4,1e-3} respectively. Adam (Kingma and Ba, 2014) is used as the base optimizer with default parameters except the learning rate. The batch size was 128. The policies and networks are parmaeterized with neural networks with ReLU hidden activation and two hidden layers of size 300 and 300. The policies have a tanh output activation, while the critic has no output activation (except for TDM, see B.1.5). For the goal-conditioned value functions, the goal was concatenated to the observation.

While any distance metric for the TDM reward function can be used, we chose L1 norm $-\|s_{t+1} - s_g\|_1$ to ensure that the scalar and vectorized TDMs are consistent.

B.1.4 Model-based setup

For the model-based comparison, we trained a neural network dynamics model with ReLU activation, no output activation, and two hidden units of size 300 and 300. The model was trained to predict the difference in state, rather than the full state. The dynamics model is trained to minimize the mean squared error between the predicted difference and the actual difference. After each state is observed, we sample a minibatch of size 128 from the replay buffer (size 1 million) and perform one step of gradient descent on this mean squared error loss. Twenty rollouts were performed to compute the (per-dimension) mean and standard deviation of the states, actions, and state differences. We used these statistics to normalize the states and actions before giving them to the model, and to normalize the state differences before computing the loss. For MPC, we simulated 512 random action sequences of length

15 through the learned dynamics model and chose the first action of the sequence with the highest reward.

B.1.5 TDM Network Architecture and Vector-based Supervision

For TDMs, since we know that the true Q-function must learn to predict (negative) distances, we incorporate this prior knowledge into the Q-function by parameterizing it as $Q(s,a,s_g,\tau) = -\|f(s,a,s_g,\tau) - s_g\|_1$. Here, f is a vector outputted by a feed-forward neural network and has the same dimension as the goal. This parameterization ensures that the Q-function outputs non-positive values, while encouraging the Q-function to learn what we call a goal-conditioned model: f is encouraged to predict what state will be reached at after τ, when the policy is trying to reach goal s_g.

The scalar supervision regresses

$$Q(s_t,a_t,s_g,\tau) = -\sum_j |f_j(s_t,a_t,s_g,\tau) - s_{g,j}|$$

onto

$$\begin{aligned} & r(s_t,a_t,s_{t+1},s_g) + 1[\tau = 0] + Q(s_{t+1},a^*,s_g,\tau-1)1[\tau \neq 0] \\ = & -\sum_j \left\{ |s - s_{t+1}|1[\tau = 0] + |f_j(s_t,a^*,s_g,\tau-1) - s_{g,j}|1[\tau \neq 0] \right\} \end{aligned}$$

where $a^* = \text{argmax}_a Q(s_{t+1},a,s_g,\tau-1)$. The vectorized supervision instead supervises each component of f, so that

$$|f_j(s_t,a_t,s_g,\tau) - s_{g,j}|$$

regresses onto

$$|s - s_{t+1}|1[\tau = 0] + |f_j(s_t,a^*,s_g,\tau-1) - s_{g,j}|1[\tau \neq 0]$$

for each dimension j of the state.

B.1.6 Task and Reward Descriptions

Benchmark tasks are designed on MuJoCo physics simulator (Todorov et al., 2012) and OpenAI Gym environments (Brockman et al., 2016). For the simulated and pushing tasks, we use (5.8) and for the other tasks we use (5.9) for policy extraction. The horizon (length of episode) for the pusher and ant tasks are 50. The other tasks have a horizon of 100.

7-DoF reacher.: The state consists of 7 joint angles, 7 joint angular velocities, and 3 XYZ observation of the tip of the arm, making it 17 dimensional. The action controls torques for each joint, totally 7 dimensional. The reward function during optimization control and for the model-free baseline is the negative Euclidean distance between the XYZ of the tip and the target XYZ. The targets are sampled randomly from all reachable locations of the arm at the beginning of each episode. The robot model is taken from the striker and pusher environments in OpenAI Gym MuJoCo domains (Brockman et al., 2016) and has the same joint limits and physical parameters.

Many tasks can be solved by expressing a desired goal state or desired goal state components. For example, the 7-Dof reacher solves the task when the end effector XYZ component of its state is equal to the goal location, (x^*, y^*, z^*). One advantage of using a goal-conditioned model f as in Equation (5.8) is that this desire can be accounted for directly: if we already know the desired values of some components in s_{t+T}, then wen can simply fix those components of s_{t+T} and optimize over the other dimensions. For example for the 7-Dof reacher, the optimization problem in Equation (5.8) needed to choose an action becomes

$$a_t = \underset{a_t, s_{t+T}[0:14]}{\arg\max} \; r_c(f(s_t, a_t, s_{t+T}[0:14] || [x^*, y^*, z^*]))$$

where $||$ denotes concatenation; $s_{t+T}[0:14]$ denotes that we only optimize over the first 14 dimensions (the joint angles and velocities), and we omit a_{t+T} since the reward is only a function of the state. Intuitively, this optimization chooses whatever goal joint angles and joint velocities make it easiest to reach (x^*, y^*, z^*). It then chooses the corresponding action to get to that goal state in T time steps. We implement the optimization over $s[0:14]$ with stochastic optimization: sample 10,000 different vectors and choose the best value. Lastly, instead of optimizing over the actions, we use the policy trained in DDPG to choose the action, since the policy is already trained to choose an action with maximum Q-value for a given state, goal state, and planning horizon. We found this optimization scheme to be reliable, but any optimizer can be used to solve Equation (5.8),(5.7), or (5.6).

Pusher: The state consists of 3 joint angles, 3 joint angular velocities, the XY location of the hand, and the XY location of the puck. The action controls torques for each of the 3 joints. The reward function is the negative Euclidean distance between the puck and the puck. Once the hand is near (with 0.1) of the puck, the reward is increased by 2 minus the Euclidean distance between the puck and the goal location. This reward function encourages the arm to reach the puck. Once the arm reaches the puck, bonus reward begins to have affect, and the arm is encouraged to bring the puck to the target.

As in the 7-DoF reacher, we set components of the goal state for the optimal control formulation. Specifically, we set the goal hand position to be the puck location. To copy the two-stage reward shaping used by our baselines, the goal XY location for the puck is initially its current location until the hand reaches the puck, at which point the goal position for the puck is the target location. Since there are no other states to optimize over, the optimal control problem is trivial.

Half-Cheetah: The environment is the same as in Brockman et al. (2016). The only difference is that the reward is the ℓ-1 norm between the velocity and desired velocity v^*. Our optimal control formulation is again trivial since we set the goal velocity to be v^*. The goal velocity for rollout was sampled uniformly in the range $[-6, 6]$.

Ant: The environment is the same as in Brockman et al. (2016), except that we lowered the gear ratio to 30 for all joints. The reward is the ℓ-1 norm between the actual and desired xy-position and xy-velocity (for the position and velocity task) of the torso center of mass. For the target-position task, the target position was any position within a 6-by-6 square. For the target-position-and-velocity task, the target position was any position within a 1-by-1 square and any velocity within a 0.05-by-0.05 velocity-box. When computing the distance for the position-and-velocity task, the velocity distance was weighted by 0.9 and the position distance was weighted by 0.1.

Sawyer Robot: The state and action spaces are the same as in the 7-DoF simulated robot except that we also included the measured torques as part of the state space since these can different from the applied torques. The reward function used is also the ℓ-2 norm to the desired XYZ position.