Model-Based Policy Gradients

An empirical study on linear quadratic environments

Ângelo Gregório Lovatto

Thesis presented to the
Institute of Mathematics and Statistics
of the University of São Paulo
in partial fulfillment
of the requirements
for the degree of
Master of Science

Program: Ciência da Computação
Advisor: Profª. Drª. Leliane Nunes de Barros

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo
February 28, 2022

Model-Based Policy Gradients

An empirical study on linear quadratic environments

Ângelo Gregório Lovatto

This is the original version of the thesis prepared by candidate Ângelo Gregório Lovatto, as submitted to the Examining Committee.

Resumo

Ângelo Gregório Lovatto. **Gradientes de Política Baseados em Modelo: *Um estudo empírico em ambientes lineares quadráticos***. Dissertação (Mestrado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.

Métodos de Gradiente de Valor Estocástico (GVE) estão por trás de muitos avanços recentes de agentes de Aprendizado por Reforço (AR) baseado em modelo em espaços de estado-ação contínuos. Tais métodos usam dados coletados por exploração no ambiente para produzir um modelo de sua dinâmica, que é então usado para aproximar o gradiente, com relação aos parâmetros do agente, da função objetivo. Apesar da significância prática desses métodos, muitas escolhas de design algorítmico ainda carecem de rigorosas justificativas teóricas ou empíricas. Em vez disso, muitos trabalhos colocam muito peso em métodos de avaliação em ambientes-referência, o que mistura as contribuições de vários componentes do design de um agente de AR para o desempenho final. Este trabalho propõe uma análise refinada de componentes algorítmicos centrais a métodos de GVE, incluindo: a fórmula de estimação do gradiente, aprendizado do modelo e aproximação de função-valor. É implementado um ambiente-referência configurável baseado no regulador Linear Quadrático Gaussiano (LQG), permitindo computar o verdadeiro GVE e compará-lo com abordagens via aprendizado. Análises são conduzidas em uma variedade de ambientes LQG, avaliando o impacto de cada componente algorítmico em tarefas de predição e controle. Os resultados mostram que um estimador de gradiente amplamente usado induz um balanço de viés e variância favorável, usando uma esperança enviesada que produz estimativas de gradiente melhores com poucas amostras em comparação à fórmula não-enviesada do gradiente. Quanto ao aprendizado do modelo, demonstra-se que o modelo pode sobreajustar-se à dados *on-policy*, levando à predições acuradas de estados mas inacuradas de gradientes, salientando a importância da exploração até em ambientes estocásticos. É também mostrado que aproximação de função-valor pode ser mais instável que aprendizado de modelo, mesmo em simples ambientes lineares. Finalmente, avalia-se o desempenho ao usar o modelo para estimar o gradiente diretamente vs. para aproximar a função-valor, concluindo que a primeira abordagem é mais efetiva tanto para predição quanto para controle.

Palavras-chave: Aprendizado por Reforço. Baseado em Modelo. Métodos de Gradiente. Aprendizado de Máquina.

Abstract

Ângelo Gregório Lovatto. **Model-Based Policy Gradients:** ***An empirical study on linear quadratic environments***. Thesis (Master's). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2022.

Stochastic Value Gradient (SVG) methods underlie many recent achievements of model-based Reinforcement Learning (RL) agents in continuous state-action spaces. Such methods use data collected by exploration in the environment to produce a model of its dynamics, which is then used to approximate the gradient of the objective function w.r.t. the agent's parameters. Despite the practical significance of these methods, many algorithm design choices still lack rigorous theoretical or empirical justification. Instead, most works rely heavily on benchmark-centric evaluation methods, which confound the contributions of several components of an RL agent's design to the final performance. In this work, we propose a fine-grained analysis of core algorithmic components of SVGs, including: the gradient estimator formula, model learning and value function approximation. We implement a configurable benchmark environment based on the Linear Quadratic Gaussian (LQG) regulator, allowing us to compute the ground-truth SVG and compare it with learning approaches. We conduct our analysis on a range of LQG environments, evaluating the impact of each algorithmic component in prediction and control tasks. Our results show that a widely used gradient estimator induces a favorable bias-variance trade-off, using a biased expectation that yields better gradient estimates in smaller sample regimes than the unbiased expression for the gradient. On model learning, we show that overfitting to on-policy data may occur, leading to accurate state predictions but inaccurate gradients, highlighting the importance of exploration even in stochastic environments. We also show that value function approximation can be more unstable than model learning, even in simple linear environments. Finally, we evaluate performance when using the model for direct gradient estimation vs. for value function approximation, concluding that the former is more effective for both prediction and control.

Keywords: Reinforcement Learning. Model-Based. Gradient Methods. Machine Learning.

Acronyms

RL Reinforcement Learning vii, 1, 2, 3, 4, 5, 6, 9, 10, 11, 15, 16, 17, 21, 22, 24, 25, 26, 27, 28, 29, 47, 49, 50, 56, 61, 62

MBRL Model-Based Reinforcement Learning 2, 3, 4, 16, 47, 52

SVG Stochastic Value Gradient 2, 3, 4, 5, 6, 7, 9, 15, 16, 17, 18, 19, 21, 25, 26, 27, 29, 30, 31, 32, 33, 34, 36, 37, 39, 40, 41, 43, 44, 47, 48, 49, 50, 51, 52, 54, 55, 56, 58, 61, 63, 68

MAAC Model-Augmented Actor-Critic 3, 33, 34, 36, 37, 38, 39, 43, 44, 45, 46, 48, 54, 55, 56, 57, 58, 59, 62, 63, 69

LQG Linear Quadratic Gaussian 4, 5, 6, 9, 11, 12, 13, 14, 15, 19, 21, 24, 25, 26, 27, 28, 29, 30, 36, 37, 38, 39, 40, 41, 44, 45, 48, 56, 61, 62

DPG Deterministic Policy Gradient 6, 33, 34, 36, 37, 38, 39, 43, 45, 47, 50, 51, 52, 54, 55, 63

LQR Linear Quadratic Regulator 6

MDP Markov Decision Process 9, 10, 11, 27, 35

SGD Stochastic Gradient Descent 15, 37, 38, 41, 50, 54, 56, 57

MLE Maximum Likelihood Estimation 16, 44

KL Kullback Leibler 17, 44, 45, 46, 56, 57, 67

SCG Stochastic Computation Graph 17, 18, 26, 27, 32, 34

MSE Mean Squared Error 49, 50, 54

TD Temporal Difference 50, 51, 52, 53, 54, 55, 56, 63

MAGE Model-based Action-Gradient-Estimator 52, 53, 54, 55, 56, 57, 58, 59, 62, 63

DDPG Deep Deterministic Policy Gradients 56

Symbols

$\mathcal{M}$ Markov Decision Process 9, 10, 13, 14, 15, 21

$\mathcal{S}$ set of possible states 9, 10, 11, 13, 16, 21, 24, 30, 37, 44, 46, 55, 56, 67

$\mathcal{A}$ set of applicable actions 9, 10, 11, 13, 16, 21, 24, 30, 37, 44, 46, 55, 56, 67

H time horizon 9, 10, 11, 12, 14, 15, 21, 26, 31, 32, 38, 44, 46, 50, 56, 67

n state space dimension 9, 11, 12, 21, 22, 23, 24, 25, 44

d action space dimension 9, 12, 21, 25, 44

$\mathcal{P}$ Mapping from set to the set of all probability distributions over it 9, 10, 16

$\mathcal{T}$ set of decision timesteps 9, 10, 13, 14

$\mathbf{F}$ transition dynamics kernel 11, 12, 13, 14, 15, 21, 65

$\mathbf{f}$ transition dynamics bias 11, 12, 13, 14, 15, 21, 65

Σ covariance matrix for transition dynamics noise 11, 12, 13, 14, 15, 21, 65

$\mathbf{K}$ dynamic gain 13, 14, 15, 25, 26, 66

$\mathbf{k}$ static gain 13, 14, 15, 26, 66

m size of model parameter vector 16

$\mathbf{F}_\mathrm{s}$ passive transition dynamics kernel 21

$\mathbf{F}_\mathrm{a}$ active transition dynamics kernel 21

$\mathbf{w}$ transition dynamics noise random variable 65

List of Figures

List of Tables

List of Algorithms

Contents

Chapter 1

Introduction

1.1 Model-based Reinforcement Learning

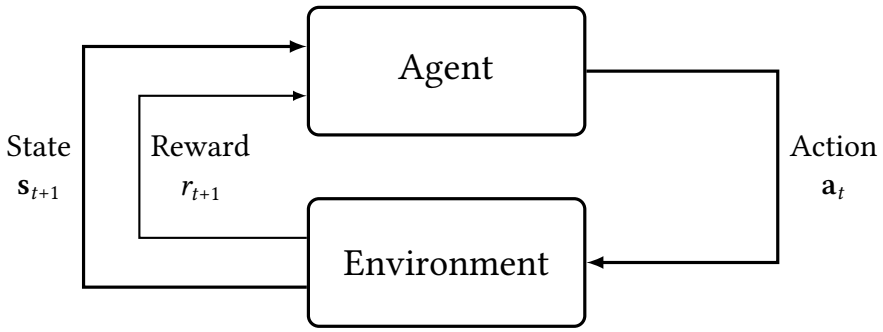

Figure 1.1: *The agent-environment interaction loop in RL*

RL is a framework for developing intelligent systems for sequential decision-making from limited data (Sutton and Barto, 2018; Szepesvári, 2010). It considers the setting in which an agent interacts with an environment in a series of discrete timesteps. Only three types of signals are exchanged between the agent and the environment: actions (sent by the agent to the environment), states and rewards (sent by the environment to the agent), as illustrated in fig. 1.1. States and rewards are random functions of the previous state and the agent's action at a given timestep. The agent's objective is to choose actions that maximize the expected sum of rewards (the *return*). More specifically, the solution to an RL problem is given in the form of a function mapping states to actions (a *policy*) to be executed at each timestep. The state comprises all of the necessary information for the agent to choose the best actions at any given timestep, i.e., the environment has the Markov property (François-Lavet *et al.*, 2018). This setting is general enough to model many sequential decision-making problems of interest.

What separates RL from other sequential decision-making frameworks is that the agent must not have access to the rules governing how new states (and sometimes rewards) are produced in the environment in response to the current state and action. Thus, RL can be used in many situations in which the environment is unknown (e.g., self-driving cars

and finance) or difficult to model (e.g., robotics and industrial control applications). On the other hand, this means that a learning algorithm has to find a policy by trial-and-error, a process that can be costly (e.g., in robotics) and very sample-inefficient. This issue is specially apparent in model-free methods, which try to find a policy without any model of the environment's dynamics.

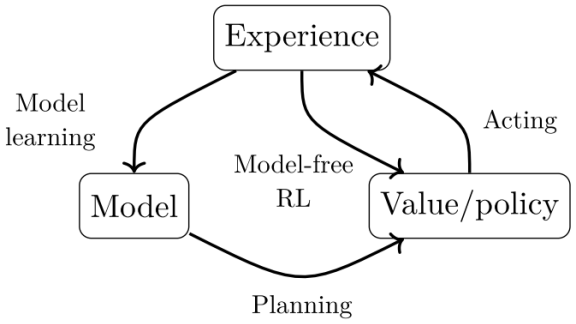

Figure 1.2: *General schema of the different methods for RL and the interaction between components in an RL agent (extracted from François-Lavet et al., 2018).*

Model-Based Reinforcement Learning (MBRL) (Polydoros and Nalpantidis, 2017; Moerland *et al.*, 2020) is a promising sub-field of RL for improving upon model-free methods. Unlike the latter, MBRL agents learn a predictive model of the environment to assist in finding a good policy. This predictive model is usually obtained via a supervised learning procedure, i.e., by fitting a function (from state and action to next state) to collected experiences (samples of state transitions). The agent can use the model to predict action outcomes, saving costly trial-and-error experimentation in the real world, or to estimate quantities useful for improving its policy. Figure 1.2 gives a high-level overview of the relationship between collected data (experiences), the model and the policy in a MBRL algorithm. In the next section we discuss a specific class of model-based methods which will be focus of this thesis.

1.2 Stochastic Value Gradient methods

Stochastic Value Gradient (SVG) methods try to find a good policy defined by a set of parameters, i.e., a function mapping the current state and a set of parameters to the action for execution. We call this a parameterized (or parametric) policy. We then reframe the RL problem as searching for the parameters that maximize the expected return of the induced policy. SVG methods use the model to estimate the value gradient: the gradient of the expected return w.r.t. the policy's parameters. With a gradient estimate in hand, we can leverage stochastic optimization methods to update our policy's parameters iteratively. We call this general process *policy optimization*, analogous to how generic parametric functions are updated with gradient-based methods.

Model-free methods can also estimate the value gradient through sampling by making use of a statistical tool called the *score-function* estimator. SVG methods, on the other hand, can leverage the model to produce gradients via the *pathwise derivative estimator*, usually found to be more stable in practice (Schulman, Heess, *et al.*, 2015a). One way to achieve this is to simulate future experiences using the model as proxy for the environment, as

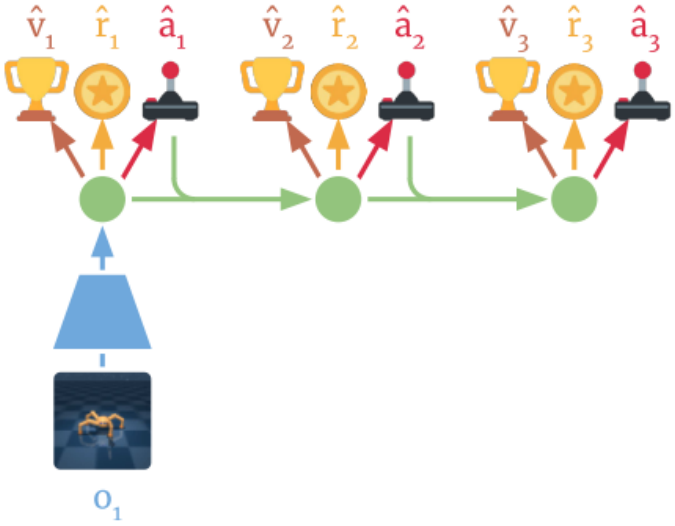

Figure 1.3: *Dreamer (Hafner et al., 2020) as an instance of an SVG method. The model is first used to condense the state representation (in this case, an image of the spider-like robot being controlled), denoted as the blue part mapping the observation to the latent state in green. Then, the model simulates experiences with the current policy (denoted by the joystick), yielding future states, rewards (stars) and values (trophies; more on values in chapter 2). The reward gradients are propagated through the model to the policy parameters.*

illustrated by fig. 1.3, then computing the gradient of future rewards w.r.t. policy parameters. One drawback, however, is that the pathwise derivative estimator limits SVG methods to problems where actions are continuous, as one can't define a differentiable policy with discrete outputs. These are called *continous control* problems. Moreover, this estimator requires not only the outputs (next states) predicted by our model, but also the gradients w.r.t. the inputs (states and actions). As we'll see in chapter 5, gradients can be harder to learn from data.

A variety of recent MBRL algorithms for continuous control have used the SVG approach. Deisenroth and Rasmussen (2011) introduce the PILCO algorithm, one of the first to leverage a learned model's derivatives to compute the value gradient with few samples, but its use of Gaussian processes hinders scalability to larger problems. The original SVG paper by Heess *et al.* (2015) introduced gradient estimation with stochastic neural network models using the reparameterization trick, an approach scalable to higher-dimensional problems. Dreamer and Imagined Value Gradients explore SVGs with latent-space models (Hafner *et al.*, 2020; Byravan *et al.*, 2019). Model-Augmented Actor-Critic (MAAC) and SAC-SVG extend the SVG framework to that of maximum-entropy RL to incentivize exploration and stabilize optimization (Clavera *et al.*, 2020; Amos, Stanton, *et al.*, 2020).

Recently proposed RL agents using the SVG approach have demonstrated its effectiveness in learning robotic locomotion from data with unprecedented sample-efficiency (Amos, Stanton, *et al.*, 2020; Clavera *et al.*, 2020; Hafner *et al.*, 2020). Thus, the SVG class of algorithms is a promising candidate for addressing one of the main issues in RL. This potential comes with it owns costs, however, as we'll discuss in the next section, which motivates the investigations done in our work.

1.3 When theory doesn't meet practice

Reliability and reproducibility concerns regarding modern RL methods have been raised by several works (Chan *et al.*, 2020; Henderson *et al.*, 2018; Islam *et al.*, 2017). Unfortunately, we don't fully understand why these algorithms fail. Recent work has shed light on the inner workings of several model-free algorithms, revealing that code-level optimizations are often the deciding factor in an algorithm's performance (Engstrom *et al.*, 2020; Liu *et al.*, 2021). In contrast, papers proposing new algorithms usually base their contributions, e.g., which objective function to differentiate, on best-case scenarios free from sampling and estimation errors. Performance, however, is usually determined by other factors. For instance, Ilyas *et al.* (2020) showed that Policy Gradient algorithms fail to produce good estimates of the policy gradient, a core tenet of policy gradient theory, even in situations which the algorithm does find a good policy. In this work, we aim to extend this type of analysis to model-based algorithms, specifically SVG methods.

Model-based algorithms present even more challenges to rigorous analysis due to their higher number of moving parts. For instance, model-free algorithms may have two interdependent learning components, the policy and value function (see section 2.1). On the other hand, model-based algorithms may introduce two more, the state dynamics and reward models, each with their own learning subroutines, that can be used in optimizing the policy and value function. This complexity makes it difficult to design new model-based algorithms, as there are several points of failure. As such, theoretically promising MBRL algorithms may fail (Ângelo G. Lovatto *et al.*, 2020) without leaving a clue as to what caused the negative result. Moreover, negative results are not often publicized, and positive ones usually only focus on the overall performance of the learned policy, not showing how each component has contributed to such result.

Our work aims to inspect the inner workings of SVG algorithms and offer clues about the most important components in the learning process. To the best of our knowledge, the work by Ilyas *et al.* (2020) is the closest to ours, since it also proposes a fine-grained analysis of RL algorithms that try to estimate the value gradient. The key differences between our work and theirs are as follows: (a) we consider model-based, instead of model-free, methods for policy optimization and (b) we take a step back from complex, sophisticated simulators and instead use the simpler but general Linear Quadratic Gaussian (LQG) as our RL environment. Point (b) is crucial, as it allows us to evaluate our algorithms against their real ultimate objective. E.g., LQG allows us to compute the exact value gradient and compare it to the estimated gradient using learned models. In contrast, Ilyas *et al.* (2020) rely on sample-based approximations of the true gradient, due to their choice of complex, non-linear environments. However, sample-based estimates may not converge to the true gradient depending on the choice of estimator function, as we show on chapter 4. Thus, our approach provides a more reliable setting to do a fine-grained analysis of gradient-based algorithms. In the next section, we describe the LQG in more detail and its usefulness as a test bed for RL methods in general.

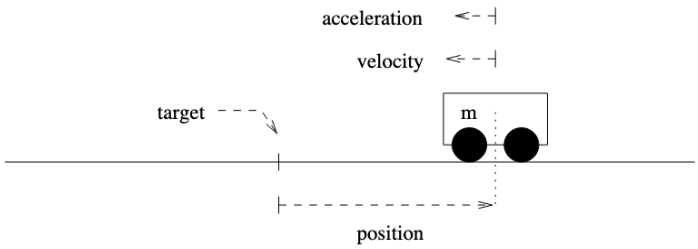

Figure 1.4: *Double Integrator environment by Santamaría et al. (1997). The agent must learn to maneuver the car to the target position as quick as possible while expending the least energy.*

1.4 Linear Quadratic Gaussian environments

The LQG framework is extensively studied in the Optimal Control literature (Richard B Vinter and R. Vinter, 2010; Emanuel Todorov, 2006). It considers a special class of continuous control problems that may be treated as RL environments. LQG is simple because: (a) its dynamics is a Gaussian distribution, with a linear function of states and actions as its average, and (b) its reward is a quadratic convex function of states and actions. Figure 1.4 illustrates a simple example of such a system: the double integrator. The state is a vector consisting of the current position and velocity of the car. The action is a single scalar representing the acceleration applied to it. By fixing a certain time interval between interactions, the dynamics are represented as a linear function of states and actions. The reward function penalizes both the distance to the target position and large control inputs, i.e., strong accelerations. Thus, the agent must learn to maneuver the car to the target position as quickly as possible while expending the least energy. See example 2.2.1 for mathematical details.

LQGs are often used as a discretization of continous-time dynamics described as linear differential equations, such as those of physical systems. Furthermore, solving LQGs is a necessary part of many optimal control methods for non-linear environments, such as the iLQG algorithm by Todorov and Weiwei Li (2005). We discuss why LQG environments are interesting for this work in what follows.

For our purposes, the main advantage of LQGs against other types of environments is that we're able to compute the ground-truth performance of a given policy. This is thanks to the constraints on the form of dynamics and reward functions and years of research by the Optimal Control community in developing analytical solutions for this problem class. As we show in chapter 3, this also gives us access to the ground-truth value gradient by combining these solutions with backpropagation algorithms perfected by Deep Learning (I. Goodfellow *et al.*, 2016) frameworks. Other more complicated, nonlinear benchmarks for continuous control don't give us this ability (Emanuel Todorov *et al.*, 2012). Thus, in LQGs, researchers can assess if SVG algorithms are estimating both the value and its gradient correctly, allowing for a better understanding of which algorithmic components are failing.

Moreover, different LQGs can be cheaply generated to simulate different scenarios, as we show in chapter 3. In contrast, a lot of RL research builds upon the same benchmarks, which can lead to algorithmic biases that causes methods to fail when applied to new

environments (Hessel *et al.*, 2019). We develop an open-source library in Python for randomly generating LQG environments and make it available to researchers.

Furthermore, LQG environments are non-trivial. Recall that RL methods, unlike in Optimal Control, don't have access to the dynamics and reward function, having instead to find a policy by trial-and-error. Recht (2019) proposed LQGs as a simple, yet nontrivial, class of environments to help evaluate RL methods after showing that it can present a challenge to model-free policy gradient methods. [1] Tsiamis and Pappas (2021) showed that it can be very challenging (i.e., requiring a large amount of data) to learn a dynamics model from limited interactions with an LQG environment. Thus, we believe LQG to be the ideal framework for both comparing different SVG methods and providing a better look into their inner workings.

1.5 Contributions of this thesis

This thesis develops methods for better understanding the core tenets behind SVG methods and offers explanations, via empirical experiments, for commonly-found patterns in the SVG literature.

We develop a benchmark for evaluation of gradient quality metrics and policy optimization performance based on the LQG framework. Chapter 3 describes the key features of our benchmark, namely: (a) random generation of LQG environments; (b) derivation of optimal policies for LQG environments; (c) ground-truth value for a given policy and environment; and (d) ground-truth value gradient for a given policy and environment. While we use these tools to inspect SVG algorithms, they are general enough to be useful to researchers evaluating other RL approaches. Thus, we make the code for this benchmark open-source and available online. [2]

We use our benchmark to analyze important algorithmic components of SVG methods. Chapter 4 analyzes the gradient estimator formula, disregarding model approximation error. We consider different estimators' effectiveness in approximating the true value gradient for fixed policies, as well as their impact on overall policy optimization. The experimental results indicate that a bias-variance trade-off occurs between estimators, favoring algorithms that use a slightly biased one that produces better gradients with fewer samples. Moreover, the choice of estimator has a significant impact on the gradient magnitude and thus the learning rate in policy optimization. This work was previously published by Ângelo Gregório Lovatto *et al.* (2021).

Chapter 5 analyzes the model learning subroutine of SVG methods. We consider how effective model-learning is in finding models that induce a good estimate of the value gradient for fixed policies. We compare it with the main model-free alternative for estimating the value gradient: Q-learning combined with the Deterministic Policy Gradient (DPG) estimator (Silver *et al.*, 2014). Our experiments show that the data-collection procedure has a significant impact on the quality of the estimated gradient in model-based estimation.

[1] To be precise, the environment used there was an Linear Quadratic Regulator (LQR), an instance of LQG with deterministic dynamics.

[2] https://github.com/angelolovatto/LQSVG.

On the other hand, model-free estimation is more consistent, but shows poorer results. We corroborate the results by D'Oro and Jaskowski (2020) and show that differentiable models can be used for improving Q-learning in the gradient space, addressing some of its shortcomings. We then contrast these findings to the statements of Amos, Stanton, *et al.* (2020) and show how the model may be used both for policy improvement and critic updates without a higher risk of failure. The chapter finishes with experiments comparing different model-based strategies for SVG methods in the overall policy optimization procedure. These experiments show that model-learning performance is mostly independent from the approach used for Q-function learning or policy improvement. On the other hand, Q-function approximation is dependent on the policy improvement step. Overall, the results suggest that the environment model is better used in the policy improvement step (value gradient estimation) and can be used to enhance Q-function learning using the approach by D'Oro and Jaskowski (2020). However, using the model exclusively for the latter while relying on a model-free gradient estimator leads to poor results, with the target policy diverging from the optimal one.

Chapter 2

Background

This chapter outlines a succinct theoretical foundation necessary for understanding the contributions of this thesis. The following sections start from the general down to the more specific. Section 2.1 gives a general background of Reinforcement Learning in continuous control with limited horizons. Section 2.2 introduces a subclass of continuous RL environments, the Linear Quadratic Gaussian, and its analytical solutions. Finally section 2.3 dives into the realm of Stochastic Value Gradient methods, a subset of RL algorithms for continuous control, and outlines their special components.

2.1 Reinforcement Learning

We consider the agent-environment interaction modeled as a continuous Markov Decision Process (MDP), defined as follows.

Definition 2.1.1: Continuous MDP

A continous MDP is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, H, R, p^*, \rho)$*, where:*

1. $\mathcal{S} \subseteq \mathbb{R}^n$ *is the space of possible states;*
2. $\mathcal{A} \subseteq \mathbb{R}^d$ *is the space of applicable actions;*
3. $H \in \mathbb{N}$ *is the time horizon*
4. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ *is the reward function;*
5. $p^* : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{P}(\mathcal{S})$ *is the transition probability kernel; and*
6. $\rho \in \mathcal{P}(\mathcal{S})$ *is the probability distribution of the initial state.*

Here, $\mathcal{P}(\mathcal{S})$ *denotes the set of all probability distributions defined over* $\mathcal{S}$ *(A.-m. Farahmand, 2018) .*

Definition 2.1.1 differs from usual definitions of MDPs (Szepesvári, 2010) in that it considers continuous state-action spaces and a limited time horizon for interaction with the environment. That is, interaction occurs in *episodes* with finite discrete timesteps $t \in \mathcal{T} = \{0, \dots, H-1\}$. The initial state is sampled from the initial state distribution, $\mathbf{s}_0 \sim \rho$. At every timestep t, the agent observes the current state $\mathbf{s}_t \in \mathcal{S}$ from the set of possible states of the environment. It must then select an action $\mathbf{a}_t \in \mathcal{A}$ from the set of possible actions to execute. The environment then transitions to the next state by sampling

from the transition probability kernel, $\mathbf{s}_{t+1} \sim p^*(\mathbf{s}_t, \mathbf{a}_t)$, and emits a reward signal using its reward function, $r_{t+1} = R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$. We overload notation to let $p^*(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ denote the probability density of $\mathbf{s}_{t+1}$ conditioned on $\mathbf{s}_t$ and $\mathbf{a}_t$.

Definition 2.1.2: Policies

Given $\mathcal{S}$, $\mathcal{A}$, and H from an MDP, a deterministic policy is a function $\mu : \mathcal{S} \times \mathcal{T} \mapsto \mathcal{A}$, while a stochastic policy is a function $\pi : \mathcal{S} \times \mathcal{T} \mapsto \mathcal{P}(\mathcal{A})$

Since we're considering time-limited episodes, definition 2.1.2 includes a timestep argument for policies (Pardo *et al.*, 2018). From this point onward, we shall use *policy* to refer to the deterministic type in definition 2.1.2, unless explicitly stated otherwise. A policy defines how the agent selects actions at each timestep, i.e., $\mathbf{a}_t = \mu(\mathbf{s}_t, t)$. It also implicitly defines the *on-policy dynamics* $p^\mu : \mathcal{S} \times \mathcal{T} \mapsto \mathcal{P}(\mathcal{S})$, defined

$$p^\mu(\mathbf{s}, t) \doteq p^*(\mathbf{s}, \mu(\mathbf{s}, t)) \,. \tag{2.1}$$

We define a policy's *performance* or *value* as its expected cumulative reward, or *return*, from the initial state:

$$J(\mu) = \mathbb{E}_\mu \left[\sum_{t=0}^{H-1} R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \right] , \tag{2.2}$$

where the expectation is implicitly w.r.t. the initial state distribution ($\mathbf{s}_0 \sim \rho$) and the sequential application of $\mathbf{s}_{t+1} \sim p^\mu(\mathbf{s}_t, t)$. We can break down the value of a policy in different states by using its *value-functions*.

Definition 2.1.3: Value functions

Given an MDP $\mathcal{M}$ and a policy μ, let $\mathcal{T}^+ = \mathcal{T} \cup \{H\}$. The on-policy value functions are $V^\mu : \mathcal{S} \times \mathcal{T}^+ \mapsto \mathbb{R}$ (state-value),

$$V^\mu(\mathbf{s}, t) \doteq \mathbb{E}_\mu \left[\sum_{k=t}^{H-1} R(\mathbf{s}_k, \mathbf{a}_k, \mathbf{s}_{k+1}) \,\middle|\, \mathbf{s}_t = \mathbf{s} \right] \tag{2.3}$$

and $Q^\mu : \mathcal{S} \times \mathcal{A} \times \mathcal{T} \mapsto \mathbb{R}$ (action-value),

$$Q^\mu(\mathbf{s}, \mathbf{a}, t) \doteq \mathbb{E}_{\mathbf{s}' \sim p^*(\mathbf{s},\mathbf{a})} \left[R(\mathbf{s}, \mathbf{a}, \mathbf{s}') + V^\mu(\mathbf{s}', t+1) \right] . \tag{2.4}$$

The optimal value functions $V^\star : \mathcal{S} \times \mathcal{T}^+ \mapsto \mathbb{R}$ and $Q^\star : \mathcal{S} \times \mathcal{A} \times \mathcal{T} \mapsto \mathbb{R}$ capture the maximum expected return attained by any policy:

$$V^\star(\mathbf{s}, t) \doteq \max_\mu V^\mu(\mathbf{s}, t) \,, \tag{2.5}$$

$$Q^\star(\mathbf{s}, \mathbf{a}, t) \doteq \mathbb{E}_{\mathbf{s}' \sim p^*(\mathbf{s},\mathbf{a})} \left[R(\mathbf{s}, \mathbf{a}, \mathbf{s}') + V^\star(\mathbf{s}', t+1) \right] . \tag{2.6}$$

Definition 2.1.3 differs slightly from usual definitions in RL in that it defines time-dependent functions, a consequence of the finite horizon. Value functions obey special

recurrence relations known as the *Bellman equations*:

$$V^{\mu}(\mathbf{s}) = \mathbb{E}_{\mathbf{s}' \sim p^*(\mathbf{s},\mu(\mathbf{s}))}\left[R(\mathbf{s},\mu(\mathbf{s}),\mathbf{s}') + V^{\mu}(\mathbf{s}')\right], \tag{2.7}$$

$$Q^{\mu}(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\mathbf{s}' \sim p^*(\mathbf{s},\mathbf{a})}\left[R(\mathbf{s},\mathbf{a},\mathbf{s}') + Q^{\mu}(\mathbf{s}',\mu(\mathbf{s}'))\right]. \tag{2.8}$$

Value functions are useful for comparing policies in different situations and, as we'll see later, play an essential role in many RL algorithms. Solving for the on-policy value functions is known as the *prediction problem* in RL.

The RL objective is to find a policy with the highest value. We can define it as searching for the policy that maximizes the expected value of the initial state:

$$\text{maximize}_{\mu} J(\mu) \equiv \text{maximize}_{\mu}\, \mathbb{E}_{\mathbf{s}\sim\rho}[V^{\mu}(\mathbf{s},0)]\,. \tag{2.9}$$

Solving for the optimal policy w.r.t. the objective above is known as the *control problem* in RL. The key difference between Optimal Control and RL, both frameworks for optimal sequential decision making, is that in the former the agent has access to the full MDP, while in the latter the agent only knows $\mathcal{S}, \mathcal{A}$, and H and has to learn its policy by trial-and-error in the environment. As a consequence, stochastic policies are useful for *exploring* the environment to find actions yielding a high return. Nevertheless, in an MDP, it is guaranteed to exist a deterministic policy that achieves the optimal return (Szepesvári, 2010).

2.2 Linear Quadratic Gaussian regulator

In this section we describe the main theoretical foundations necessary for working with LQGs. We show how to interpret the LQG as an MDP and how the control and prediction problems can be solved in closed form.

2.2.1 Problem statement

The LQG is a special class of continuous MDP in which the transition kernel is linear Gaussian and the reward function is quadratic concave (Emanuel Todorov, 2006).

Definition 2.2.1: LQG MDP

An LQG is a continuous MDP $(\mathcal{S}, \mathcal{A}, H, R, p^*, \rho)$ *with the following components. The initial state distribution is Gaussian,*

$$\rho = \mathcal{N}(\boldsymbol{\mu}_{\rho}, \Sigma_{\rho})\,, \tag{2.10}$$

with $\boldsymbol{\mu}_{\rho} \in \mathbb{R}^{n\times n}$ *and* $\Sigma_{\rho} \in \mathbb{R}^{n\times n}$ *symmetric positive definite. The transition probability kernel is linear Gaussian,*

$$p^*(\mathbf{s},\mathbf{a},t) = \mathcal{N}\left(\mathbf{F}_t \begin{bmatrix}\mathbf{s}\\ \mathbf{a}\end{bmatrix} + \mathbf{f}_t, \Sigma_t\right), \tag{2.11}$$

with $\mathbf{F}_t \in \mathbb{R}^{n\times(n+d)}, \mathbf{f}_t \in \mathbb{R}^n$, *and* $\mathbf{\Sigma}_t \in \mathbb{R}^{n\times n}$ *symmetric positive definite. The reward function is quadratic concave,*

$$R(\mathbf{s}, \mathbf{a}, \mathbf{s}', t) = -\left(\frac{1}{2} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix}^\mathsf{T} \mathbf{C}_t \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} + \mathbf{c}_t^\mathsf{T} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} + \mathbb{1}_{\{H\}}(t) R_f(\mathbf{s}') \right), \tag{2.12}$$

where R_f *defines the reward at the final state,*

$$R_f(\mathbf{s}) = -\left(\frac{1}{2} \mathbf{s}'^\mathsf{T} \mathbf{C}_f \mathbf{s}' + \mathbf{c}_f^\mathsf{T} \mathbf{s}' \right), \tag{2.13}$$

with $\mathbf{C}_t \in \mathbb{R}^{(n+d)\times(n+d)}$ *and* $\mathbf{C}_f \in \mathbb{R}^{n\times n}$ *symmetric positive semi-definite,* $\mathbf{c}_t \in \mathbb{R}^{n+d}$, *and* $\mathbf{c}_f \in \mathbb{R}^n$. *We assume* $\mathbf{C}_{\mathbf{aa}\,t}$, *the block of* $\mathbf{C}_t$ *multiplying the left and right action vectors in eq.* (2.12), *is positive definite.*

Note that definition 2.2.1 uses a *time-dependent* transition kernel and reward function, in contrast with the *stationary* ones in definition 2.1.1. This means that the next-state distribution and reward function may change depending on the current timestep of the episode. We chose to list the timestep as an additional argument to each function for clarity. Alternatively, we could have absorved the timestep variable into the state representation (the approach we use in the implementation). The indicator function $\mathbb{1}_{\{H\}}(t)$ is 1 if $t \in \{H\}$ and 0 otherwise.

Also note that the rewards in definition 2.2.1 have a negative sign. This is inherited from the original definition of LQGs in Optimal Control, which considers the problem of minimizing cumulative costs instead of maximizing return. Furthermore, there is always a cost associated with a non-zero action, since $\mathbf{C}_{\mathbf{aa}}$ is positive definite.

Example 2.2.1: Double integrator

The double integrator illustrated by fig. 1.4 can be modeled as an LQG as follows. The state is a 2-dimensional vector containing the current position, p, *and velocity,* v, *of the car, or* $\mathbf{s} = [p\ v]^\mathsf{T}$ *in vectorial representation. The action is a scalar representing the acceleration,* a, *applied to the car, or* $\mathbf{a} = [a]$ *in vectorial notation. The time horizon may be arbitrarily set by the task designer, e.g.,* $H = 100$. *Without loss of generality, the target position is the origin:* $\mathbf{s}_g = [0\ 0]^\mathsf{T}$. *The reward function penalizes the distance to the target position and the magnitude of the acceleration. This can be represented by a quadratic function of state and action as*

$$R(\mathbf{s}, \mathbf{a}, \mathbf{s}', t) = -\frac{1}{2} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix}^\mathsf{T} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} - \mathbb{1}_{\{H\}}(t) \frac{1}{2} \mathbf{s}'^\mathsf{T} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \mathbf{s}' .$$

The next state given the current state and action is computed as

$$\mathbf{s}' = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} \Delta_t ,$$

where Δ_t is a predefined interval between timesteps, e.g., $\Delta_t = 0.05$. The deterministic dynamics above can be framed as a degenerate linear Gaussian transition probability kernel with $\Sigma_t = \mathbf{0}$. The covariance matrix may be nonzero if the task designer wishes to model exogenous random influences in the car's movement, e.g., wind.

2.2.2 Solutions by dynamic programming

In this section we show how to solve the prediction and control problem analytically in the LQG. We assume, without loss of generality, time-invariant (stationary) dynamics, i.e., $\mathbf{F}_t = \mathbf{F}, \mathbf{f}_t = \mathbf{f}, \Sigma_t = \Sigma$ for $t \in \mathcal{T}$, and time-invariant rewards, i.e., $\mathbf{C}_t = \mathbf{C}, \mathbf{c}_t = \mathbf{c}$ for $t \in \mathcal{T}$. Note that all the following results apply to time-dependent dynamics and rewards by substituting $\mathbf{F}_t, \mathbf{f}_t, \Sigma_t, \mathbf{C}_t, \mathbf{c}_t$ for $\mathbf{F}, \mathbf{f}, \Sigma, \mathbf{C}, \mathbf{c}$ respectively at the appropriate steps.

Recall that the control problem involves searching for a policy of maximum value. In LQGs, we consider a special class of policies, namely *time-varying linear policies*.

Definition 2.2.2: Time-varying linear policy

A time-varying linear policy is a function $\mu_\theta : \mathcal{S} \times \mathcal{T} \mapsto \mathcal{A}$ such that

$$\mu_\theta(\mathbf{s}, t) \doteq \mathbf{K}_t \mathbf{s} + \mathbf{k}_t \tag{2.14}$$

where $\mathbf{K}_t \in \mathbb{R}^{d \times n}$, $\mathbf{k}_t \in \mathbb{R}^d$ and $\theta = \{\mathbf{K}_t, \mathbf{k}_t\}_{t \in \mathcal{T}}$.

From definition 2.2.2, we can fully define a policy μ_θ by its collection of *dynamic gains* $\mathbf{K}_t$ and *static gains* $\mathbf{k}_t$. Thus, we frame the control problem as searching for the parameters θ that induce the policy with the highest value. The following lemma states that the optimal policy for an LQG is time-varying linear.

Lemma 2.2.1: Optimal LQG policy

The optimal policy in an LQG $\mathcal{M}$ is time-varying linear with dynamic and static gains $\{(\mathbf{K}_t, \mathbf{k}_t)\}_{t \in \mathcal{T}}$, where

$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{aa}_t}^{-1} \mathbf{Q}_{\mathbf{as}_t}, \quad \mathbf{k}_t = -\mathbf{Q}_{\mathbf{aa}_t}^{-1} \mathbf{q}_{\mathbf{a}_t}, \quad t \in \mathcal{T}\,. \tag{2.15}$$

The policy gains are derived from the optimal value functions, with quadratic forms:

$$Q^\star(\mathbf{s}, \mathbf{a}, t) = -\tfrac{1}{2} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix}^\mathsf{T} \mathbf{Q}_t \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} + \mathbf{q}_t^\mathsf{T} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} + q_t\,, \quad t \in \mathcal{T}\,, \tag{2.16}$$

where

$$\mathbf{Q}_t = \mathbf{C} + \mathbf{F}^\mathsf{T} \mathbf{V}_{t+1} \mathbf{F}, \tag{2.16a}$$

$$\mathbf{q}_t = \mathbf{F}^\mathsf{T} \mathbf{V}_{t+1} \mathbf{f} + \mathbf{F}^\mathsf{T} \mathbf{v}_{t+1} + \mathbf{c}, \tag{2.16b}$$

$$q_t = \tfrac{1}{2} \operatorname{Tr}(\Sigma \mathbf{V}_{t+1}) + \tfrac{1}{2} \mathbf{f}^\mathsf{T} \mathbf{V}_{t+1} \mathbf{f} + \mathbf{v}_{t+1}^\mathsf{T} \mathbf{f} + v_{t+1}\,. \tag{2.16c}$$

and

$$V^\star(\mathbf{s}, t) = -\tfrac{1}{2}\mathbf{s}^\intercal \mathbf{V}_t \mathbf{s} + \mathbf{v}_t^\intercal \mathbf{s} + \upsilon_t, \quad t \in \mathcal{T}^+, \tag{2.17}$$

where

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{ss}_t} + \mathbf{Q}_{\mathbf{sa}_t}\mathbf{K}_t + \mathbf{K}_t^\intercal \mathbf{Q}_{\mathbf{aa}_t}\mathbf{K}_t + \mathbf{K}_t^\intercal \mathbf{Q}_{\mathbf{as}_t}, \tag{2.17a}$$

$$\mathbf{v}_t = \mathbf{Q}_{\mathbf{sa}_t}\mathbf{k}_t + \mathbf{K}_t^\intercal \mathbf{Q}_{\mathbf{aa}_t}\mathbf{k}_t + \mathbf{q}_{\mathbf{s}_t} + \mathbf{K}_t^\intercal \mathbf{q}_{\mathbf{a}_t}, \tag{2.17b}$$

$$\upsilon_t = \tfrac{1}{2}\mathbf{k}_t^\intercal \mathbf{Q}_{\mathbf{aa}_t}\mathbf{k}_t + \mathbf{q}_{\mathbf{a}_t}^\intercal \mathbf{k}_t + q_t, \tag{2.17c}$$

$$\textit{and} \;\; \mathbf{V}_N = \mathbf{C}_f, \mathbf{v}_N = \mathbf{c}_f, \upsilon_N = 0. \tag{2.17d}$$

A proof of lemma 2.2.1 is in appendix A. The lemma naturally leads to an iterative, *dynamic programming* procedure for computing the optimal policy, described in algorithm 1. The algorithm computes the value function at termination trivially from the reward of terminal states given by definition 2.2.1. It then uses the recurrence relations from lemma 2.2.1 to compute the value functions and policy for each timestep, going backwards in time from the final decision timestep. Algorithm 1 solves the control problem.

We can adapt algorithm 1 to compute the value functions of an arbitrary policy, resulting in algorithm 2. Note that the only changes are: 1) the algorithm receives both an LQG and a policy as input; and 2) lines 6 and 7 of algorithm 1 are discarded. It is important to note that the value function coefficients may not be optimal now, since we're given an arbitrary policy (notice the updated comments in the pseudocode). Algorithm 2 solves the prediction problem.

Algorithm 1: LQG control

Input: LQG $\mathcal{M}$

Output: Optimal policy and value functions for $\mathcal{M}$

1 $\mathbf{V}_H, \mathbf{v}_H, \upsilon_H \leftarrow \mathbf{C}_f, \mathbf{c}_f, 0$

2 **for** $t = H-1, \dots, 0$ **do**

// Compute $Q^\star(\mathbf{s}, \mathbf{a}, t) = \mathbb{E}_{\mathbf{s}'}[R(\mathbf{s}, \mathbf{a}, \mathbf{s}') + V^\star(\mathbf{s}', t+1)]$

3 $\mathbf{Q}_t \leftarrow \mathbf{C}_t + \mathbf{F}_t^\intercal \mathbf{V}_{t+1}\mathbf{F}_t$

4 $\mathbf{q}_t \leftarrow \mathbf{c}_t + \mathbf{F}_t^\intercal \mathbf{V}_{t+1}\mathbf{f}_t + \mathbf{F}_t^\intercal \mathbf{v}_{t+1}$

5 $q_t \leftarrow \frac{1}{2}\operatorname{Tr}(\Sigma_t \mathbf{V}_{t+1}) + \frac{1}{2}\mathbf{f}_t^\intercal \mathbf{V}_{t+1}\mathbf{f}_t + \mathbf{v}_{t+1}^\intercal \mathbf{f}_t + \upsilon_{t+1}$

// Solve $\mu_\theta^\star(\mathbf{s}_t) = \arg\max_{\mathbf{a}} Q^\star(\mathbf{s}_t, \mathbf{a})$

6 $\mathbf{K}_t \leftarrow -\mathbf{Q}_{\mathbf{aa}_t}^{-1}\mathbf{Q}_{\mathbf{as}_t}$

7 $\mathbf{k}_t \leftarrow -\mathbf{Q}_{\mathbf{aa}_t}^{-1}\mathbf{q}_{\mathbf{a}_t}$

// Compute $V^\star(\mathbf{s}, t) = Q^\star(\mathbf{s}, \mu_\theta^\star(\mathbf{s}, t), t)$

8 $\mathbf{V}_t \leftarrow \mathbf{Q}_{\mathbf{ss}_t} + \mathbf{Q}_{\mathbf{sa}_t}\mathbf{K}_t + \mathbf{K}_t^\intercal \mathbf{Q}_{\mathbf{aa}_t}\mathbf{K}_t + \mathbf{K}_t^\intercal \mathbf{Q}_{\mathbf{as}_t}$

9 $\mathbf{v}_t \leftarrow \mathbf{Q}_{\mathbf{sa}_t}\mathbf{k}_t + \mathbf{K}_t^\intercal \mathbf{Q}_{\mathbf{aa}_t}\mathbf{k}_t + \mathbf{q}_{\mathbf{s}_t} + \mathbf{K}_t^\intercal \mathbf{q}_{\mathbf{a}_t}$

10 $\upsilon_t \leftarrow \frac{1}{2}\mathbf{k}_t^\intercal \mathbf{Q}_{\mathbf{aa}_t}\mathbf{k}_t + \mathbf{q}_{\mathbf{a}_t}^\intercal \mathbf{k}_t + q_t$

11 **return** $\mu_\theta^\star, V^\star, Q^\star$

Algorithm 2: LQG Prediction

Input: LQG $\mathcal{M}$, time-varying linear policy μ_θ
Output: The value functions for μ_θ

$\mathbf{V}_H, \mathbf{v}_H, v_H \leftarrow \mathbf{C}_f, \mathbf{c}_f, 0$
for $t = H-1, \dots, 0$ **do**
  // Compute $Q^{\mu_\theta}(\mathbf{s}, \mathbf{a}, t) = \mathbb{E}_{\mathbf{s}'}[R(\mathbf{s}, \mathbf{a}, \mathbf{s}') + V^{\mu_\theta}(\mathbf{s}', t+1)]$
  $\mathbf{Q}_t \leftarrow \mathbf{C}_t + \mathbf{F}_t^\intercal \mathbf{V}_{t+1} \mathbf{F}_t$
  $\mathbf{q}_t \leftarrow \mathbf{c}_t + \mathbf{F}_t^\intercal \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^\intercal \mathbf{v}_{t+1}$
  $q_t \leftarrow \frac{1}{2} \mathrm{Tr}(\Sigma_t \mathbf{V}_{t+1}) + \frac{1}{2} \mathbf{f}_t^\intercal \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{v}_{t+1}^\intercal \mathbf{f}_t + v_{t+1}$
  // Compute $V^{\mu_\theta}(\mathbf{s}, t) = Q^{\mu_\theta}(\mathbf{s}, \mu_\theta(\mathbf{s}, t), t)$
  $\mathbf{V}_t \leftarrow \mathbf{Q}_{\mathbf{ss}_t} + \mathbf{Q}_{\mathbf{sa}_t} \mathbf{K}_t + \mathbf{K}_t^\intercal \mathbf{Q}_{\mathbf{aa}_t} \mathbf{K}_t$
  $\mathbf{v}_t \leftarrow \mathbf{Q}_{\mathbf{sa}_t} \mathbf{k}_t + \mathbf{K}_t^\intercal \mathbf{Q}_{\mathbf{aa}_t} \mathbf{k}_t + \mathbf{q}_{\mathbf{s}_t} + \mathbf{K}_t^\intercal \mathbf{q}_{\mathbf{a}_t}$
  $v_t \leftarrow \frac{1}{2} \mathbf{k}_t^\intercal \mathbf{Q}_{\mathbf{aa}_t} \mathbf{k}_t + \mathbf{q}_{\mathbf{a}_t}^\intercal \mathbf{k}_t + q_t$
return $V^{\mu_\theta}, Q^{\mu_\theta}$

Algorithm 3: Policy optimization via Stochastic Value Gradients

Input: Environment $\mathcal{M}$ (black-box)
Output: Sub-optimal policy μ_θ

1 Initialize θ
2 $\mathcal{D} \leftarrow \emptyset$
3 **for** $k = 0, \dots, P$ **do**
4 Collect $\mathcal{D}_k = \{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_i'\}_{i=1}^N$ by interacting with $\mathcal{M}$
5 $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_k$
6 $p_\psi \leftarrow$ `model_learning(`$\mathcal{D}$`)`
7 $\widetilde{\nabla} J(\theta) \leftarrow$ `estimator(`$\psi, \mathcal{D}$`)`
8 $\theta \leftarrow$ `optimizer(`$\theta, \widetilde{\nabla} J(\theta)$`)`
9 **return** μ_θ

2.3 Stochastic Value Gradient methods

In the broader RL context, methods that learn parameterized policies, often called *policy optimization* methods, have gained traction in the recent decade. As function approximation research, specially on deep learning, has advanced, parameterized policies were able to unify perception (processing sensor readings from the environment) and decision-making (choosing actions to maximize return) tasks (Mnih *et al.*, 2015). To improve such parameterized approximators from data, the workhorse behind many policy optimization methods is Stochastic Gradient Descent (SGD) (Ruder, 2016). I.e., policies are iteratively updated by navigating the expected return (value) surface, using a first-order approximation of eq. (2.2), towards a local (and possibly global) optimum. Thus, it is imperative to estimate the gradient of a policy's value w.r.t. its parameters, a.k.a. the *value gradient*, from data (states, actions and rewards) collected via interaction with the environment. SVG methods take a *model-based* approach to estimating the value gradient.

Algorithm 3 outlines a generic SVG algorithm. We describe an iteration (lines 4-8) in

what follows. As usual in policy optimization, each major iteration starts with collecting data (experiences) in the environment (line 4). This may be done by following the current policy being optimized, a.k.a. the *target policy*, or an exploratory policy for discovering previously untested states and actions, a.k.a. a *behavior policy*. The algorithm then combines this data with previously observed data in a *replay buffer*, i.e., a big dataset of transitions (line 5). The specifics of how this replay buffer is maintained vary between algorithms and are beyond the scope of this work. We detail our approach in the experiments where we make use of replay buffers.

The algorithm then proceeds to fit its environment model to the replay buffer data (line 6). Next, a gradient estimation procedure is used to derive an approximate value gradient using the current model and replay buffer data (line 7). Finally, the policy parameters are updated using a stochastic optimization algorithm, e.g., Adam or RMSprop (Ruder, 2016), that handles how to apply the approximate gradient to the policy parameters (line 8). What makes model-based algorithms special is the model learning step combined with how the model is used to improve the policy. In the case of SVG algorithms, the gradient estimation procedure is of particular importance. We focus on model learning and gradient estimation in the following subsections.

2.3.1 Model learning

A *model* of the dynamics is a function approximator $p_\psi : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{P}(\mathcal{S})$, where $\psi \in \mathbb{R}^m$ is a parameter vector. Here we consider settings where the model is learned in tandem with the policy as implied by algorithm 3. No pre-training of the model is done with data gathered previously by another agent, as is common with *batch RL* methods (Le *et al.*, 2019; Chen and Jiang, 2019).

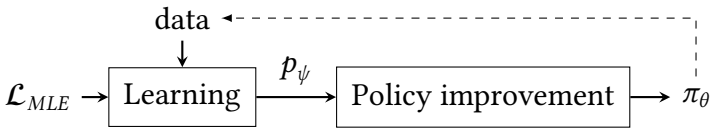

Figure 2.1: *Model learning via Maximum Likelihood Estimation: the agents trains a parametric model p_ψ by maximizing $p_\psi(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ on data gathered in the environment. It then uses its model for policy improvement.*

The vast majority of MBRL algorithms learn their models in a supervised manner by optimizing the model w.r.t. some type of probabilistic loss (Heess *et al.*, 2015; Levine and Abbeel, 2014; Chua *et al.*, 2018; Janner *et al.*, 2019a). The diagram in fig. 2.1 illustrates the general flow of information in these approaches. Model learning is mostly orthogonal to how its used for policy improvement, with only some dependency between the processes because the data used to update the model usually comes, at least in part, by following the current policy.

MBRL algorithms usually optimize the model to maximize the *likelihood* of observed transitions in the environment. Intuitively, the model should assign higher probability to transitions similar to those previously observed from exploration in the environment, generalizing predictions across the state and action spaces. This gives rise to the Maximum

Likelihood Estimation (MLE) model (Myung, 2003), which is obtained by minimizing the loss [1] :

$$\mathcal{L}_{\text{MLE}}(p_\psi) = \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[-\log p_\psi\left(\mathbf{s}' \mid \mathbf{s},\mathbf{a}\right)\right], \tag{2.18}$$

where $\mathcal{D}$ is a dataset of observed transitions from the environment as in algorithm 3. Minimizing equation eq. (2.18) is equivalent to minimizing the expected Kullback Leibler (KL) divergence (Gibbs and Su, 2002) between p^* and p_ψ for each $\mathbf{s},\mathbf{a} \in \mathcal{D}$,

$$\text{KL}\left(p^*(\mathbf{s},\mathbf{a}) \| p_\psi(\mathbf{s},\mathbf{a})\right) = \mathbb{E}_{\mathbf{s}'\sim p^*(\mathbf{s},\mathbf{a})}\left[\log p^*\left(\mathbf{s}' \mid \mathbf{s},\mathbf{a}\right) - \log p_\psi\left(\mathbf{s}' \mid \mathbf{s},\mathbf{a}\right)\right], \tag{2.19}$$

which is a common way of measuring divergence between distributions, that is, how dissimilar two distributions are. The KL divergence between distributions is nonnegative; and it is zero only if both are equivalent [2] .

The approach outlined above has the advantage of being general enough to be a "plug and play" solution to model learning, regardless of how the model will be used afterwards. Furthermore, many generative model architectures are specially designed to have tractable likelihood computation (or a suitable approximation), which makes it easy to swap between different models to see which provides better performance. Plus, once a model has achieved sufficient accuracy, it can be used for different tasks in the same environment (i.e., by changing the reward function).

There are alternative model learning methods tailored to the specific model-based policy improvement step (A. M. Farahmand *et al.*, 2017; A.-m. Farahmand, 2018; Asadi *et al.*, 2018; D'Oro, Metelli, *et al.*, 2019; Donti *et al.*, 2017; Amos, Rodriguez, *et al.*, 2018). However, these are still niche methods and not thoroughly tested in RL benchmarks; thus, we leave an analysis of their role in SVG algorithms to future work.

2.3.2 Value gradient estimation

While value gradient estimation is not unique to SVG methods, the way in which they compute the gradient is what sets them apart from other RL algorithms. We deviate slightly from the definition by Amos, Stanton, *et al.* (2020) [3] and consider SVG methods any RL algorithm that: (a) uses a model-based K-step expansion of a value function and (b) computes the gradient of this expansion using the derivatives of the dynamics model. For instance, consider the following 1-step expansion of the state-value function:

$$V^\mu(\mathbf{s}) = \mathbb{E}_{\mathbf{s}'\sim p_\psi(\mathbf{s},\mu(\mathbf{s}))}\left[R(\mathbf{s},\mu(\mathbf{s}),\mathbf{s}') + V^\mu(\mathbf{s}')\right] . \tag{2.20}$$

Note that the above differs from eq. (2.7) by using the dynamics model to sample the next state. We illustrate how to compute eq. (2.20) by adopting the formalism of Stochastic Computation Graphs (SCGs).

---

[1] The log is for computational convenience since it allows us to turn the product of densities into a sum in the Monte-Carlo estimate of the loss.

[2] We don't use the term *metric* here since the KL is not symmetric.

[3] The work referenced originally considered simply "methods that update the policy with an H-step value expansion".

Definition 2.3.1: Stochastic Computation Graph (Schulman, Heess, *et al.*, 2015b)

A directed, acyclic graph, with three types of nodes:

1. *Input nodes, which are set externally, including the parameters we differentiate with respect to.*
2. *Deterministic nodes, which are functions of their parents.*
3. *Stochastic nodes, which are distributed conditionally on their parents.*

Each parent v of a non-input node w is connected to it by a directed edge (v, w).

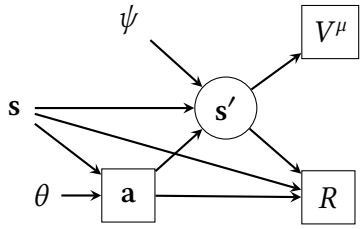

Figure 2.2: *SCG of 1-step state-value expansion using a dynamics model. Notice how the the next-state is a stochastic "function" of the previous state, action and model parameters.*

Figure 2.2 shows the SCG for eq. (2.20). We adopt the graphical notation from Schulman, Heess, *et al.* (2015b): squares denote deterministic nodes, circles, stochastic nodes, and borderless nodes with no parents, inputs nodes. The challenge is to compute the gradient of the reward and state value when they depend on a stochastic variable ($\mathbf{s}'$) which in turn depends on the model parameters. SVG algorithms compute the gradient of this equation by first using the *reparameterization trick* (Schulman, Heess, *et al.*, 2015b):

$$V^\mu(\mathbf{s}) = \mathbb{E}_\xi \left[R(\mathbf{s}, \mu(\mathbf{s}), f_\psi(\mathbf{s}, \mu(\mathbf{s}), \xi)) + V^\mu(f_\psi(\mathbf{s}, \mu(\mathbf{s}), \xi)) \right] . \tag{2.21}$$

I.e., by transforming the model from a probability measure to a deterministic function of states, actions and an external noise variable. For instance, sampling from a normal distribution, $y \sim \mathcal{N}(x, \sigma)$, is equivalent to computing $y = x + \sigma\xi$, where $\xi \sim \mathcal{N}(0, 1)$. The advantage of this trick is that computing the gradient of this expectation is straightforward since ξ is independent of model parameters, while $\mathbf{s}'$ is not.

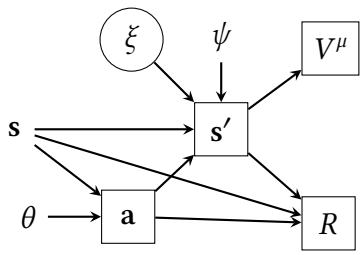

Figure 2.3: *SCG of 1-step state-value expansion using a reparameterized dynamics model. Notice how the the next-state is a deteriministic function of the previous state, action, model parameters and an exogenous noise variable.*

Figure 2.3 shows the SCG of the reparemeterized model-based 1-step expansion of the

state-value function. The gradient of this expression becomes

$$\nabla V^{\mu}(\mathbf{s}) = \mathbb{E}_{\xi}\left[\nabla_{\mathbf{s}} R(\mathbf{s}, \mu(\mathbf{s}), f_{\psi}(\mathbf{s}, \mu(\mathbf{s}), \xi)) + \nabla_{\mathbf{s}} V^{\mu}(f_{\psi}(\mathbf{s}, \mu(\mathbf{s}), \xi))\right] ,$$

where we have not applied the chain rule of derivatives for clarity. Note that we are able to push the gradient inside the expectation, making it easy to estimate the gradient via Monte Carlo methods. This approach is also known as the *pathwise derivative estimator* (Schulman, Heess, *et al.*, 2015b). However, it is limited to environments with continuous states and actions, of which LQG is a subset of.

As we'll see in chapter 4, there is more than one way to combine the gradient of value functions into the full value gradient. Nevertheless, as long as an algorithm uses some form of the above to compute the value gradient, we consider it an SVG method. This allows us to include approaches that use model derivatives to learn a parametric value function (Fairbank and Alonso, 2012; D'Oro and Jaskowski, 2020) before estimating the full value gradient. As we'll see in chapter 5, using the model to directly estimate the value gradient or to learn a parametric value function can be complementary approaches.

Chapter 3

The LQG Benchmark

In this chapter, we describe our test environment and explain the design decisions behind its implementation. We begin by explaining how we generate random LQGs as RL environments in section 3.1. Then, in section 3.2, we describe how to randomly generate initial policies for given LQGs that are viable to deploy for data collection and gradient estimation. Section 3.3 shows how we can exploit the knowledge of the environment's dynamics and reward functions to compute a policy's value (expected return from the initial state) and its gradient. Finally, section 3.4 shows visualizations of the environment and policy variety induced by our generating process, corroborating its usefulness in stress testing RL, and specially SVG, methods.

3.1 Randomized LQG instances

To perform our investigations across a wide variety of scenarios, we define how to sample LQG instances, $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, p^*, \rho)$, to run our experiments on. The main hyperparameters are: state dimension (n), action dimension (d) and time horizon (H). From these parameters we define the state space $\mathcal{S} = \mathbb{R}^n$, action space $\mathcal{A} = \mathbb{R}^d$ and timesteps $t \in \mathcal{T} = \{0, \dots, H-1\}$. In what follows, we define the random generation process for the transition dynamics, cost function, and initial state distribution

3.1.1 Transition dynamics

Firstly, we define how to sample the linear stochastic dynamics' parameters, $\{\mathbf{F}_t, \mathbf{f}_t, \Sigma_t\}_{t\in\mathcal{T}}$. Our defaults correspond to a simple setting: stationary LQG dynamics with no transition bias and standard Gaussian transition noise:

$$p^*(\cdot \mid \mathbf{s}, \mathbf{a}) = \mathcal{N}(\cdot \mid \mathbf{F}_\mathbf{s}\mathbf{s} + \mathbf{F}_\mathbf{a}\mathbf{a}, \Sigma) = \mathcal{N}(\cdot \mid \mathbf{F}_\mathbf{s}\mathbf{s} + \mathbf{F}_\mathbf{a}\mathbf{a}, I)\,. \tag{3.1}$$

To simplify the notation, let $\mathbf{F}_\mathbf{s} = \mathbf{A}$ and $\mathbf{F}_\mathbf{a} = \mathbf{B}$ (following the usual notation from the Control literature). Thus, the average of the next state distribution, $p^*(\cdot \mid \mathbf{s}, \mathbf{a})$, is given by $\mathbf{As} + \mathbf{Ba}$. We'll call $\mathbf{A}$ the *passive dynamics* (since it doesn't depend on the action) and $\mathbf{B}$ the *active dynamics* (since it multiplies the action inputs).

We would like a random generating process for $\mathbf{A}$ and $\mathbf{B}$ to satisfy a few properties. Firstly, the set of possible transition dynamics should be diverse enough so that we have a better chance of capturing failure modes of policy and model learning methods, thus having a better estimate of their robustness to environment variations. Secondly, we should be able to perform rollouts with random policies, since data collection is a crucial step in the RL process. This second point elevates the importance of treating **stable** and **unstable** dynamics.

Definition 3.1.1: Linear dynamics stability

Let $\mathbf{A} \in \mathbb{R}^n$ and $\mathbf{s}_{t+1} = \mathbf{A}\mathbf{s}_t$ be the update rule in a discrete-time dynamical system. Then the system is stable if and only if $|\lambda_i| < 1, i = 1, \dots, n$, where $\lambda_i \in \mathbb{C}$ are the eigenvalues of $\mathbf{A}$.

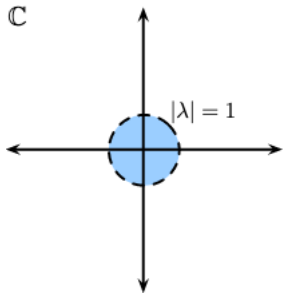

Figure 3.1: *An LQG has stable dynamics if and only if the eigenvalues of the passive dynamics lie within the unit circle in the complex plane (figure by Brunton and Kutz (2019)).*

In the usual Control setting, a system is stable if and only if the eigenvalues of $\mathbf{A}$ lie within the unit circle in the complex plane, as per definition 3.1.1 (Brunton and Kutz, 2019). The state $\mathbf{s}_t$ in an unactuated (that is, when the control inputs $\mathbf{a}_t$ are all zero) stable system will tend to zero over time, regardless of starting state $\mathbf{s}_0$. On the other hand, some state variables in an unstable system (i.e., when at least one eigenvalue of $\mathbf{A}$ has magnitude 1 or greater) will tend to either plus or minus infinity. In colloquial terms, we say the state tends to *blow up*. This is undesirable for several reasons, two of them being numerical simulation errors and exploding costs.

Example 3.1.1: Unstable laplacian dynamics (Recht, 2019)

Consider an idealized instance of data center cooling, a popular application of RL. Define the model to have three heat sources coupled to their own cooling devices. Each component of the state $\mathbf{s} \in \mathbb{R}^3$ is the internal temperature of one heat source, and the sources heat up under a constant load. They also shed heat to their neighbors. This can be approximately modeled by a linear dynamical system with state-transition matrices

$$\mathbf{A} = \begin{bmatrix} 1.01 & 0.01 & 0 \\ 0.01 & 1.01 & 0.01 \\ 0 & 0.01 & 1.01 \end{bmatrix}, \qquad \mathbf{B} = I\,.$$

Note that the open loop system here is unstable: with any nonzero initial condition, the state vector will blow up because the limit of $\mathbf{A}^k$ is infinite. Moreover, if a method

estimates one of the diagonal entries of $\mathbf{A}$ *to be less than 1, we might guess that this mode is actually stable and put less effort into cooling that source. So it is imperative to obtain a high-quality estimate of the system's true behavior for near-optimal control. Or, rather, we must be able to ascertain whether our current policy is safe, or the consequences can be disastrous.*

However, working exclusively with stable dynamics is limiting and not representative of some relevant real problems that may be interpreted as LQGs. Example 3.1.1 illustrates how unstable systems may arise in practice, the consequences of not stabilizing them, and how important it is that we learn accurate dynamics models when instability is at play. Given the relevance of unstable systems, we set out to find ways to incorporate them in our experiments.

Generating unstable dynamics

We took the following steps to create diverse passive dynamics $\mathbf{A}$ to run our experiments on.

1. Sample eigenvalues $\{\lambda_i\}_{i=1}^n$ such that $|\lambda_i| \sim \mathcal{U}(a, b)$. By setting $b \geq 1$ we have a nonzero probability of sampling an unstable system.
2. Sample column eigenvectors as an orthogonal matrix $\mathbf{W}$ using the `scipy` package [1]
3. Compute the passive dynamics as $\mathbf{A} = \mathbf{W}\,\mathrm{diag}(\lambda_1, \dots, \lambda_n)\mathbf{W}^{-1} = \mathbf{W}\,\mathrm{diag}(\lambda_1, \dots, \lambda_n)\mathbf{W}^{\mathsf{T}}$

For step 1 above, we implement the following: (a) we ensure that each eigenvalue has an algebraic multiplicity of 1, i.e., there are no repeated eigenvalues; (b) we sample eigenvalue magnitudes uniformly from a discretized interval between a and b using `np.linspace(a, b, 1000)` to ensure values are sufficiently different; and (c) we ensure none of the eigenvalues are 0 to prevent the resulting matrix from being rank-defficient. These substeps ensure we can make the resulting system controllable, as we explain in the following subsection.

Generating controllable dynamics

We must ensure that collecting data in our generated systems is practical, even if the passive dynamics are unstable. Thus, the active dynamics $\mathbf{B}$ must allow some behavior policies to stabilize the system, so that state variables don't blow up during exploration. Such systems are said to be *controllable* in the control literature.

Definition 3.1.2: Linear dynamics controllability

The Hautus lemma for controllability (Dayawansa, 2001; Zabczyk, 1992), a.k.a. the PBH test, says that given passive and active dynamics $\mathbf{A}$ *and* $\mathbf{B}$ *the following are equivalent:*

1. *The pair* $(\mathbf{A}, \mathbf{B})$ *is controllable*

[1] We used the Haar distribution: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ortho_group.html

2. *For all* $\lambda \in \mathbb{C}$ *it holds that* $\text{rank}[\lambda I - \mathbf{A}, \mathbf{B}] = n$
3. *For all* $\lambda \in \mathbb{C}$ *that are eigenvalues of* $\mathbf{A}$ *it holds that* $\text{rank}[\lambda I - \mathbf{A}, \mathbf{B}] = n$

Intuitively, controllability assures us that it is possible to "steer" the state variables towards arbitrary values using the appropriate action inputs. Thus, there is a policy $\mu : \mathcal{S} \mapsto \mathcal{A}$ that can interact with the system safely even if the dynamics are unstable. This property is crucial for RL, which depends on repeated data collection by interaction with the environment.

We explore the third equivalence in definition 3.1.2 to generate $\mathbf{B}$ given $\mathbf{A}$ so that the pair $(\mathbf{A}, \mathbf{B})$ is controllable. The process is as follows.

1. We sample each entry in $\mathbf{B}$ independently from $\mathcal{N}(0, 1)$
2. We normalize each column $\mathbf{B}_i$ by its norm: $\mathbf{B}_i \longleftarrow \mathbf{B}_i / \|\mathbf{B}_i\|$
3. If any entry in $\mathbf{B}$ is zero (or close to it), we reject this sample and go back to step 1
4. Finally, we set $\mathbf{B} \longleftarrow \mathbf{W}\mathbf{B}$, ensuring the resulting active dynamics have a component in each eigenvector direction

Step 2 ensures each control direction has unit magnitude. Furthermore, $\mathbf{B}_i$ is uniformly distributed in the unit sphere, [2] ensuring a diversity of directions

Since we ensured that $\mathbf{A}$ is full rank, $\lambda I - \mathbf{A}$ will be rank-defficient precisely when λ is an eigenvalue of $\mathbf{A}$ and its null space will be the eigenvector direction corresponding to that eigenvalue (which is unique since each eigenvalue has an algebraic and thus geometric multiplicity of 1). Our generating process for the active dynamics thus ensures that $\text{rank}[\lambda I - \mathbf{A}, \mathbf{B}] = n$ always since each column of $\mathbf{B}$ has a non-zero component in each eigenvector direction. Thus, by condition 3 of definition 3.1.2, the system is controllable.

With a controllable system, we can be assured that there's a policy that stabilizes it, thus, enabling safe and numerically stable exploration of the environment. What "stabilizes" means and how we may derive such a policy is explained in section 3.2.

3.1.2 Cost function

For each LQG instance, we generate a random, stationary quadratic cost with no state-action cross terms ($\mathbf{C}_{\text{sa}} = \mathbf{0}$ and $\mathbf{C}_{\text{as}} = \mathbf{0}$) and no linear term ($\mathbf{c} = \mathbf{0}$):

$$c(\mathbf{s}, \mathbf{a}) = \frac{1}{2} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix}^{\mathsf{T}} \mathbf{C} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} + \mathbf{c}^{\mathsf{T}} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} = \frac{1}{2} \left(\mathbf{s}^{\mathsf{T}} \mathbf{C}_{\text{ss}} \mathbf{s} + \mathbf{a}^{\mathsf{T}} \mathbf{C}_{\text{aa}} \mathbf{a} \right), \tag{3.2}$$

$$R(\mathbf{s}, \mathbf{a}, \mathbf{s}') = -c(\mathbf{s}, \mathbf{a}). \tag{3.3}$$

We use the `sklearn` library to generate $\mathbf{C}_{\text{ss}}$ and $\mathbf{C}_{\text{aa}}$ randomly while ensuring they are symmetric positive definite matrices, as definition 2.2.1 requires. [3] Furthermore, to simplify

[2] http://corysimon.github.io/articles/uniformdistn-on-sphere/

[3] We use scikit learn's `make_spd_matrix` function.

matters, we set $R_f(\mathbf{s}) = 0$ always.

3.1.3 Initial state distribution

We set the initial state distribution to a standard multivariate diagonal Gaussian of appropriate dimensionality: $\rho(\mathbf{s}) = \mathcal{N}(\mathbf{s} \mid \boldsymbol{\mu}_\rho, \Sigma_\rho) = \mathcal{N}(\mathbf{s} \mid \mathbf{0}, I)$.

3.2 Randomized linear policies

In this work we consider the problem of learning SVGs for linear policies in LQGs. While exclusively studying linear policies may seem limiting, it allows us to compute optimal solutions to the control and prediction problems in LQGs via algorithms 1 and 2 respectively. These solutions then serve as references for comparison against learned solutions. Furthermore, linear policies are useful even outside the realm of LQGs, as research has found that they provide reasonable performance even in nonlinear systems, while being easier to interpret and optimize (Rajeswaran *et al.*, 2017).

With a controllable pair $(\mathbf{A}, \mathbf{B})$, we can derive a dynamic gain $\mathbf{K} \in \mathbb{R}^{d \times n}$ such that the eigenvalues of $\mathbf{A} - \mathbf{BK}$ are any of our choosing. Thus, the system $\mathbf{s}' = (\mathbf{A} - \mathbf{BK})\mathbf{s}$ can be manipulated via our choice of $\mathbf{K}$ to be stable. We say that $\mathbf{K}$ *places* the eigenvalues of the resulting system. Furthermore, the dynamic gain defines a policy, $\mu_\theta(\mathbf{s}) = -\mathbf{Ka}$. If $\mathbf{K}$ places the eigenvalues of the system in a stable range (i.e., with magnitudes less than one), we say that the policy stabilizes the system.

Although the theoretical framework discussed here is more often used for stationary systems with infinite horizon, it is equally useful when the horizon is limited. That is because state variables in an unstable system will blow up in a relatively small amount of timesteps (enough to incur numerical simulation errors).

The details of how to compute $\mathbf{K}$ are beyond the scope of this work. Fortunately, given that we're able to generate controllable $(\mathbf{A}, \mathbf{B})$ pairs, there are a variety of existing algorithms to compute a dynamic gain that places the eigenvalues of the system in a desired combination (Kautsky *et al.*, 1985; Tits and Yang, 1996). The procedure we use to generate a random stabilizing policy is as follows:

1. Sample target eigenvalues $\Lambda = \{\hat{\lambda}_i \mid |\hat{\lambda}_i| \sim \mathcal{U}(0, 1)\}_{i=1}^n$
2. Use `scipy.signal.place_poles` to compute $\mathbf{K}$ such that the eigenvalues of $\mathbf{A} - \mathbf{BK}$ are Λ. [4]
3. Set $\mu_\theta(\mathbf{s}) = -\mathbf{Ks} + \mathbf{0}$

The procedure above ensures that the resulting policy can be safely (in terms of numerical stability or costs, as in example 3.1.1) deployed to collect data in the LQG environment. This policy can be used as a starting point for optimization via RL or a fixed target for the environment model to learn its expected return and SVG. Finally, our policy generating procedure serves to mimic practical situations were engineers have devised a

[4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.place_poles.html

policy which can keep a system stable, but is not able to optimize running costs, which is where RL can serve to fine-tune it.

3.3 Analytical solutions

In this section, we describe how we may exploit the knowledge of the LQG underlying the environment to compute a given linear policy's value and SVG analytically. The true value and SVG of a policy are the targets for SVG algorithms, which aim to approximate these quantities using models, samples and optimization. The ability to compute these targets efficiently is one of the main advantages of working with LQGs. With these in hand, we're able to quantify errors in our models with a degree of precision not available in other benchmarks, such as the MuJoCo physics simulator (Emanuel Todorov *et al.*, 2012).

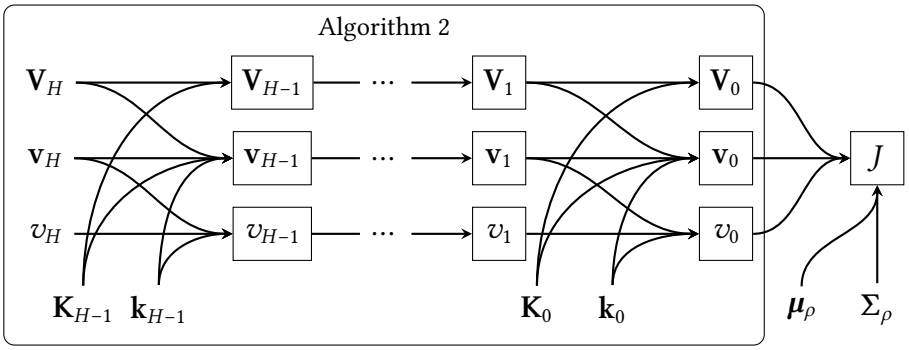

Figure 3.2: *Stochastic Computation Graph of the analytical formula for a policy's value.*

We illustrate how to use algorithm 2 to compute a policy's true value, and its gradient, using the formalism of SCGs (refer back to section 2.3.2). Figure 3.2 shows the SCG for the policy value, J. The steps computed during algorithm 2 are grouped inside the "Algorithm 2" box. For brevity, we omit intermediary steps between the calculations of the state-value function coefficients (such as the computation of the action-value function coefficients). Furthermore, we assume the environment is fixed and omit the input nodes for the transition and cost function coefficients (in practice, the algorithm takes arbitrary LQG parameters as inputs, as in algorithm 2). These simplifications are meant to highlight the paths between the policy parameters (dynamic and static gains) and the policy value.

Note that the parents of J are the state-value function coefficients ($\mathbf{V}_0$, $\mathbf{v}_0$, v_0), the mean ($\boldsymbol{\mu}_\rho$), and covariance matrix (Σ_ρ) for the initial state. This is because the policy value is computed analytically from these parameters as follows.

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\mathbf{s}\sim\rho}[V^{\mu_\theta}(\mathbf{s}, 0)] \\ &= \mathrm{Tr}(\mathbf{V}_0\Sigma_\rho) + \boldsymbol{\mu}_\rho{}^\mathsf{T}\mathbf{V}_0\boldsymbol{\mu}_\rho + \mathbf{v}^\mathsf{T}\boldsymbol{\mu}_\rho + v_0\,, \end{aligned} \tag{3.4}$$

where Tr denotes the trace operator.

We have therefore a fully deterministic formula for the value of a linear policy in an

LQG. [5] This allows us to automatically compute the gradient of the value w.r.t. the policy's parameters by implementing the SCG of fig. 3.2 using an automatic differentiation library. We use PyTorch (Paszke *et al.*, 2019) as our framework of choice to compute the gradients w.r.t. to input nodes via reverse-mode automatic differentiation. We use this procedure in section 3.4 to plot histograms of the policy values and their gradient norms.

3.4 Visualizations

In this section we show visualizations of the environment and policy variety induced by our generating process, corroborating its usefulness in stress testing RL, and specially SVG, methods. These visualizations also ensure that we run our experiments on numerically stable environments in subsequent chapters. Since rewards are all negative in LQGs, we use costs in the plots below (negative rewards), allowing us visualize results in log scale.

3.4.1 Environment diversity

We use variation in the optimal cost for each environment as a measure of problem variety. Figures 3.3 and 3.4 show the distribution of optimal policy costs for randomly generated LQGs with different state and action space dimensionalities. In fig. 3.3 we use the full LQG generating procedure described in section 3.1. We set the limits (a, b) for the magnitude of the passive dynamics's eigenvalues to 0.5 and 1.5 respectively, allowing the occasional generation of unstable, controllable systems. In fig. 3.4 (left) we generate a single transition dynamics for each choice of state and action space dimension and sample several cost functions. This allows us to isolate the contribution of the cost function to the overall variability of the optimal policy cost. Analogously, in fig. 3.4 (right), we sample the transition dynamics multiple times while fixing a cost function. Overall, we see that both components can induce very different optimal costs for the environment depending on their initialization, with the cost function accounting for a slightly greater variability.

Overall, we see that the optimal cost for different environments varies by orders of magnitude. We see this as a good sign of problem variety, which should help us to identify if an RL method is sensitive to environment variation (i.e., if it fails to learn depending on the environment). This is an important characteristic for an RL benchmark, since many learning methods assume an arbitrary MDP in theory, but often require very specific (i.e., favorable) environmental conditions in practice. We also see that both transition dynamics and cost function contribute to this variety.

The graphs also show that we're able to generate and solve LQGs with very high dimensionality. Such instances provide a way to test if our RL methods scale with problem dimensionality. Moreover, larger instances usually require more samples for policy and model learning, providing us with a way to increase the difficulty for RL agents.

[5] We didn't need to use the formalism of SCGs in this case, however, it highlights that fact that, even though the policy value is an expectation, it has a deterministic, practical formula.

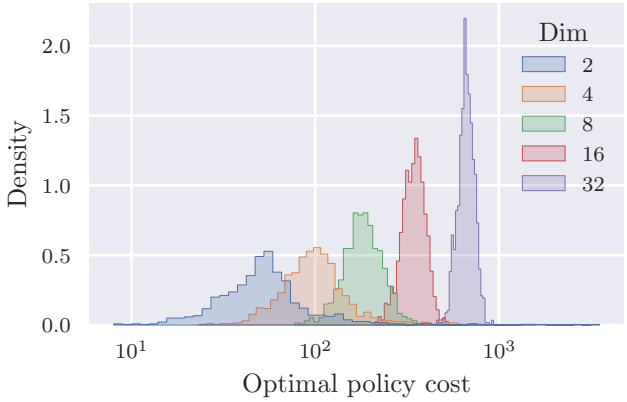

Figure 3.3: *Distributions of optimal policy costs for randomly generated LQGs following the procedures described in section 3.1. Each histogram corresponds to different dimensionalities of state and action spaces.*

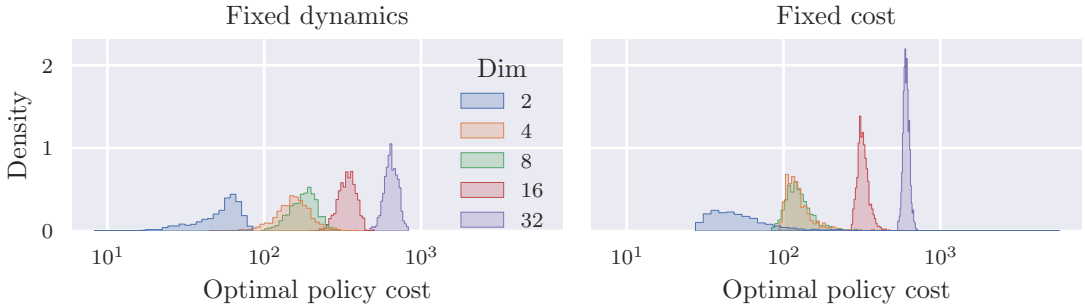

Figure 3.4: *Distributions of optimal policy costs for randomly generated LQGs with fixed dynamics (left) and cost function (right).*

3.4.2 Policy diversity

We're also interested in generating sub-optimal behavior policies for data collection and as starting points for RL algorithms to optimize. We use the procedure described in section 3.2 to generate several stabilizing policies.

Figure 3.5 shows the distribution of the suboptimality gap of randomly generated policies for three LQGs, each with different state and action space dimensions. The suboptimality gap of a policy with total cost C depends on the optimal cost for the respective environment, $C^\star$, and is computed as

$$\left|1 - \frac{C}{C^\star}\right|. \tag{3.5}$$

Thus, we have a sense for the difference between the randomly generated policies and the optimal one measured by the relative difference between their total costs (for each environment). We can see that the cost for a policy generated by our procedure is, in general, orders of magnitude larger than the cost for the optimal one in that environment, specially with larger state and action dimensions. Thus, our implementation successfully produces initial policies for numerically stable interaction with the environment while

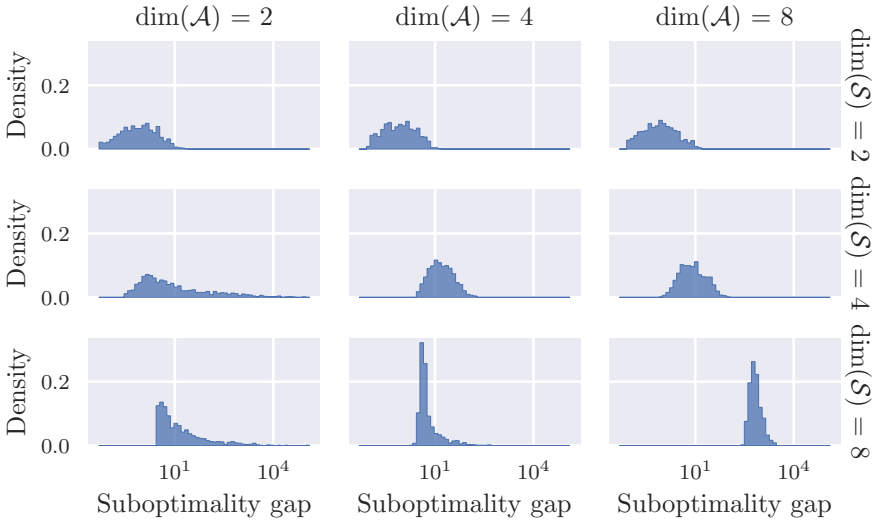

Figure 3.5: *Distributions of total costs of random stabilizing policies for fixed LQGs with differing dimensionalities.*

being sub-optimal and thus suitable for optimization via RL.

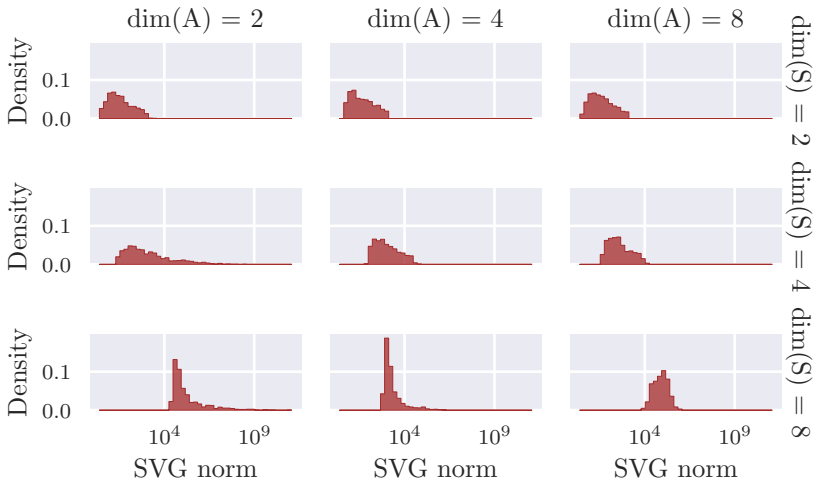

Figure 3.6: *Distribution of value gradient norms of random stabilizing policies for fixed LQGs with differing dimensionalities.*

Figure 3.6 shows, for the same environments and policies as in fig. 3.5, the histograms for the true SVG norm of the generated policies. We use the procedure described in section 3.3 to compute the SVG for random policies analytically. We can see that the average gradient norm for random policies tends to increase with problem dimensionality. One should also keep in mind that the norms are quite higher than that of approximate gradients in RL algorithms, which usually don't surpass the thousands. This further motivates our adoption of special gradient quality metrics from prior works (Ilyas *et al.*, 2020), such as the cosine similarity, that are less dependent on the magnitude of the gradient and instead focus on its direction.

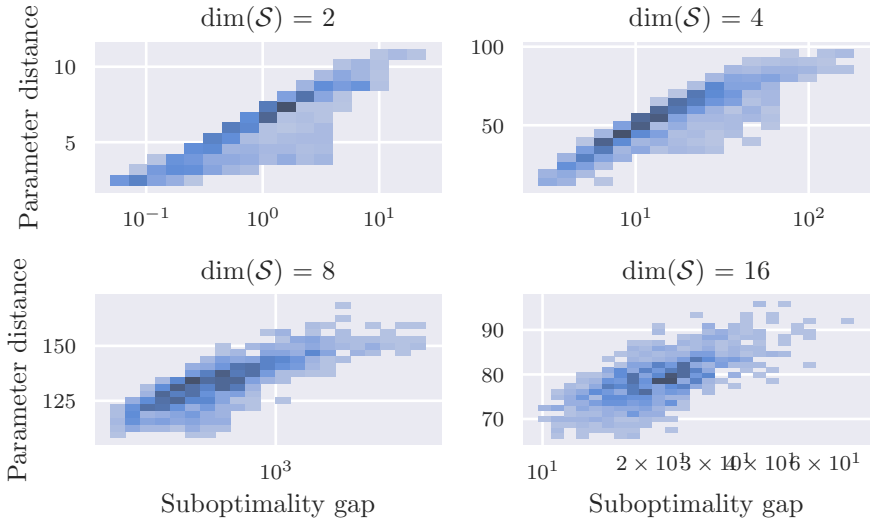

Figure 3.7: *Distribution of the norm of the difference between random and optimal policies vs. the suboptimality gap (in log scale). Generated by fixing an LQG and sampling 1000 random policies.*

Lastly, fig. 3.7 shows histograms of suboptimality gap vs. distance to optimal parameters of random policies for different environment sizes. In this plot, we make $\dim(\mathcal{S}) = \dim(\mathcal{A})$, so only the state dimensionality is indicated for each subplot. Darker colors indicate that more datapoints were observed in the respective region. Distance to optimal parameters is measured as the $L2$-norm of the difference between the random policy's parameters, θ, and the optimal parameters, $\theta^\star$: $\|\theta - \theta^\star\|_2$. Notice how the suboptimality gap (in log space) correlates almost linearly with the distance to optimal parameters. Thus, the distance to optimal parameters could be a good measure of policy suboptimality. However, we prefer to keep the suboptimality gap as our measure going forward as it is more interpretable.

3.5 Discussion

Overall, the experiments in this section indicate that we have a robust and flexible benchmark for evaluation of SVG algorithms. We demonstrated that we can reliably generate both stable and unstable environments as well as policies that stabilize either type. These policies also differ between each other and have tractable ground-truth value gradients. Finally, there's a lot of room for SVG methods to optimize these random policies toward the optimal ones. Our next chapter will leverage this benchmark to investigate the impact of one design choice in SVG algorithms: the gradient estimator formula.

Chapter 4

Value Gradient Estimation

In this chapter we explore our first research questions, regarding the different gradient estimators (Schulman, Heess, *et al.*, 2015a; Mohamed *et al.*, 2020) used in the SVG literature. Our focus here is to study, empirically, the properties of these estimators alone, without mixing in the problem of learning the environment model. Therefore, we leave the study of model learning techniques and their impact for the subsequent chapter. Our experiments aim to highlight the differences between these estimators and why one may be preferred over the other in practice. In doing so, we also aim to fill the gap between the theory and implementations of existing SVG algorithms by providing explicit measures of the advantages of certain design choices.

4.1 Gradient estimation in SVG methods

As indicated in algorithm 3 (line 7), one of the main steps in an SVG method is leveraging a differentiable model of the environment's dynamics to produce an estimate of the value gradient. For the purposes of this chapter, we'll consider the model to be a non-parametric probability density $\mathbf{s}' \sim \hat{p}(\mathbf{s}, \mathbf{a})$, reparameterizable as $\mathbf{s}' = \hat{f}(\mathbf{s}, \mathbf{a}, \xi)$, where ξ is an independent random variable (refer back to section 2.3.2). Also, we'll consider that the reward function is known and only dependent on the current state and action: $R(\mathbf{s}, \mathbf{a})$.

We first address the obvious way to use the model to estimate the value gradient: simulate episodes with the current policy and compute the gradient of the average return. The formula for this estimator would be

$$\nabla_\theta \, \mathbb{E}_{\mathbf{s}_0 \sim \rho} \left[\sum_{t=0}^{H-1} R(\mathbf{s}_t, \mu_\theta(\mathbf{s}_t)) \right], \tag{4.1}$$

where $\mathbf{s}_{t+1} \sim \hat{p}(\mathbf{s}_t, \mu_\theta(\mathbf{s}_t))$. Heess *et al.* (2015) name this the SVG(∞) estimator, since its generalization to infinite-horizon environments would require infinite applications of the dynamics model. We can approximate this by Monte Carlo: sampling the initial state uniformly at random from a dataset of previously observed initial states, simulating

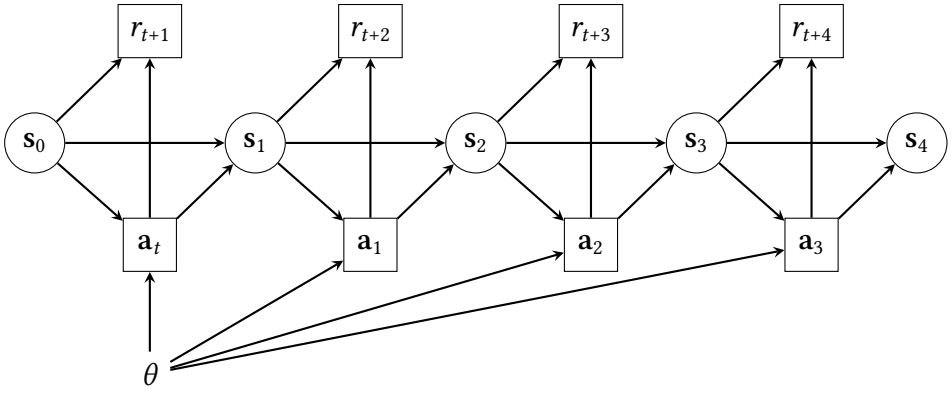

Figure 4.1: *Stochastic computation graph of SVG(∞) for an environment with horizon $H = 4$.*

trajectories with the policy and model and then computing the gradient of the average returns. Figure 4.1 shows the SCG for SVG(∞) in a limited-horizon environment. Although eq. (4.1) differentiates an almost exact replica of the policy's value, it is not practical as a value gradient estimator for two reasons: (1) state predictions chained together for several timesteps tend to compound model errors (Janner *et al.*, 2019b) and (2) backpropagating the reward gradients for long prediction horizons tends to incur vanishing or exploding gradients (Hochreiter, 1998).

Instead, we consider formulas that rely less on recursive model prediction. Often, these produce a value different from the policy's value, even with a perfect model. However, these formulas yield an estimate of the value gradient when differentiated. We introduce two such value gradient estimation formulas in the following subsections. Here we contrast their theoretical underpinnings, while in later sections we propose an empirical analysis of their estimation quality and impact in a generic SVG algorithm.

4.1.1 The MAAC estimator

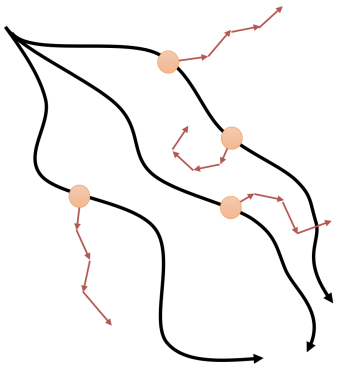

Figure 4.2: *Illustration of model-based rollouts used to estimate the SVG. Solid black lines indicate the trajectories generated by the (behavior) policy while exploring the environment. Circles indicate states randomly sampled from the replay buffer and arrows the model-based rollouts using the target policy to choose actions.*

A common approach to leveraging a differentiable model $\hat{p}$ (either learned or given through prior knowledge) is as follows. First, the agent collects B states via interaction

with the environment, potentially with an exploratory policy β. The distribution of states induced by following this policy in the environment is called the *stationary distribution* of β. E.g., when we collect several trajectories with β and then sample one of the observed states uniformly at random, we're implicitly sampling from this distribution. The probability measure associated with it is called the *occupancy measure* of β and we denote it as d^β, where $d^\beta(\mathbf{s})$ is the probability density of state $\mathbf{s}$.

After collecting states, the agent generates short model-based trajectories with the current target policy μ_θ, branching off the states previous collected. Figure 4.2 illustrates these model-based trajectories branching off real states previously observed and stored in the replay buffer. Finally, it averages the model-based returns and computes its gradient using backpropagation (I. J. Goodfellow *et al.*, 2016) to form an estimate of the value gradient:

$$\nabla J(\theta) \approx \nabla_\theta \, \mathbb{E}_{\mathbf{s}_t \sim d^\beta} \left[\sum_{l=0}^{K-1} R(\mathbf{s}_{t+l}, \mu_\theta(\mathbf{s}_{t+l})) + \hat{Q}^{\mu_\theta}(\mathbf{s}_{t+K}, \mu_\theta(\mathbf{s}_{t+K})) \right], \tag{4.2}$$

where $\mathbf{s}_{t'} \sim \hat{p}(\mathbf{s}_{t'-1}, \mathbf{a}_{t'-1})$, reparameterized to leverage the pathwise derivative estimator. Here, $\hat{Q}^{\mu_\theta}$ is an approximation (e.g., a learned neural network) of the policy's action-value function (recall definition 2.1.3).

Notice how eq. (4.2) is implicitly computing a K-step expansion of the state-value function from each state sampled from the stationary distribution. Moreover, computing its derivatives requires both $\nabla_{\mathbf{s}} \hat{f}$ and $\nabla_{\mathbf{a}} \hat{f}$. Thus, an algorithm using this estimator fits our definition of an SVG method in section 2.3.2. We refer to eq. (4.2) as the MAAC(K) estimator, as it uses K steps of simulated interaction and was featured prominently in the Model-Augmented Actor-Critic algorithm by Clavera *et al.* (2020). [1] It is implicitly used by other prominent SVG algorithms (Hafner *et al.*, 2020; Amos, Stanton, *et al.*, 2020). Thus, MAAC(K) is a representation of a general approach to estimating the value gradient.

4.1.2 The DPG estimator

We question, however, if eq. (4.2) actually provides good empirical estimates of the true value gradient. To elucidate this matter, we compare MAAC(K) to the value gradient estimator provided by the Deterministic Policy Gradient (DPG) theorem (Silver *et al.*, 2014):

$$\nabla J(\theta) = \mathop{\mathbb{E}}_{\mathbf{s}_t \sim d^{\mu_\theta}} \left[\nabla_\theta \mu_\theta(\mathbf{s}_t) \nabla_{\mathbf{a}} \, Q^{\mu_\theta}(\mathbf{s}_t, \mathbf{a})|_{\mathbf{a}=\mu_\theta(\mathbf{s}_t)} \right]. \tag{4.3}$$

Besides the fact that eq. (4.3) requires us to use the on-policy distribution of states d^{μ_θ}, more subtle differences with eq. (4.2) can be seen by expanding the definition of the action-value function to form a K-step version of eq. (4.3):

$$\nabla J(\theta) = \mathop{\mathbb{E}}_{\mathbf{s}_t \sim d^{\mu_\theta}} \left[\nabla_\theta \mu_\theta(\mathbf{s}_t) \nabla_{\mathbf{a}_t} \left(\sum_{l=t}^{t+K-1} R(\mathbf{s}_l, \mathbf{a}_l) + Q^{\mu_\theta}(\mathbf{s}_{t+K}, \mathbf{a}_{t+K}) \right) \Big|_{\mathbf{a}_t = \mu_\theta(\mathbf{s}_t)} \right], \tag{4.4}$$

[1] Our formula differs slightly from the original in that it considers a deterministic policy instead of a stochastic one.

where $s_{l+1} = \hat{f}(s_l, a_l, \xi)$ and actions are taken with the target policy. We call eq. (4.4) the DPG(K) estimator. Figure 4.3 shows the SCGs of the MAAC(K) and DPG(K) estimators. Because of the $\nabla_{\mathbf{a}_t}(\dots)$ term in eq. (4.4), we're not allowed to compute the gradients of future actions w.r.t. policy parameters in DPG(K), hence why only the first action has a link with θ. On the other hand, MAAC(K) backpropagates the gradients of the rewards and value-function through all intermediate actions.

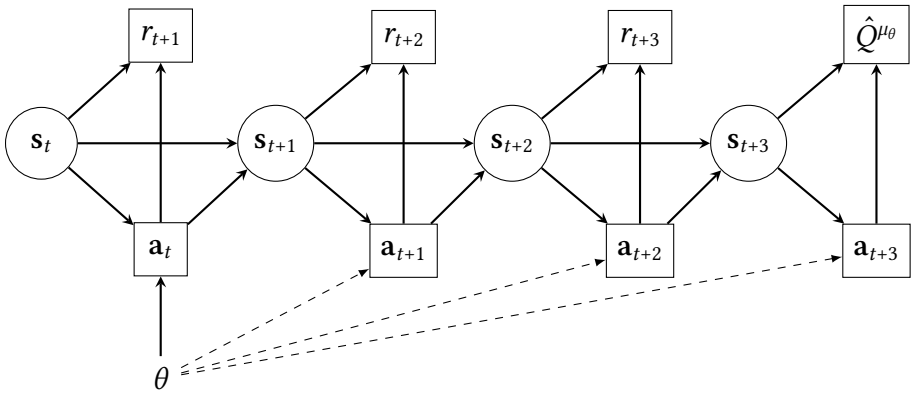

Figure 4.3: *Stochastic computation graphs for value gradients. The dashed edges represent the K-step deterministic dependencies of the policy parameters in MAAC(K) for $K = 3$; DPG(K) ignores these dependencies when backpropagating the action gradients.*

This addition of action-gradient terms from MAAC(K) in relation to DPG(K) suggests that the former is a *biased* estimator, meaning its expected value is different from the actual SVG it is intended to estimate. This difference is orthogonal to the stationary distribution used to sample the initial states for virtual rollouts (we can easily eliminate that difference by using, for both estimators, the target policy to explore the environment and generate the real trajectories in fig. 4.2).

However, to best our knowledge, some of the best-performing SVG methods use MAAC(K) (Clavera *et al.*, 2020; Amos, Stanton, *et al.*, 2020; Hafner *et al.*, 2020), while few use DPG(0) (Fairbank and Alonso, 2012; D'Oro and Jaskowski, 2020) and none use its model-based generalization, DPG(K) with $K > 0$. Our subsequent analysis aims to identify the practical implications of these differences and perhaps help explain why MAAC(K) has been used in SVG methods and not DPG(K).

4.2 Proposed analysis

In this chapter, we propose a fine-grained analysis of the properties of DPG(K) and MAAC(K) in practice. We simplify our evaluation by using *on-policy* versions of the gradient estimators, i.e., by sampling the starting state for model-based rollouts from the stationary distribution of the target policy, d^{μ_θ}, in eqs. (4.2) and (4.3). We also opted for using perfect models of the environment dynamics and rewards, instead of learning them from data, to focus on the differences between gradient estimators. Thus, we approximate the expectations in eqs. (4.2) and (4.3) via Monte Carlo sampling, using the actual transition kernel p^* and reward function R, to generate (virtual) transitions and compute the K-step returns. One can view this setting as the best possible case in an SVG algorithm: when the

model-learning subroutine has perfectly approximated the true MDP, allowing us to focus on the gradient estimation analysis.

We also compute the true action-value function, required for the K-step returns in eqs. (4.2) and (4.3), recursively using algorithm 2. Computing the ground-truth action-value function allows us to further isolate any observed differences between the estimators as a consequence of their properties alone.

4.3 Empirical results

We analyze the behavior of each estimator on two main settings: (I) gradient estimation for fixed policies and (II) impact of gradient quality on policy optimization.

4.3.1 Gradient estimation for fixed policies

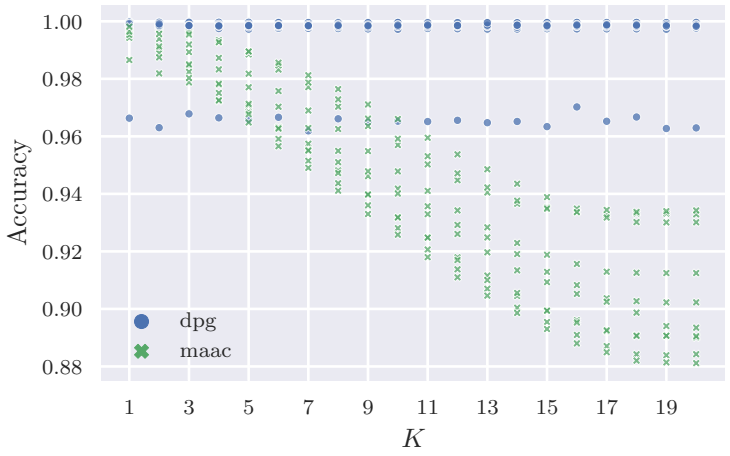

Figure 4.4: *Gradient accuracy for each estimator near convergence for different virtual rollout lengths (K). We used 50000 states sampled from the policy to approximate the expected value.*

Following previous work on model-free policy gradients by Ilyas *et al.* (2020), we evaluate the quality of the gradient estimates, for a given policy, using two metrics: (i) the average cosine similarity with the true policy gradient and (ii) the average pairwise cosine similarity.

Average cosine similarity with the true policy gradient is a measure of gradient *accuracy* and we denote it as such in the following plots. For a given minibatch size B and step size K, we compute 10 estimates of the gradient, each using B initial states sampled on-policy ($\mathbf{s}_t \sim d^{\mu_\theta}$) and K-step model-based rollouts from each state. Then, we compute the accuracy as the average cosine similarity of each of the 10 estimates with the true policy gradient (see section 3.3).

Average pairwise cosine similarity is a measure of gradient *precision* and we denote it as such in the following plots. Again, we compute 10 estimates of the gradient in the

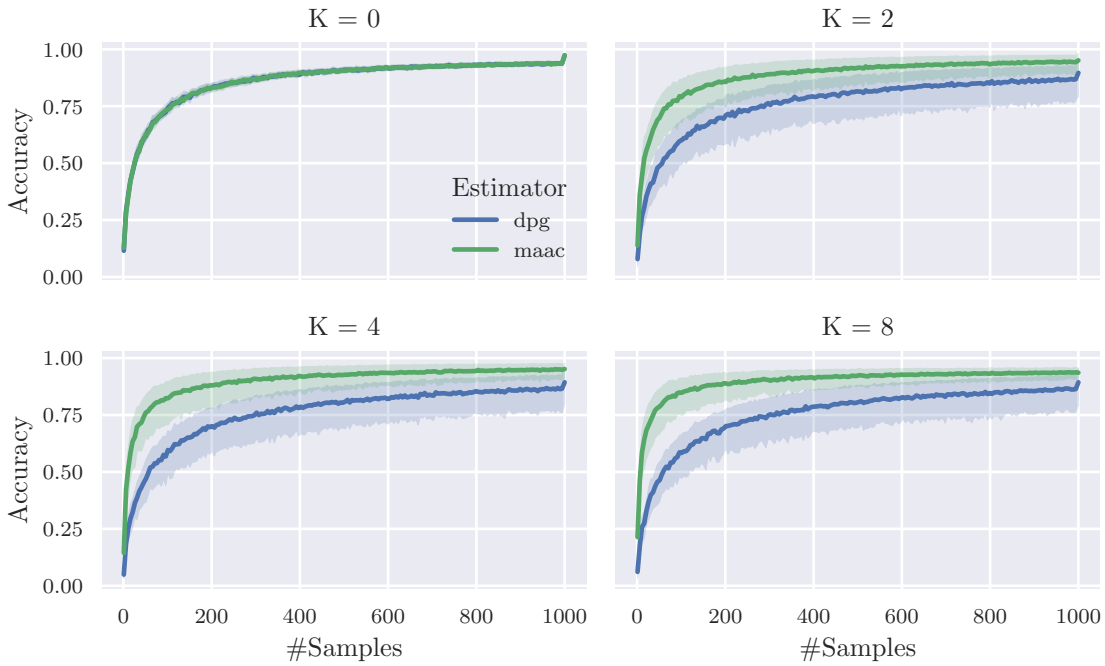

Figure 4.5: *Gradient accuracy for each estimator for different minibatch sizes (B = #Samples) and virtual rollout lengths (K).*

same manner used in computing the accuracy. Then, we compute the precision as the average pairwise cosine similarity of the 10 estimates (the higher this quantity, the lower the variance).

We first analyze the accuracy of each estimator when given enough states from the policy's distribution d^{μ_θ} to approximate their true expected values. Figure 4.4 shows the accuracy obtained by DPG and MAAC for different values of K using 50000 states from the policy's distribution. The LQGs considered have state and action spaces of dimension 2 and horizon of length 20. For each value of K, we initialize 10 different environment-policy pairs and compute the accuracy for each, denoted as different markers in each vertical line. [2] Note how all but one of the instances using DPG(K) converged to the true value gradient, indicating that it is indeed an unbiased estimator. On the other hand, MAAC(K) incurs a larger bias with increasing values of K, indicating that the added action-gradient terms (see fig. 4.3) influence the final gradient direction.

Although the results above indicate that MAAC(K) is biased at convergence, most SVG algorithms operate on a much smaller sample regime. Figure 4.5 shows the accuracy across 10 different environment-policy pairs; this time, however, using smaller sample sizes from the stationary state distribution. Lines denote the average results and shaded areas, the 95% confidence interval. [3] For $K = 0$, the estimators are equivalent, which is verified in practice. In this more practical sample regime, we see that MAAC(K) produces more accurate results, specially for larger values of K.

[2] We use the same 10 random seeds for experiments across values of K.

[3] We use `seaborn.lineplot` to produce the aggregated curves.

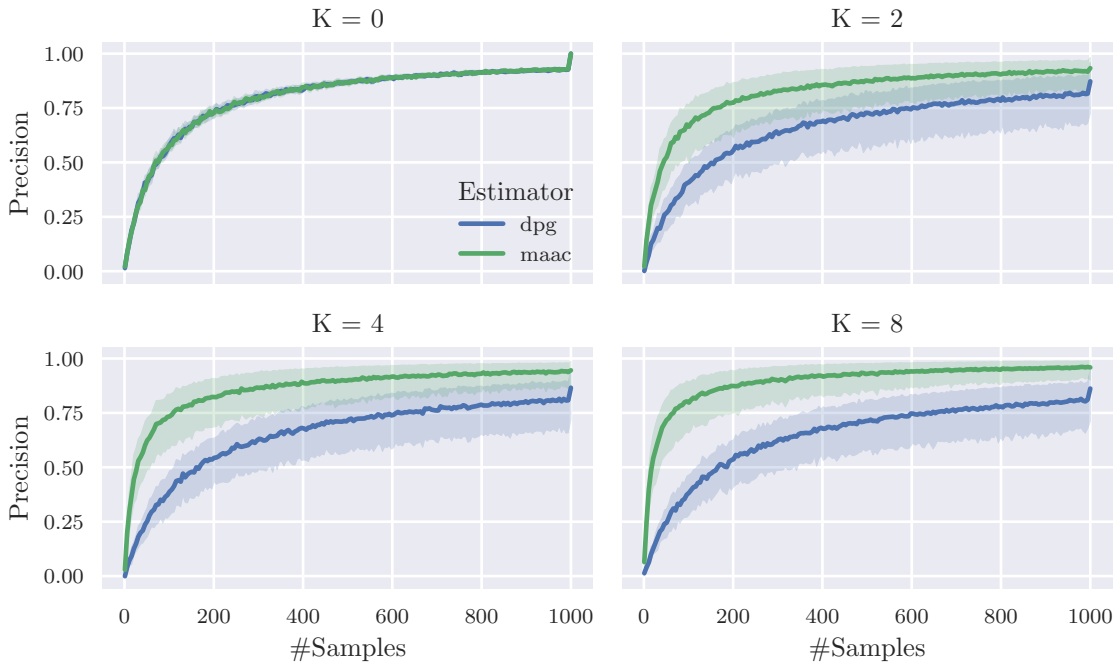

Figure 4.6: *Gradient precision for each estimator for different minibatch sizes (B = #Samples) and virtual rollout lengths (K).*

Similar to fig. 4.5, fig. 4.6 shows the gradient precision in the same setting. We see that the variance of MAAC(K) is lower than that of DPG(K) across all tested values of $K > 0$. Overall, figs. 4.4 to 4.6 illustrate a classic instance of the bias-variance trade-off in machine learning: MAAC(K) introduces bias, although a small one, in return for a much more stable (less variable) estimate of the gradient, whereas the unbiased DPG(K) demands much more samples to justify its use.

Note that the accuracy and precision metrics only account for differences in gradient direction and orientation. The magnitude may also be important, as it influences the learning rate when used to update policy parameters. Figure 4.7 shows that MAAC(K) produces gradients with higher norms compared to DPG(K). One should keep this in mind when choosing the learning rate for SGD, as the following experiments show that the gradient norm has a significant impact on policy optimization.

4.3.2 Impact of gradient quality on policy optimization

Ilyas *et al.* (2020) have shown that model-free policy optimization algorithms can improve a policy despite using poor gradient estimation. Thus, we cannot ascertain that better value gradient estimation translates to more stability or faster convergence in SVG algorithms. We therefore conduct our next experiments comparing the MAAC(K) and DPG(K) estimators by iteratively updating the policy with the gradients produced by each.

Figures 4.8 and 4.9 show learning curves as total cost (negative return) against the number of SGD iterations across several instances of LQGs ($\dim(\mathcal{S}) = \dim(\mathcal{A}) = 2$ and

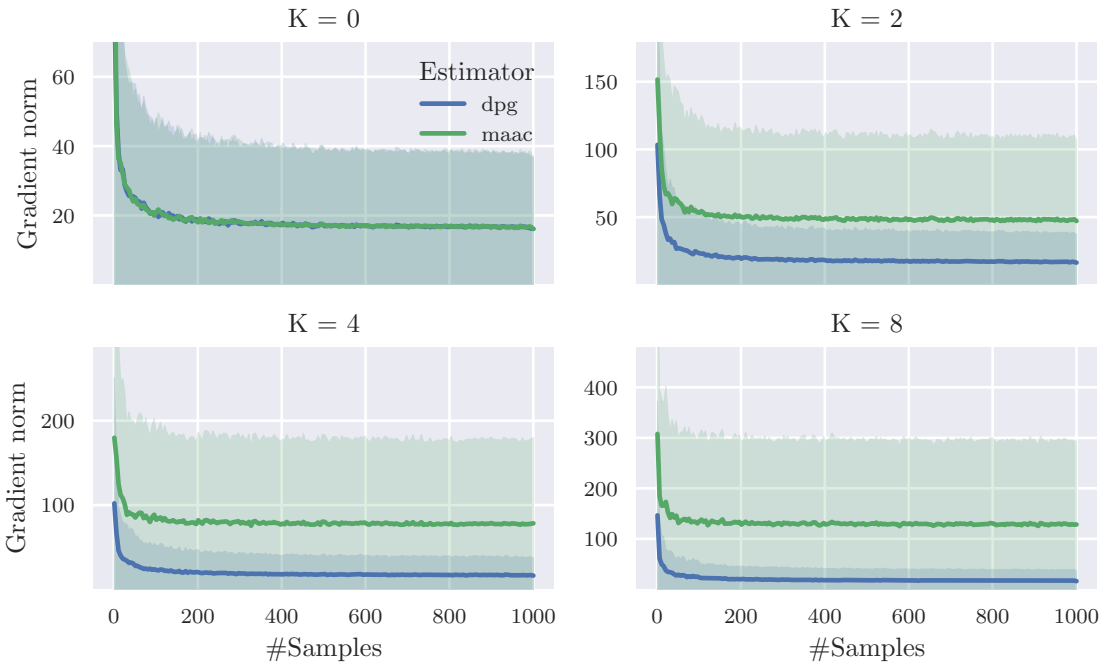

Figure 4.7: *Gradient norm for each estimator for different minibatch sizes (B = #Samples) and virtual rollout lengths (K).*

$H = 20$). We use the same hyperparameters for both estimators. [4] The results in fig. 4.8 suggest that the better quality metrics observed for MAAC(K) in figs. 4.5 and 4.6 do translate to faster and more stable policy optimization. However, if we normalize the gradient estimates before passing them to SGD, as in fig. 4.9, we see that both estimators are evenly matched. These results suggest that the main advantage of MAAC(K) over DPG(K) is in its stronger gradient norm (see fig. 4.7), which has been alluded to by Clavera *et al.* (2020) as a "strong learning signal", inducing a faster learning rate.

Estimator	Time (min)	LQG dimension								
		2	3	4	5	6	7	8	9	10
DPG	1	29.10	218.75	242.94	1730.07	1567.81	4129.88	1100.74	6111.44	7290.04
	3	6.32	53.21	138.89	439.54	465.29	3468.03	552.87	277.38	6445.64
	5	2.66	27.63	91.20	400.31	241.32	2877.18	263.54	2297.37	4830.16
MAAC	1	2.33	20.31	45.05	302.72	255.53	2065.97	340.63	3477.36	5008.28
	3	0.55	3.57	11.28	80.26	38.87	317.76	45.87	1468.44	3568.37
	5	0.38	1.92	6.34	40.13	21.23	290.91	23.23	330.21	2004.51

Table 4.1: *Median suboptimality gap, the percentage difference in expected return against the optimal policy, across 10 seeds. LQG dimension refers to the dimension of state and action spaces. We use $K = 8$ and $B = 20$ for both estimators.*

We also evaluate if our previous findings generalize to higher state-action space dimensions, where sample-based estimation gets progressively harder. Our performance metric is the suboptimality gap, i.e., the percentage difference in expected return between

[4] Learning rate of 10^{-2}, $B = 200$, and $K = 8$.

the current policy and the optimal one: $100 \times (J(\mu_\theta^\star) - J(\mu_\theta))/J(\mu_\theta^\star)$. [5] Table 4.1 summarizes our results with policy optimization with varying LQG sizes and time budgets. [6] We don't normalize gradients in this case, as that is not a common practice in SVG algorithms. [7] Our findings show that the performance gap between DPG(K) and MAAC(K) tends to widen with higher dimensionalities, with policies trained via the latter outperforming those using the former. These results further emphasize the practicality of MAAC(K) over DPG(K), justifying the former's use in recent SVG methods (Amos, Stanton, *et al.*, 2020; Clavera *et al.*, 2020; Hafner *et al.*, 2020).

4.4 Discussion

In this chapter, we take an important step towards a better understanding of current SVG methods. Using the LQG framework, we show that the gradient estimation used by MAAC and similar methods induces a slight bias compared to the true value gradient. On the other hand, using a corresponding unbiased estimator such as the K-step DPG increases sample-complexity due to high variance. Moreover, the MAAC gradient estimates have higher magnitudes, which could help explain the fast learning performance of current methods. Indeed, we found that policies trained with MAAC converge faster to the optimal policies than those using the K-step DPG across several LQG instances.

These results do not take into account the interplay between model learning and gradient estimation, lines 6 and 7 of algorithm 3. In the next chapter, we explore this interplay in more detail.

[5] Recall from chapter 3 that LQG allows us to compute the optimal policy analytically.

[6] We found that the computation times for both estimators were equivalent.

[7] We only clip the gradient norm at a maximum of 100 to avoid numerical errors.

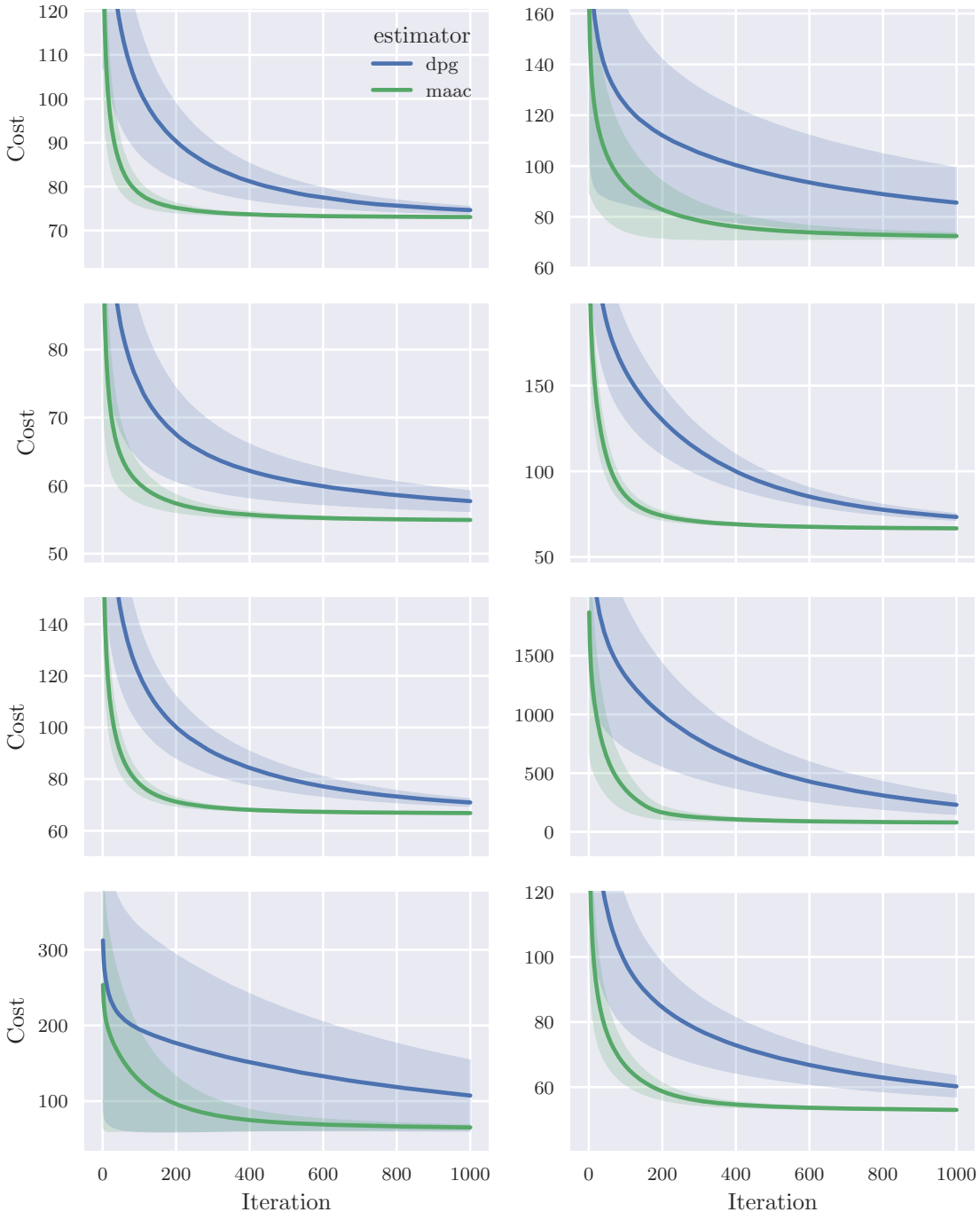

Figure 4.8: *Policy optimization with unnormalized SVG estimation. Each panel corresponds to a different LQG instance (generated via different random seeds). Lines denote the average results and shaded regions, one standard deviation, across 10 runs of the algorithm, each with a different random initial policy. Results obtained with the 8-step versions of each estimator.*

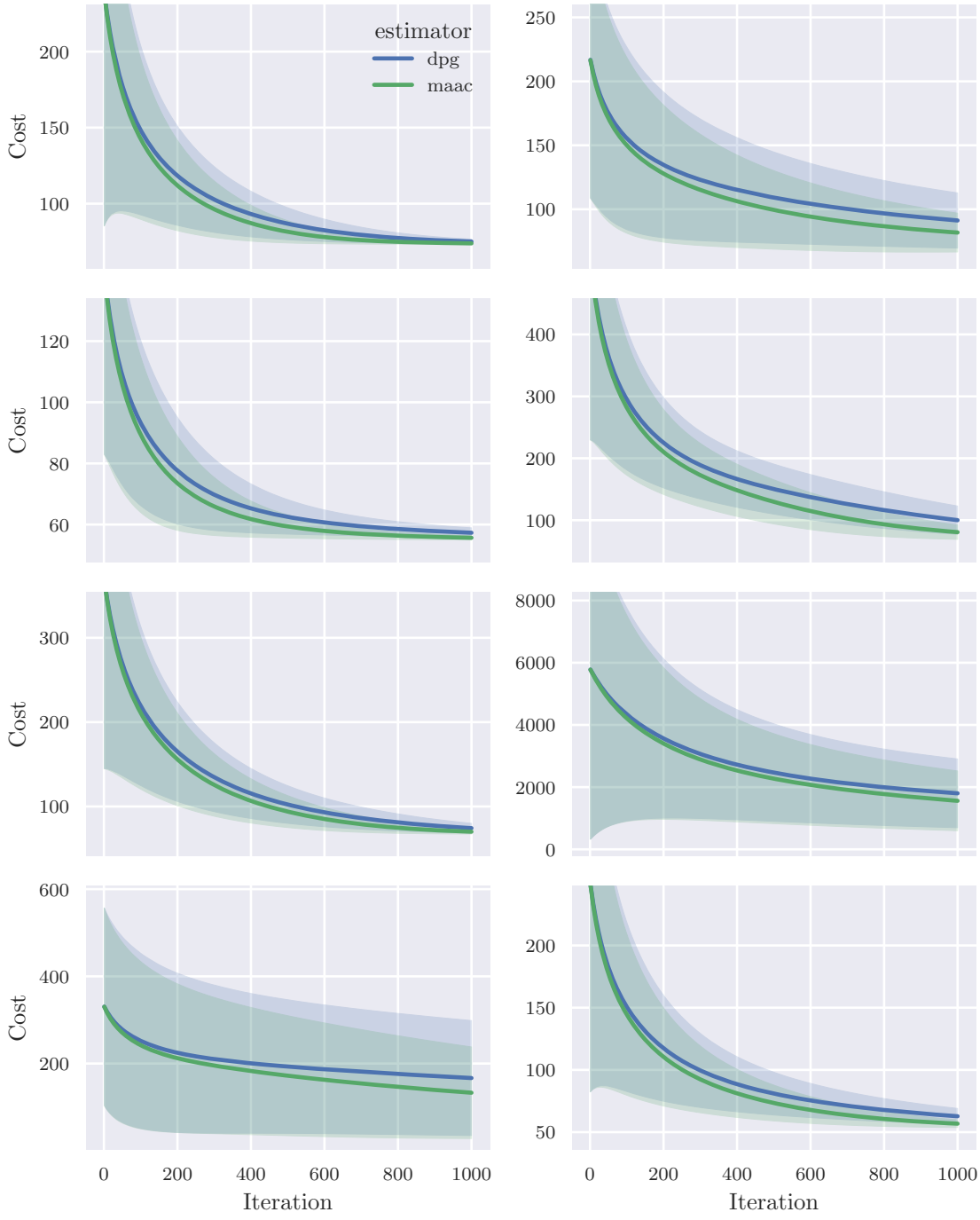

Figure 4.9: *Policy optimization with normalized SVG estimation. Each panel corresponds to a different LQG instance (generated via different random seeds). Lines denote the average results and shaded regions, one standard deviation, across 10 runs of the algorithm, each with a different random initial policy. Results obtained with the 8-step versions of each estimator. Gradients were normalized before being passed to SGD.*

Chapter 5

Model Learning

In this chapter, we analyze how model learning factors in the performance of SVG methods. Again, we use the gradient quality metrics of chapter 4 to inspect if learned models are successful in predicting value gradients. We consider the interplay between the performance in the sub-task of learning a model and the task of predicting accurate gradients, lines 6 and 7 of algorithm 3 respectively. Ultimately, the algorithm's objective is to improve the policy's performance in the environment, so we also consider it in our experiments.

Additionally, we compare the model-based MAAC(K) estimator with the model-free DPG estimator, all with learned models and value-functions. With these experiments, we aim to understand if model-based approaches are more effective than model-free ones that rely on learning a value-function instead of a transition model. Overall, our objectives are: (a) to identify the conditions under which model learning fails; (b) to evaluate different ways of using the model for gradient estimation; and (c) compare the model-based and value-based approaches to estimating value gradients.

5.1 Model-based prediction

In this section, we investigate the following questions:

1. Is log-likelihood loss a good predictor of gradient accuracy?
2. Is value prediction accuracy correlated with gradient prediction accuracy?
3. Does gradient accuracy deteriorate when the model is fitted to off-policy data instead of on-policy data?
4. Is model-based prediction superior to value-based prediction in gradient space?

5.1.1 Model learning in isolation

In this section, we tackle questions 1-3 above. Our environment model parameterizes the mean and (diagonal) covariance of a multivariate Gaussian distribution as linear

functions of the state and action:

$$p_\psi(\mathbf{s}, \mathbf{a}) \doteq \mathcal{N}\left(\mu_\psi(\mathbf{s}, \mathbf{a}), \operatorname{diag}\left(\sigma_\psi(\mathbf{s}, \mathbf{a})\right)\right) , \tag{5.1}$$

$$\mu_\psi(\mathbf{s}, \mathbf{a}) = \mathbf{W}_\mu \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} + \mathbf{b}_\mu , \tag{5.1a}$$

$$\sigma_\psi(\mathbf{s}, \mathbf{a}) = \text{softplus}(\mathbf{b}_\sigma) . \tag{5.1b}$$

Where $\mathbf{W}_\mu \in \mathbb{R}^{n \times (n+d)}$ and $\mathbf{b}_\mu, \mathbf{b}_\sigma \in \mathbb{R}^n$ are the model parameters (denoted by ψ) and $\text{softplus}(x) = \log(1 + \exp(x))$ is an element-wise function ensuring the output is positive.

Fitting the model above to environment data is very close to system identification in Optimal Control, which also uses linear models. However, a key difference between eq. (5.1) and models in system identification is that the latter deterministically map a state-action pair to a next state using a linear function, i.e., $f_\psi(\mathbf{s}, \mathbf{a}) = \mu_\psi(\mathbf{s}, \mathbf{a})$. Linear Least Squares methods are then used to fit these deterministic models to the observed data. We chose eq. (5.1) to match other model parameterizations in SVG algorithms, which are stochastic. Besides, MLE with Gaussian models can be seen as a generalization of Least Squares optimization (Charnes *et al.*, 1976).

Exploration is crucial for model-based prediction

Our first experimental observation is that, surprisingly, learning models exclusively from on-policy data can be detrimental for gradient estimation. Recall from chapter 4 that, ideally, we want to sample starting states for model-based rollouts from the on-policy state distribution. Naturally, then, an approach would be to build a replay buffer of trajectories on-policy and use the same data for both model learning and value gradient estimation for maximum sample-efficiency. This is what we investigate next.

The experimental setup consists of 20 randomly-generated LQGs (following section 3.1 with $\dim(\mathcal{S}) = \dim(\mathcal{A}) = 2$ and $H = 50$). To isolate the model's influence in the results, we perform our experiment with full access to the ground-truth reward function and the ground-truth value function for bootstrapping. For each environment, we perform a model-based prediction run as follows. We collect 2000 trajectories with a randomly generated, stabilizing policy (refer back to section 3.2). Trajectories are split into a training (90%) and a validation (10%) dataset. The training dataset is shuffled before each iteration of model learning (an *epoch*). Model learning fits a dynamics model via MLE: maximizing the likelihood of trajectory segments of length 4 in mini-batches of size 128. After every epoch (implying every datapoint in the training set has been used), we compute the model's performance on the validation set, including: (1) the average negative likelihood of all trajectory segments, or *loss*; (2) the empirical KL divergence between the model and the environment dynamics; (3) the absolute relative error between the real state-value function and its 4-step expansion with the model (from random states in the dataset); and (4) the gradient accuracy of MAAC(4).

Figure 5.1 shows the model's performance metrics during training against the number of epochs. Each line represents one independent run, each on one of the 20 sampled environments. Notice how every single run is successful at improving the model's loss,

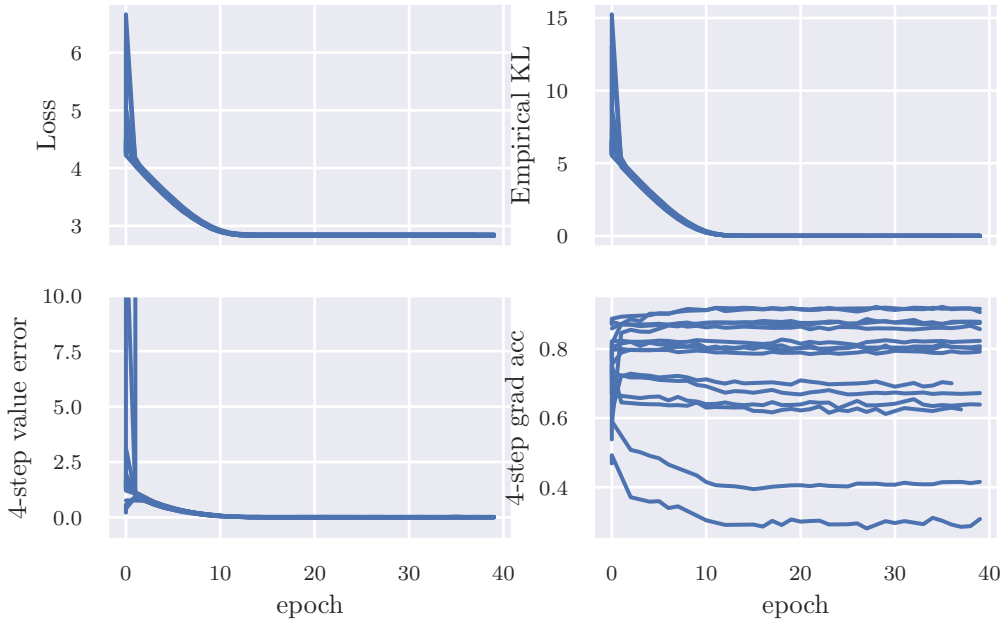

Figure 5.1: *Loss, empirical KL, value error, and gradient accuracy during model learning with on-policy data*

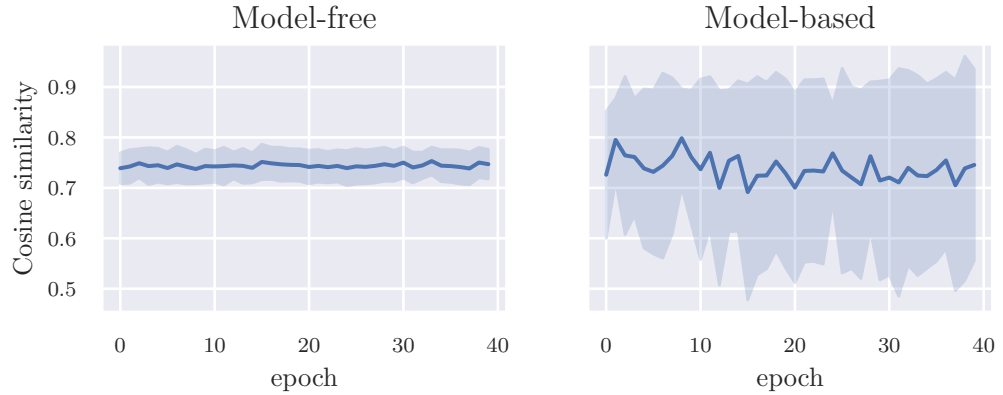

Figure 5.2: *Comparison of gradient accuracy between DPG(0) (model-free) and MAAC(K) with a model fitted to on-policy data.*

empirical KL and 4-step value error, indicating that the model is predicting states correctly and helping induce a good estimate of the value function. However, the gradient accuracy of the MAAC(4) estimates do not always improve with model learning, with some ending up worse than with the initial, randomly initialized model. We also compare the average gradient accuracy of a model-free estimator, MAAC(0) using the ground-truth action-value function, with the model-based MAAC(4) in fig. 5.2. The figure highlights that, on average, the estimators have the same accuracy, but the learned model introduces a lot of variance, which is not desirable. To see why this may be happening, we inspect a single instance of model learning in greater detail in what follows.

Consider the following, simple LQG (randomly generated by the procedure described

in section 3.1):

$$p^*(\mathbf{s}, \mathbf{a}) = \mathcal{N}\left(\begin{bmatrix} 1.0689 & 0.0089 & 0.9776 & 0.9827 \\ 0.0089 & 1.0637 & 0.2107 & -0.1850 \end{bmatrix} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix}, I\right), \tag{5.2}$$

$$R(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \tfrac{1}{2}\mathbf{s}^\intercal \begin{bmatrix} 0.4530 & -0.2178 \\ -0.2178 & 3.5398 \end{bmatrix} \mathbf{s} + \tfrac{1}{2}\mathbf{a}^\intercal \begin{bmatrix} 1.3662 & -0.0420 \\ -0.0420 & 0.7659 \end{bmatrix} \mathbf{a}, \tag{5.3}$$

$$\rho(\mathbf{s}) = \mathcal{N}(\mathbf{0}, I), \tag{5.4}$$

where $\dim(S) = \dim(\mathcal{A}) = 2$ with a horizon of $H = 50$. Policy generation, data collection and model learning are all done as in the experiments of fig. 5.1. We stop fitting the model once its loss on the validation set stops improving for 3 consecutive epochs, a technique known as *early stopping* to avoid overfitting the model to training data. Our model learning run found the following parameters:

$$\mathbf{W}_\mu = \begin{bmatrix} -0.1072 & -0.0347 & -0.0238 & 0.0202 \\ 0.0403 & 0.3481 & -0.2826 & 0.3444 \end{bmatrix},$$
$$\mathbf{b}_\mu = [0.0042, -0.0055]^\intercal,$$
$$\sigma_\psi = [0.9984, 1.0047]^\intercal.$$

We can see that the final model correctly predicts the diagonal of the dynamics covariance matrix and its transition bias is close to real one, zero. On the other hand, the transition kernel $\mathbf{W}_\mu$ is much different than the dynamics' kernel. Despite this, the learned model achieves an empirical KL divergence of 0.0004 (on the validation set) with the environment dynamics, implying that its state predictions are very accurate. Indeed, the relative error between the 4-step state-value expansion using this model and the real state-value function is, on average, 0.01. However, the accuracy of the MAAC(4) estimates is, on average over the validation set, approximately -0.806, meaning that the estimated gradients point in almost the opposite direction of the real value gradient.

Figure 5.3 confirms this observation with a visualization of the optimization surface around the policy parameters. The vertical axis corresponds to the policy's value (computed analytically as in fig. 3.2). Each point (x, y) in the horizontal axes correspond to a policy parameter update of $\theta \leftarrow \theta + x\mathbf{u} + y\mathbf{g}$, where $\mathbf{u}$ is a random direction in the parameter space (sampled uniformly at random from the unit sphere) and $\mathbf{g}$ is the normalized gradient estimated with the model. We can see that updating the policy in the direction of the estimated gradient leads to decreasing total value, further confirming that the model, despite accurate in some metrics, is inadequate for improving the policy.

To explain these observations, we take a closer look at the model's Jacobians. Specifically, we compare the reparameterized model and environment dynamics when both are combined the target policy:

$$f_*^\mu(\mathbf{s}, \xi) = f_*(\mathbf{s}, \mu_\theta(\mathbf{s}), \xi), \tag{5.5}$$

$$f_\psi^\mu(\mathbf{s}, \xi) = f_\psi(\mathbf{s}, \mu_\theta(\mathbf{s}), \xi). \tag{5.6}$$

The noise variable does not affect the Jacobian of these functions, for a given state, since its effect is additive. We sample a random noise variable from a standard Gaussian and a

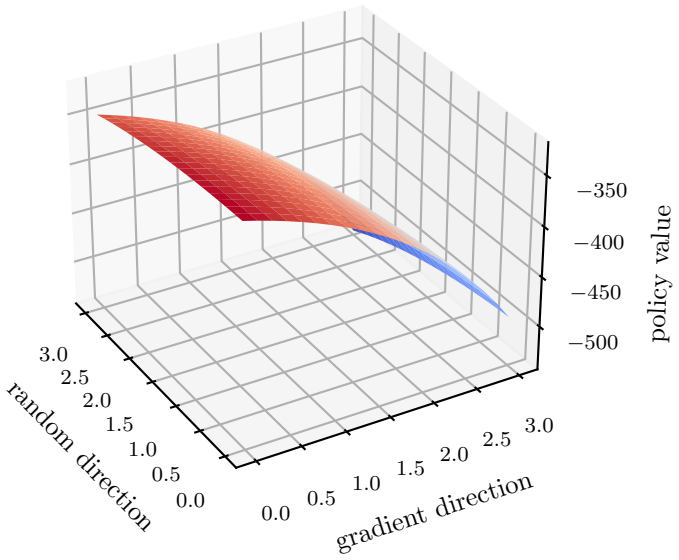

Figure 5.3: *Optimization surface induced by the SVG estimated with a model fitted to on-policy data.*

state from the replay buffer and compute the Jacobians w.r.t. to the state inputs (leveraging the automatic differentiation of PyTorch), arriving at:

$$\nabla_{\mathbf{s}} f^{\mu}_{\star}(\mathbf{s}, \xi) = \begin{bmatrix} -0.1049 & \approx 0 \\ \approx 0 & 0.7852 \end{bmatrix}$$

and

$$\nabla_{\mathbf{s}} f^{\mu}_{\psi}(\mathbf{s}, \xi) = \begin{bmatrix} -0.1057 & -0.0037 \\ -0.0069 & 0.7887 \end{bmatrix} .$$

We also verify that $\|\nabla_{\mathbf{s}} f^{\mu}_{\star}(\mathbf{s}, \xi) - \nabla_{\mathbf{s}} f^{\mu}_{\psi}(\mathbf{s}, \xi)\| = 0.0086$. Thus, we see that the model obtained mimics the on-policy dynamics (see eq. (2.1)) in first-order. Indeed, since we collected data exclusively with the target policy, the model was fed a variety of states, but not a variety of actions for each state (only the action that the policy would take). Therefore, model learning is not able to correctly estimate the dynamics around a particular state and overfits to the on-policy distribution.

We then consider collecting data off-policy, using a stochastic behavior policy, for model learning. Ideally, we would also have access to on-policy data for value gradient estimation, however, that would increase the data requirements of an RL agent, which is precisely the opposite of what MBRL proposes to do. Therefore, there needs to be a consideration of how much "off-policy" our data collection is, so as to not hurt either the model learning or SVG estimation subroutines. Since our target policy is deterministic, we borrow from DPG-style algorithms (Silver *et al.*, 2014; Fujimoto *et al.*, 2018; D'Oro and Jaskowski, 2020) and use a behavior one that adds white noise to the target policy's actions: $\mu_\theta(\mathbf{s}) + \xi$, $\xi \sim \mathcal{N}(\mathbf{0}, 0.3I)$. The rest of the experimental setup is kept the same as used for fig. 5.2.

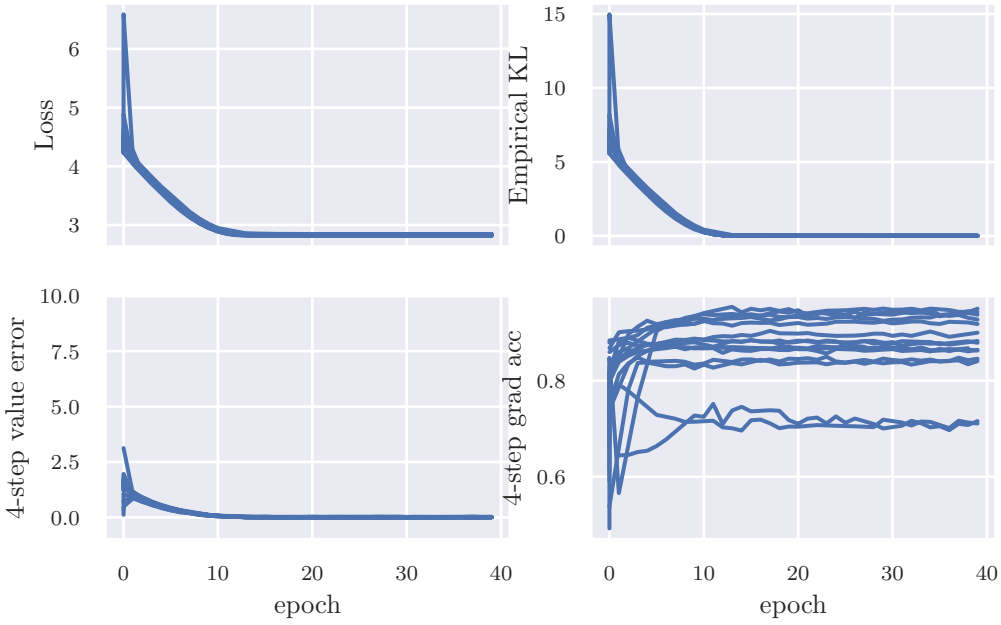

Figure 5.4: *Loss, empirical KL, value error, and gradient accuracy during model learning with off-policy data*

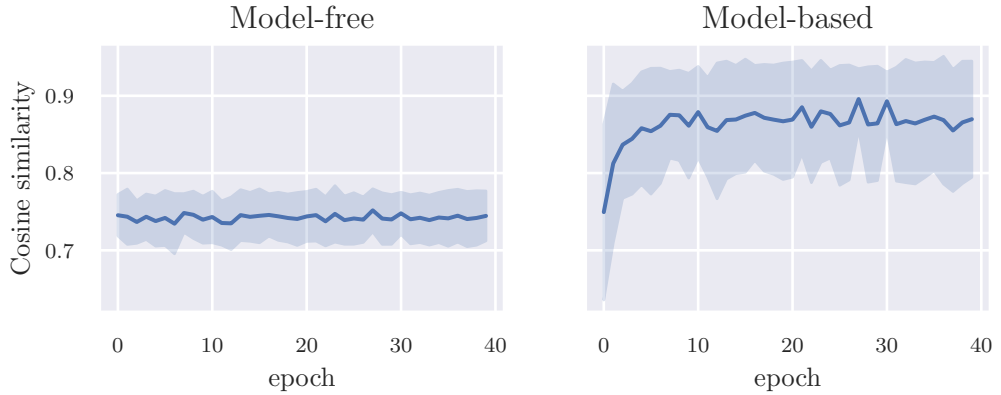

Figure 5.5: *Estimated SVG accuracy resulting from models fitted from on-policy (left) and off-policy (right) data.*

Figure 5.4 shows the performance metrics for the model against the number of training epochs in the off-policy training dataset. Note that now MAAC(4) gradient accuracy also improves with model learning, with the exception of one outlier. We aggregate the value gradient accuracy results in fig. 5.5, contrasting the model-free MAAC(0) with the model-based MAAC(4). With a model trained on off-policy data, one can see a benefit of using a model-based estimator over a model-free one, unlike in fig. 5.2.

Overall, these experiments on the LQG framework help us identify with precision the important issue of data collection for model learning and its impact on the learned model. They also serve as a reminder that the log-likelihood loss is not always a good signal of model adequacy for value gradient estimation, even though it is often the only monitored

metric in model-based RL. Even the idealized value prediction accuracy (since it requires the true value function to be computed) is not a good metric for the model in an SVG algorithm. The challenge for future work is to devise metrics that don't require knowledge of the environment's dynamics and are more predictive of a model's adequacy for value gradient estimation.

Learning reward functions

We also investigate the interplay between exploration and learning the reward function. Our reward model mimics the true one of the environment, ignoring the next state and with no final-state reward:

$$R_\psi(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \frac{1}{2} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix}^\intercal \mathbf{W}_R \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} + \mathbf{w}_R^\intercal \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} . \tag{5.7}$$

The parameters $\psi = \{\mathbf{W}_R, \mathbf{w}_R\}$ are randomly initialized. We use a similar experimental setup as in the previous subsection. Here, however, we optimize the reward model to minimize the Mean Squared Error (MSE) with the observed environment rewards:

$$\mathbb{E}_{\mathbf{s},\mathbf{a},r\sim\mathcal{D}_{\text{train}}} \left[\left(R_\psi(\mathbf{s}, \mathbf{a}) - r \right)^2 \right] . \tag{5.8}$$

On the validation set, we monitor two metrics every half an epoch: the cosine similarity between $\nabla_\mathbf{s} R_\psi(\mathbf{s}, \mathbf{a})$ and $\nabla_\mathbf{s} R(\mathbf{s}, \mathbf{a})$, and the same between $\nabla_\mathbf{a} R_\psi(\mathbf{s}, \mathbf{a})$ and $\nabla_\mathbf{a} R_\psi(\mathbf{s}, \mathbf{a})$.

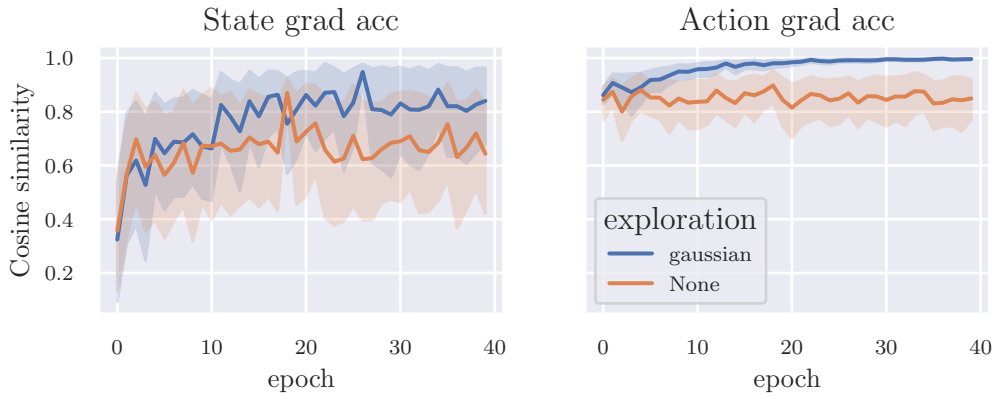

Figure 5.6: *Reward gradient accuracy during reward model learning. Left: accuracy of reward gradient w.r.t. the state input. Right: accuracy of reward gradient w.r.t. the action input.*

Figure 5.6 shows the validation metrics during reward model learning against the number of epochs. We compare learning on on-policy data (exploration = "None") and off-policy data (exploration = "gaussian"). The latter uses a behavior policy like the one used in the experiment for figs. 5.4 and 5.5. We can see that, in general, the learned model predicts the state and action gradients pretty well. Nevertheless, using off-policy data provides a boost to the accuracy of state-gradients and specially to action-gradients, almost matching the real ones.

5.1.2 Model-based vs. value-based prediction

Next, we focus on question 4 of section 5.1: "Is model-based prediction superior to value-based prediction?". This is pertinent to SVG algorithms because learning a value function is a necessary part in an RL setting, where we don't know the true value function for bootstrapping the gradient estimator. Why not, then, simply use the learned value function in the model-free DPG estimator and not have to worry about also fitting dynamics and reward models to the data? To identify the potential shortcomings of this approach, we empirically analyze fitted Q-learning and its effectiveness in finding and accurate action-value model in gradient space.

We choose a quadratic action-value model, similar in form to the true action-value function, defined

$$Q_\phi(\mathbf{s}, \mathbf{a}, t) = \frac{1}{2} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix}^\mathsf{T} \mathbf{W}_{Qt} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} + \mathbf{w}_{Qt}^\mathsf{T} \begin{bmatrix} \mathbf{s} \\ \mathbf{a} \end{bmatrix} + w_{Qt} \,. \tag{5.9}$$

Note that we have separate parameters per timestep and the parameter set $\phi = \{\mathbf{W}_{Qt}, \mathbf{w}_{Qt}, w_{Qt}\}_{t=0}^{H}$. Thus, there is no *approximation error*, i.e., the true action-value function lies within the set of functions representable by this parametric model. We may only suffer from *estimation error*, influenced by sampling and the choice of learning algorithm.

Sampling is done as in the previous prediction experiments: either with the target policy (on-policy) or, a behavior policy that adds white noise to the target policy's actions. The learning algorithm is Fitted Q-Learning (Antos *et al.*, 2008; Lillicrap *et al.*, 2016). Specifically, given a training dataset $\mathcal{D}_{\text{train}}$ of observed environment transitions, we minimize the MSE between the action-value model and the Temporal Difference (TD) target,

$$\mathbb{E}_{\mathbf{s},\mathbf{a},r,\mathbf{s}',t\sim\mathcal{D}_{\text{train}}} \left[\left(Q_\phi(\mathbf{s}, \mathbf{a}, t) - \left(r + Q_\phi(\mathbf{s}', \mu_\theta(\mathbf{s}'), t+1) \right) \right)^2 \right] , \tag{5.10}$$

w.r.t. the parameters ϕ. We optimize eq. (5.10) with minibatch SGD on $\mathcal{D}_{\text{train}}$ and reserve a validation set $\mathcal{D}_{\text{val}}$ to evaluate performance metrics and generalization. The actual learning algorithm incorporates many tricks by Fujimoto *et al.* (2018) (target value functions, Clipped Double Q-learning, target policy smoothing) which deviates from the strict definition of eq. (5.10), but have been shown to stabilize Q-learning.

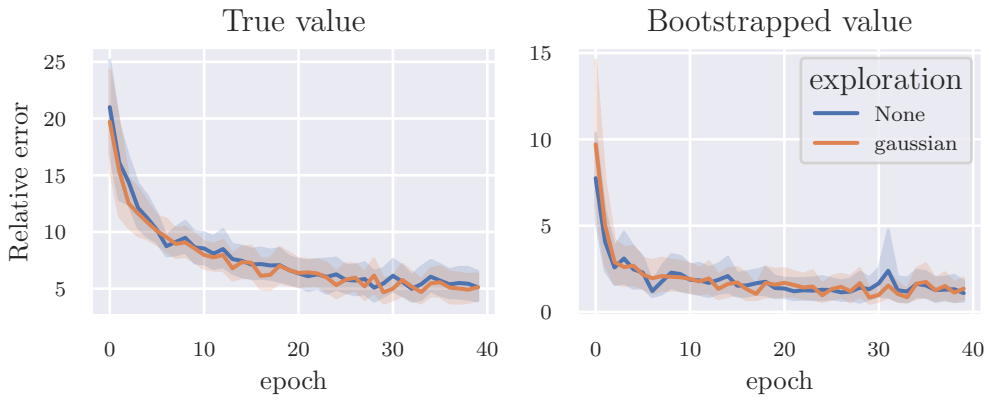

Figure 5.7: *Errors of the approximate action-value function, during training, against the true value function (left) and the bootstrapped TD target (right).*

Figure 5.7 shows the relative error of the learned value function against the true action-value (left) and its 1-step bootstrapped approximation (right), i.e., the TD target. These metrics are computed on the validation set, which consists of 10% of the total 2000 trajectories collected in the environment. We execute 10 independent runs of fitted Q-learning, each on a different environment with a random target policy, for each exploration setting (on-policy of off-policy). Epochs here are the same as in the dynamics model experiments, meaning a full pass through the shuffled training data. We can see that learning improves the action-value predictions over time, slowing down as the value model gets closer to the target.

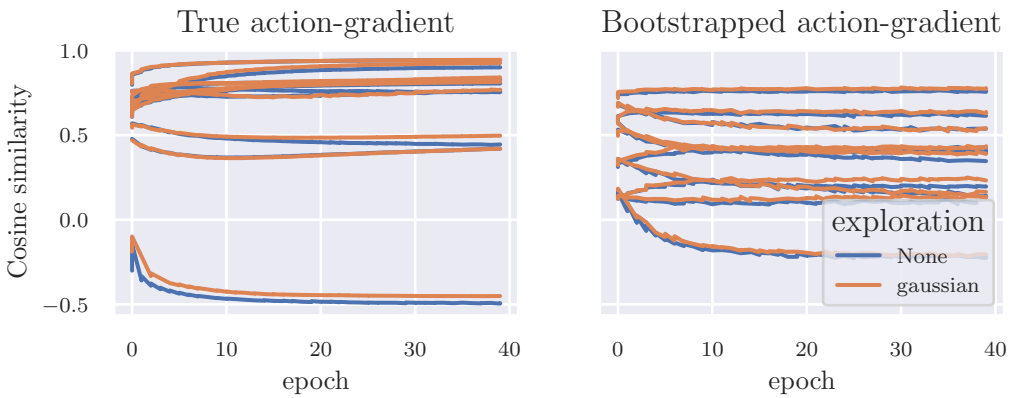

Figure 5.8: *Action-gradient accuracy of the approximate action-value function, during training, against the true action-gradient (left) and the bootstrapped TD target's action-gradient (right).*

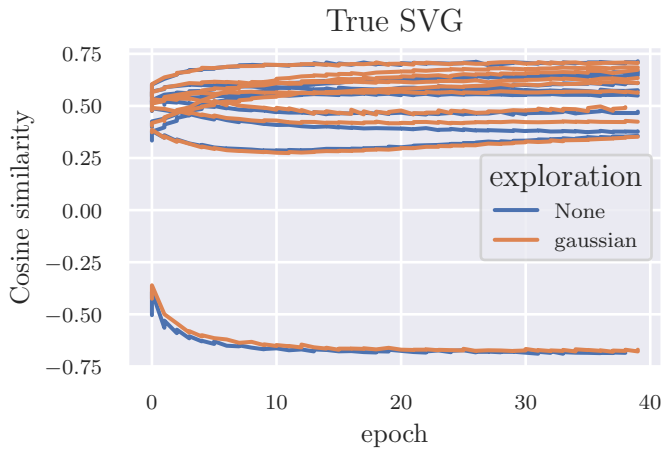

Figure 5.9: *Accuracy of the SVG estimated via DPG(0) using the learned action-value function during training.*

On the other hand, fig. 5.8 (left) shows no obvious improvement in the cosine similarity between $\nabla_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$ and $\nabla_{\mathbf{a}} Q^\mu(\mathbf{s}, \mathbf{a})$ during training. We only end up with and accurate action-gradient if we were lucky to sample good parameters for Q_ϕ before training. This behavior is also unaffected by the data collection being either on-policy or off-policy, unlike the dynamics and reward model approach which at least fare well with off-policy data. If we can't rely on Q-learning to improve action-gradients, then even if we sample a

good initial Q_ϕ, it will become inaccurate as the policy is updated over time. We also plot the cosine similarity between $\nabla_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$ and $\nabla_{\mathbf{a}}(r + Q_\phi(\mathbf{s}', \mu_\theta(\mathbf{s}')))$ in fig. 5.8 (right). It shows that the estimated action-gradient does not improve in accuracy even against the gradient of the TD target used for training. The impact all this on the accuracy of the estimated SVG is shown in fig. 5.9, in which we observe that accuracy does not consistently improve with training.

Overall, these results show that learning an accurate action-value function, specially in gradient space, can be harder that dynamics and reward model learning. Dong *et al.* (2020) made a similar observation about this difficulty, specifically when using neural networks to approximate the Q-function or the dynamics. This may be the reason why model-based approaches that simply use model generated data for augmenting Q-learning (Feinberg *et al.*, 2018; Buckman *et al.*, 2018; Janner *et al.*, 2019b) have not been as successful as SVG methods. Amos, Stanton, *et al.* (2020) observe the same in their experiments and recommend using dynamics models for policy improvement (i.e., value gradient estimation) and not for Q-learning.

However, recent work by D'Oro and Jaskowski (2020) has proposed a new model-based approach to improving Q-learning in gradient space, which could be used in place of or together with model-based value gradient estimation. We analyze this approach in next subsection.

5.1.3 Improving value-based prediction using MAGE

Model-based Action-Gradient-Estimator (MAGE) Policy Optimization is a novel MBRL method by D'Oro and Jaskowski (2020) that aims to improve Q-function action-gradient estimates by leveraging differentiable dynamics and reward models. The main concern raised by the authors is that vanilla fitted Q-learning focuses on obtaining accurate action-value estimates, while policy optimization algorithms that use the DPG estimator need accurate Q-function action gradients. Their proposed approach attempts instead to solve

$$\text{minimize}_\phi \mathop{\mathbb{E}}_{\substack{\mathbf{s},t\sim D_{\text{train}} \\ \mathbf{a}=\mu(\mathbf{s}) \\ \mathbf{s}'\sim p_\psi(\mathbf{s},\mathbf{a})}} \left\| \nabla_{\mathbf{a}} \left[Q_\phi(\mathbf{s}, \mathbf{a}, t) - \left(R_\psi(\mathbf{s}, \mathbf{a}) + Q_\phi(\mathbf{s}', \mu_\theta(\mathbf{s}'), t+1) \right) \right] \right\| . \tag{5.11}$$

In other words, the objective is to minimize the error between the estimated Q-function action gradient (or action-gradient for short) and the action-gradient of the TD target, which stands in for the true action-gradient.

Note that computing the objective function in eq. (5.11) involves differentiating through the sampled state $\mathbf{s}' \sim p_\psi(\mathbf{s}, \mathbf{a})$. Both the original MAGE implementation and ours achieve this by reparameterizing the dynamics model and using the pathwise derivative estimator. Thus, MAGE fits our definition of an SVG method since it uses a model-based 1-step expansion of the value function and computes its gradient using the derivatives of the dynamics model.

In this subsection, we seek to evaluate if MAGE can solve the issues with Q-learning identified in the previous subsection. We use the same quadratic Q-function model, data-

collection and preprocessing procedures, and tricks by Fujimoto *et al.* (2018) for Q-function stabilization (which were also used in the original MAGE paper). Following D'Oro and Jaskowski (2020), we add the penalty term $-\lambda\big(Q_\phi(\mathbf{s}, \mathbf{a}, t) - \big(r + Q_\phi(\mathbf{s}', \mu_\theta(\mathbf{s}'), t+1)\big)\big)^2$ to the objective function of eq. (5.11) as a regularization to avoid degenerate solutions (see Appendix B.1 of D'Oro and Jaskowski (2020)). [1] Furthermore, we use the true dynamics and reward function to isolate MAGE from the effects of model learning.

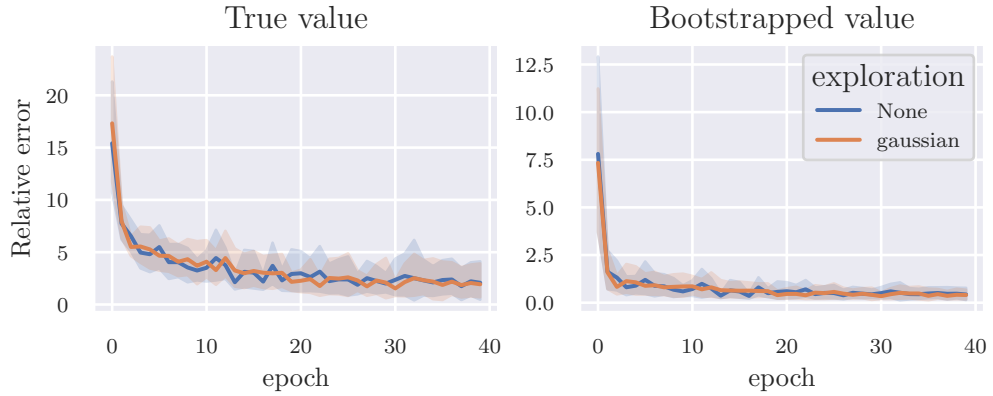

Figure 5.10: *Errors of the approximate action-value function against the true value function (left) and the bootstrapped TD target (right) using MAGE learning.*

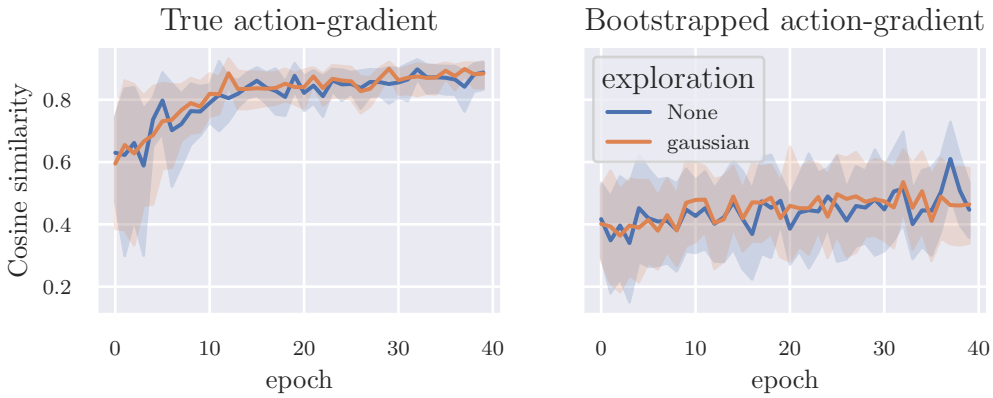

Figure 5.11: *Action-gradient accuracy of the approximate action-value function against the true action-gradient (left) and the TD target's action-gradient (right) using MAGE learning.*

Figure 5.10 shows the relative error of the learned Q-function against the true Q-function (left) and the TD target (right). Similar to fitted Q-learning in fig. 5.7, MAGE improves the action-value predictions over time, likely in part due to the regularization term (the value errors are in fact a little lower than the ones observed in fitted Q-learning). On the other hand, fig. 5.11 (left) shows that MAGE is consistent in improving the action-gradient accuracy against the true action-gradient, unlike fitted Q-learning in fig. 5.8. One interesting observation is that the action-gradient accuracy against the TD target action-gradient, which is the one actually used as a target during training, only slightly improves during training and has high variance, as seen in fig. 5.11 (right).

[1] We use $\lambda = 0.05$ as in the original paper.

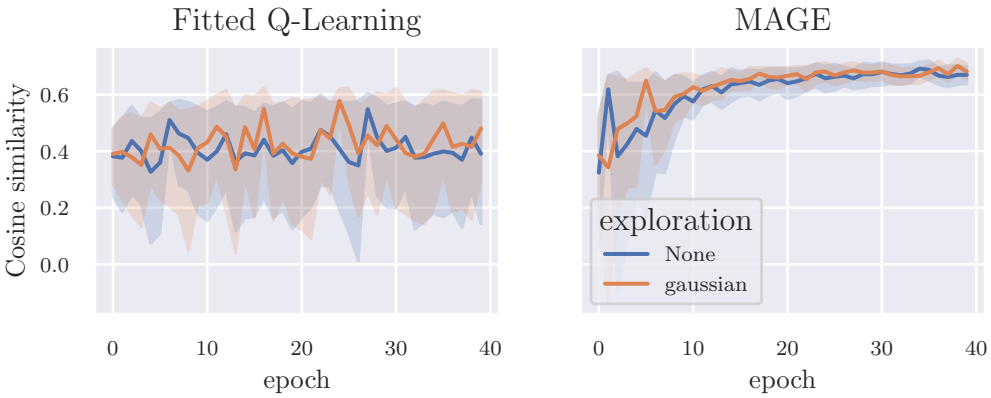

Figure 5.12: *Accuracy of the SVG estimated via DPG(0) using the approximate action-value function obtained via TD(0) (left) and the one obtained via MAGE (right).*

Finally, fig. 5.12 compares the accuracy of SVGs estimated via DPG using the learned Q-function obtained via fitted Q-learning (left) vs. MAGE (right). We can observe that MAGE gives better value gradient estimates and is able to consistently improve them from the starting accuracy induced by randomly generated Q-function models, unlike fitted Q-learning. However, this comes at the cost of introducing dynamics and reward models, which also need to be learned from data.

In this subsection, we did not take into account the interaction between learning environment models and learning value functions. In the next subsection, we wrap up this section's investigation on model-based prediction by combining both and comparing the use of the environment model for MAGE, MAAC or both.

5.1.4 Model-based prediction roundup

We wrap up our experiments on model-based prediction with a comparison of the different approaches discussed so far. This time, we learn environment models and Q-functions from data, specifically 2000 trajectories collected with an exploratory policy $\mu_\theta(\mathbf{s}) + \xi, \xi \sim \mathcal{N}(\mathbf{0}, 0.3I)$. We fit dynamics and reward model to the training data by optimizing the negative log-likelihood loss and MSE loss respectively. We stop training each model when their respective losses stop improving in the validation set for 3 consecutive epochs, a technique known as early stopping. The models are then used in different ways depending on the prediction approach.

The first approach is MAGE, in which we use the models to compute the action-gradient loss of eq. (5.11) to be optimized in the training set. We run SGD on the Q-function with this loss until the relative error between the Q-function and its TD target stops improving in the validation dataset for 3 consecutive epochs. We chose to monitor this metric on the validation set since the action-gradient loss is much more unstable (refer back to fig. 5.11, right). Once traning of the Q-function stops, we use it in the DPG(0) estimator to obtain estimates of the SVG with minibatches of 256 states sampled from the validation set.

The second approach is MAAC, in which we obtain the Q-function by fitted Q-learning as in section 5.1.2. Here we also use early stopping based on the relative error between the

Q-function and its TD target. The Q-function is then used in combination with the learned environment models to compute an SVG estimate using MAAC(4) with minibatches of 256 states sampled from the validation set.

The third approach is MAAC + MAGE, in which we fit a Q-function as in MAGE, but combine it with the environment models to estimate the SVG using MAAC(4). This is a novel combination that, to best of our knowledge, hasn't been tried in previous work. In fact, Amos, Stanton, *et al.* (2020) went so far as to say that there is more risk of failure when using the model for Q-function updates. However, this is likely due to their use regular fitted Q-learning in their experiments, which we saw is faulty in section 5.1.2.

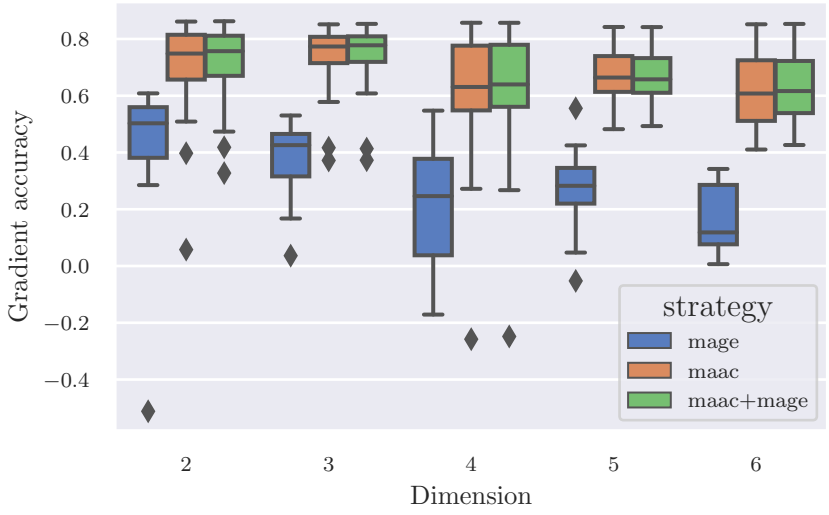

Figure 5.13: *Gradient accuracy distribution of different model-based prediction approaches against environment dimensionality.*

We evaluate each approach across environments of different dimensions, where $\dim(\mathcal{S}) = \dim(\mathcal{A})$. For each dimension size, we sample 20 different environments, run each model-based prediction approach and compute the accuracy of the estimated SVGs Figure 5.13 shows the gradient accuracy distributions as box-plots against the environment dimensions. We observe that MAGE is consistently behind the other two, with its accuracy worsening with increasing environment dimension. This is likely due to its use of the DPG estimator, which we saw is inferior to MAAC(K) in chapter 4. This hypothesis is corroborated by the fact that MAAC+MAGE is as good as MAAC alone, so it is unlikely that Q-function learning is the cause for the underperformance of MAGE. On the other hand, we see that using a Q-function learned via MAGE in MAAC(K) does not provide a significant improvement in gradient accuracy.

This wraps up our investigation on model-based prediction in gradient space. In the next section, we compare these different approaches in a full SVG algorithm for model-based control.

5.2 Model-based control

We finish our exploration of model learning in SVG methods by integrating the approaches analyzed in section 5.1.4 with a full SVG algorithm like algorithm 3.

Our dynamics and reward model learning remain the same, except that the data now comes from a replay buffer generated from the execution of the exploratory policy in the environment over all past iterations of algorithm 3. In the following experiments, we always use the "Gaussian" exploration setting of previous experiments, as it showed to be the superior data collection method for learning environment and Q-function models.

To more closely mimic existing SVG algorithms, e.g., SAC-SVG (Amos, Stanton, *et al.*, 2020), we perform one step of SGD on the Q-function's objective (either the MAGE or the fitted Q-learning one) before each policy improvement step (line 8) of algorithm 3. This is in contrast to the approach in previous sections in which we would fit the Q-function until its relative error against its TD target stopped decreasing on a validation. Most RL algorithms, however, do not have a validation set for the Q-function, a characteristic that seems to have been inherited from early actor-critic algorithms such as Deep Deterministic Policy Gradients (DDPG) (Lillicrap *et al.*, 2016).

We collect results from runs of the algorithm across 20 different environment initializations, each with its respective random initial policy. Before any iterations of algorithm 3, we initialize the replay buffer by collecting 20 trajectories with the initial exploration policy. This is so that we have enough data by the first time model learning starts, an approach commonly used to avoid overfitting models to the small amount of data collected over the first few iterations.

At the end of every iteration of the algorithm, we compute metrics on a randomly sampled batch of transitions from the replay buffer. Figures 5.14 to 5.17 plot these metrics against the number of iterations. Lines denote the median results across the 20 runs and the shaded regions, their 95% confidence interval. All results in these section are in environments with $\dim(\mathcal{S}) = \dim(\mathcal{A}) = 2$ and $H = 100$. Additional results in environments with $\dim(\mathcal{S}) = \dim(\mathcal{A}) = 4$ can be found in appendix B.1, which show similar behavior to the ones in this section.

Figure 5.14 shows the empirical KL divergence between the learned model and the environment dynamics. We see that model learning consistently finds good models, at least in KL divergence terms, regardless of the approach used for Q-function learning or policy improvement. Recall that we're also using an exploratory behavior policy and mixing the experience of the current iteration with ones from past iterations in the replay buffer. Thus, our models should be protected from overfitting to on-policy dynamics as we've seen in section 5.1.1.

Figure 5.15 shows the suboptimality gap of the target policy over the course of the algorithm. Consistent with our observations in section 5.1.4, MAAC and MAAC+MAGE offer the best results, likely due to their superior gradient prediction quality, while MAGE produces the worst results. In fact, MAGE alone leads to policy divergence, worsening the suboptimality gap of the target policy over time. This shows how LQG is not a trivial problem, even for sophisticated RL algorithms like MAGE.

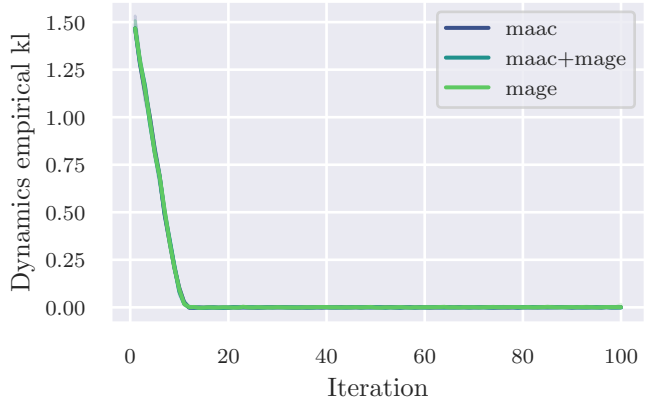

Figure 5.14: *Empirical KL divergence between the learned model and the true dynamics against the number of iterations of algorithm 3.*

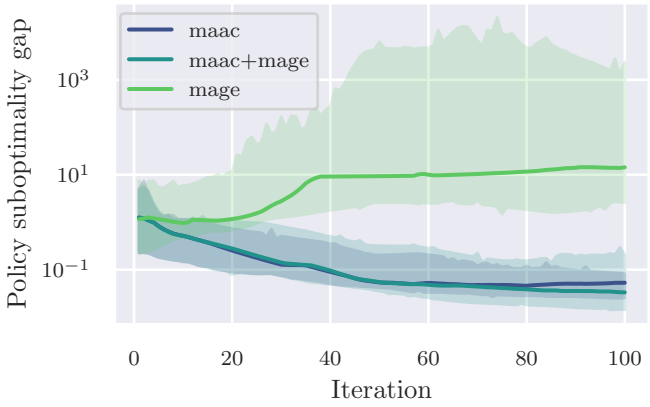

Figure 5.15: *Suboptimality gap of the target policy vs. the number of iterations of algorithm 3.*

Figure 5.16 shows the accuracy of gradient estimates along training for the different approaches. As we can see, MAGE alone produces very poor estimates in the beginning of training, even producing some with negative cosine similarity with the true value gradient, i.e., directions in policy parameter space that are not ascent directions in the objective function. This is likely the reason for the poor suboptimality gap results of MAGE in fig. 5.15. On the other hand, both MAAC and MAAC+MAGE start out with good gradient accuracy but show a gradual decrease in that metric over time. However, this does not seem to be a reflection of the quality of the models, since results with MAAC using perfect models also showed a downward trend in gradient accuracy (see appendix B.2). Instead, this trend seems to be due to the convergence to the optimal policy, which implies lower gradient norms, thus making the gradient estimation task harder (since small errors lead to large deviations in gradient direction).

We also look at metrics for the approximate Q-function during training in fig. 5.17 Recall that the Q-function is learned in tandem with the policy, with every policy improvement (gradient) step preceded by one SGD step on the Q-function's objective, be that from MAGE or regular fitted Q-learning. Interestingly, the quality of the learned Q-function seems to be better when the MAAC estimator is being used and less dependent on the

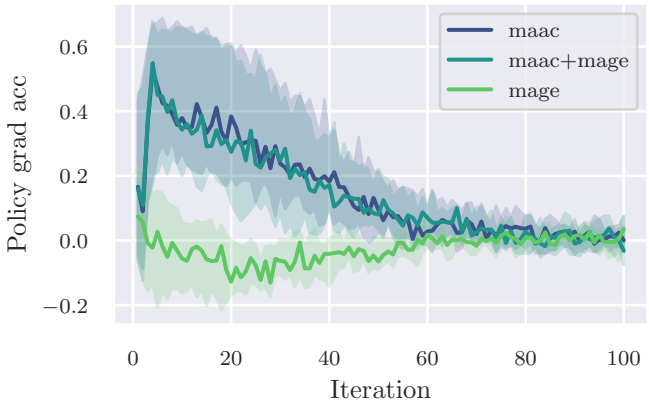

Figure 5.16: *Gradient accuracy of estimated SVG against the number of iterations of algorithm 3*

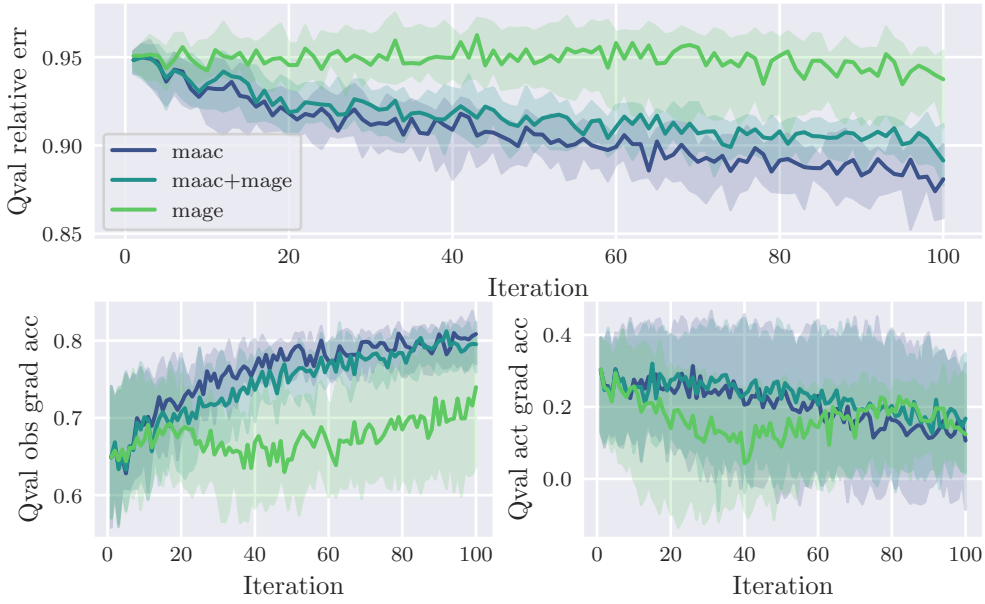

Figure 5.17: *Top: relative error between learned and true Q-functions. Bottom left: state-gradient accuracy of learned Q-function. Bottom right: action-gradient accuracy of learned Q-function.*

use of MAGE. This exposes a complex relation between Q-function learning and policy improvement. Since changing the policy moves the target for the approximate Q-function (by changing the on-policy true Q-function), the quality of the gradient estimator seems to have an indirect impact on the quality of the learned Q-function.

Overall, the results in this section seem to partially corroborate the observation by Amos, Stanton, *et al.* (2020) that there is more risk of failure when using the model for Q-function updates, even through sophisticated methods like MAGE. However, this only seems to be the case when the model is used exclusively for MAGE Q-function updates. In fact, we've shown that one may combine MAGE and MAAC to achieve as good, if not slightly better, results as using MAAC with regular, model-free fitted Q-learning. We've also shown evidence that environment model learning seems to not be affected by

either the Q-function learning or policy improvement steps, while these last two seem interdependent. Future work may dive deeper into the interplay between the choice of gradient estimator and its indirect impact on the performance of Q-function learning. Furthermore, empirical studies can be done on harder environments to evaluate if there are significant benefits of incurring the cost of combining MAAC and MAGE (the latter is significantly more computationally intensive that fitted Q-learning).

Chapter 6

Conclusions & Frontiers

Deep Reinforcement Learning has made great strides in developing autonomous agents that learn to control a system from scratch. Model-based methods, and Stochastic Value Gradient methods in particular, now justify the added complexity and computational cost of learning environment models in tandem with a policy by achieving unprecedented sample efficiency in continuous control benchmarks. However, this added complexity further exacerbates the brittleness of model-free RL. Past studies already showed a lack of understanding by the RL community of how the behavior of RL methods reflects the conceptual framework motivating their development. Furthermore, SVG algorithms are usually evaluated w.r.t. the overall performance of the learned policy, not showing how each component has contributed to such result. As such, these algorithms can fail in unexpected ways, making designing new model-based methods and deploying them in real applications a daunting task.

This thesis proposes a fine-grained analysis of the SVG framework by using Linear Quadratic Gaussian environments, which enables us to compute the ground-truth value gradient and value functions and compare them to the corresponding estimates produced by the RL algorithm. We developed a benchmark to automatically generate random LQG environments that allow for evaluating SVG methods across a variety of scenarios with varying difficulties. Crucially, our implementation supports easy computation of the optimal policy and the ground-truth value, value functions and value gradient for a given (linear deterministic) policy. Although we use these tools to evaluate the components of SVG methods, they are general enough to be useful to researchers studying other RL algorithms.

Our analysis consists of several empirical experiments leveraging our LQG benchmark to inspect the inner workings of SVG methods. We provide a comparison of two broad categories of gradient estimator formulas using perfect models to highlight the impact of sample approximation error. The results show that a bias-variance trade-off occurs between estimators, favoring a biased formula for the value gradient that produces better estimates with fewer samples. We also perform experiments analyzing both model and Q-function learning for the tasks of prediction (estimating the value and gradient for the current policy) and control (iteratively using the estimated gradient to update the target policy). The results show that, overall, value gradient estimation using a Q-function

obtained through fitted Q-learning tends to be more unstable than using a learned model for gradient estimation, as long as the data is not exclusively on-policy. We also show that the model can be used to obtain better Q-function approximators using the Model-based Action-Gradient-Estimator approach. However, this technique is not as effective as using the model directly in the gradient estimation step.

Many open problems remain which relate to and could build on this thesis' work. We describe below some of the frontiers which we consider most relevant, mostly extending the experiments of this thesis.

1. *Environments with high controllability index.* Intuitively, the controllability index of an environment measures the time lag between the execution of an action and the time by which we see an effect in all states. Thus, the higher the index, the less controllable the environment is. Tsiamis and Pappas (2021) showed that system identification (using the least squares algorithm) requires a number of samples bounded by an exponential function of the controllability index. They also showed that there are environments with controllability indexes that grow linearly with the state dimension; a subclass of these are exponentially hard to learn. It is possible to implement the generation of such environments in our benchmark. An interesting study would be to investigate the relationship between the controllability index and the hardness of estimating the value gradient. Given that models can be harder to learn in these environments, a comparison of model-based and value-based methods for value gradient estimation could be interesting.

2. *Model transferability between environments.* We focused on model learning for control of a single environment in this work. However, environment dynamics in the real world can often change over time (weather conditions or upgrades to the hardware of a robot, for example). Furthermore, new tasks can be specified by tweaking the reward function. In such cases, it may be possible to reuse a previously learned model to update the target policy and reduce the amount of data collected for model learning and gradient estimation. Future work may leverage the LQG benchmark to generate perturbations of an environment and evaluate how model accuracy degrades as a function of divergence between environment parameters. A comparison with how a learned value function degrades as the environment changes could give guidance as to which RL methods are better suited to transfer learning.

3. *MAAC and MAGE in non-linear environments.* An obvious limitation of this thesis is the exclusive use of environments with linear dynamics. While LQG environments can be useful to discard less promising RL methods, they don't serve to distinguish more promising approaches. For instance, both MAAC and MAAC+MAGE perform equally well for control in the experiments of section 5.2. However, perhaps a non-linear environment could be challenging enough to show a gap between the two approaches, possibly justifying the added complexity and computational cost of MAGE. On the other hand, doing a fine-grained analysis similar to this thesis in non-linear environments is challenging since general closed-form solutions (like those given by LQG) don't exist for non-linear dynamics. A compromise could be to implement a customizable non-linear environment with known dynamics, like the Industrial Benchmark by Hein *et al.* (2017), using frameworks like PyTorch (Paszke

et al., 2019) to produces estimates of the real value and its gradient using Monte Carlo methods. This benchmark could also serve to evaluate the learning of more complicated, non-linear dynamics models.

4. *Balancing model- and value-based approaches with MAAC(λ).* One could also build on top of this work and explore the balance between model and value function use in the MAAC(K) estimator. Specifically, by varying the number of steps K in the rollout, we can induce more or less reliance on model vs. value function accuracy. Longer rollouts tend to lead compounding model errors and vanishing/exploding gradients, as we saw with MAAC(∞). On the other hand, we saw with MAAC(0) and DPG(0) that only relying on the value function isn't ideal for SVG prediction. A common way to interpolate between these two extremes is to average all rollout lengths with exponentially decaying weights for longer ones. A classical example is TD(λ), where $\lambda \in (0, 1)$ is the parameter that interpolates between using 1-step rollouts ($\lambda \longrightarrow 0$) and full ones with no bootstrapping ($\lambda \longrightarrow 1$) (Sutton and Barto, 2018). This approach has been used in modern algorithms by Schulman, Moritz, *et al.* (2016) to construct estimates of the *advantage function* for actor-critic algorithms using real experience in the environment to construct the K-steps estimates, rather than model-based rollouts. One can similarly construct a MAAC(λ) estimator and use the benchmark developed here to evaluate its behavior as λ varies from 0 to 1. It is possible that with the right choice of λ and a MAGE-learned Q-function, one can achieve better learning stability and higher returns.

Appendix A

LQG derivations

Proof of lemma 2.2.1.
Base case Optimal cost-to-go from timestep N onwards.

$$\begin{aligned} V^\star(\mathbf{s}, N) &= \tfrac{1}{2}\mathbf{s}^\mathsf{T}\mathbf{C}_f\mathbf{s} + \mathbf{c}_f^\mathsf{T}\mathbf{s} \\ &= \tfrac{1}{2}\mathbf{s}^\mathsf{T}\mathbf{V}_N\mathbf{s} + \mathbf{v}_N^\mathsf{T}\mathbf{s} + v_N\,, \end{aligned}$$

for

$$\mathbf{V}_N = \mathbf{C}_f\,, \mathbf{v}_N = \mathbf{c}_f\,, v_N = 0\,.$$

Recursion Let $t < N$ and assume $V^\star(\mathbf{s}, t+1)$ is quadratic with symmetric $\mathbf{V}_{t+1} \in \mathbb{R}^{n\times n}$, $\mathbf{v}_{t+1} \in \mathbb{R}^n$ and $v_{t+1} \in \mathbb{R}$. We may express $Q^\star(\mathbf{s}, t)$ as follows

$$Q^\star(\mathbf{s}, \mathbf{a}, t) = c(\mathbf{s}, \mathbf{a}) + \mathbb{E}\left[\tfrac{1}{2}f(\mathbf{s}, \mathbf{a}, \mathbf{w}_t)^\mathsf{T}\mathbf{V}_{t+1}f(\mathbf{s}, \mathbf{a}, \mathbf{w}_t) + \mathbf{v}_{t+1}^\mathsf{T}f(\mathbf{s}, \mathbf{a}, \mathbf{w}_t) + v_{t+1}\right]. \tag{A.1}$$

Let $\tau = [\mathbf{s}^\mathsf{T}\ \mathbf{a}^\mathsf{T}]^\mathsf{T}$. Expanding the expectation in eq. (A.1),

$$\begin{aligned} &\mathbb{E}\left[\tfrac{1}{2}(\mathbf{F}\tau + \mathbf{f} + \mathbf{w}_t)^\mathsf{T}\mathbf{V}_{t+1}(\mathbf{F}\tau + \mathbf{f} + \mathbf{w}_t) + \mathbf{v}_{t+1}^\mathsf{T}(\mathbf{F}\tau + \mathbf{f} + \mathbf{w}_t) + v_{t+1}\right] \\ &= \tfrac{1}{2}(\mathbf{F}\tau + \mathbf{f})^\mathsf{T}\mathbf{V}_{t+1}(\mathbf{F}\tau + \mathbf{f}) + \tfrac{1}{2}\,\mathbb{E}\left[\mathbf{w}_t^\mathsf{T}\mathbf{V}_{t+1}\mathbf{w}_t\right] + \mathbf{v}_{t+1}^\mathsf{T}\mathbf{F}\tau + \mathbf{v}_{t+1}^\mathsf{T}\mathbf{f} + v_{t+1} \\ &= \tfrac{1}{2}\tau^\mathsf{T}\mathbf{F}^\mathsf{T}\mathbf{V}_{t+1}\mathbf{F}\tau + \left(\mathbf{f}^\mathsf{T}\mathbf{V}_{t+1}\mathbf{F} + \mathbf{v}_{t+1}^\mathsf{T}\mathbf{F}\right)\tau + \underbrace{\tfrac{1}{2}\operatorname{Tr}(\Sigma\mathbf{V}_{t+1}) + \tfrac{1}{2}\mathbf{f}^\mathsf{T}\mathbf{V}_{t+1}\mathbf{f} + \mathbf{v}_{t+1}^\mathsf{T}\mathbf{f} + v_{t+1}}_{q_t}\,. \end{aligned}$$

Where in the first equality above we used that $\mathbb{E}[\mathbf{w}_t] = \mathbf{0}$ and the second equality follows from the cyclic property of the Tr operator and that $\mathbb{E}[\mathbf{w}_t\mathbf{w}_t^\mathsf{T}] = \Sigma$. Substituting the result above in eq. (A.1) and expanding the definition of $c(\mathbf{s}, \mathbf{a})$,

$$Q^\star(\mathbf{s}, \mathbf{a}, t) = \tfrac{1}{2}\tau^\mathsf{T}\underbrace{\left(\mathbf{C} + \mathbf{F}^\mathsf{T}\mathbf{V}_{t+1}\mathbf{F}\right)}_{\mathbf{Q}_t}\tau + \underbrace{\left(\mathbf{F}^\mathsf{T}\mathbf{V}_{t+1}\mathbf{f} + \mathbf{F}^\mathsf{T}\mathbf{v}_{t+1} + \mathbf{c}\right)^\mathsf{T}}_{\mathbf{q}_t}\tau + q_t\,. \tag{A.2}$$

Taking the gradient of eq. (A.2) w.r.t. the action and setting it to zero,

$$\mathbf{Q}_{\mathbf{as}}\mathbf{s} + \mathbf{Q}_{\mathbf{aa}}\mathbf{a} + \mathbf{q}_{\mathbf{a}} = 0 \Longrightarrow \mathbf{a} = \underbrace{-\mathbf{Q}_{\mathbf{aa}}{}^{-1}\mathbf{Q}_{\mathbf{as}}}_{\mathbf{K}_t}\,\mathbf{s} + \underbrace{\left(-\mathbf{Q}_{\mathbf{aa}}{}^{-1}\mathbf{q}_{\mathbf{a}}\right)}_{\mathbf{k}_t}.$$

Finally, substituting $\mathbf{K}_t\mathbf{s} + \mathbf{k}_t$ for $\mathbf{a}$ in eq. (A.2),

$$\begin{aligned} V^\star(\mathbf{s}, t) = \tfrac{1}{2}\mathbf{s}^\intercal \underbrace{\left(\mathbf{Q}_{\mathbf{ss}_t} + \mathbf{Q}_{\mathbf{sa}_t}\mathbf{K}_t + \mathbf{K}_t^\intercal\mathbf{Q}_{\mathbf{aa}}\mathbf{K}_t\right)}_{\mathbf{V}_t}\mathbf{s} \\ + \underbrace{\left(\mathbf{Q}_{\mathbf{sa}_t}\mathbf{k}_t + \mathbf{K}_t^\intercal\mathbf{Q}_{\mathbf{aa}_t}\mathbf{k}_t + \mathbf{q}_{\mathbf{s}_t} + \mathbf{K}_t^\intercal\mathbf{q}_{\mathbf{a}_t}\right)}_{\mathbf{v}_t}\mathbf{s} \\ + \underbrace{\tfrac{1}{2}\mathbf{k}_t^\intercal\mathbf{Q}_{\mathbf{aa}_t}\mathbf{k}_t + \mathbf{q}_{\mathbf{a}_t}{}^\intercal\mathbf{k}_t + q_t}_{v_t}. \end{aligned} \tag{A.3}$$

□

Appendix B

Extra model-based control results

B.1 More state and action variables

Figures B.1 to B.4 show metrics from runs of algorithm 3 across 20 different environments with $\dim(\mathcal{S}) = \dim(\mathcal{A}) = 4$ and $H = 100$. Lines denote the median results and shaded regions, their 95% confidence interval.

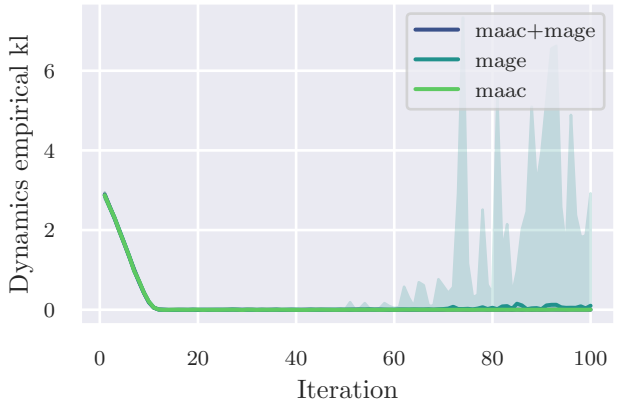

Figure B.1: *Empirical KL divergence between the learned model and the true dynamics against the number of iterations of algorithm 3.*

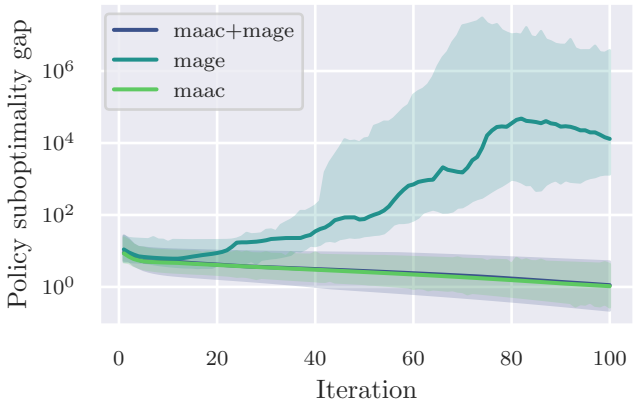

Figure B.2: *Suboptimality gap of the target policy vs. the number of iterations of algorithm 3.*

Figure B.3: *Gradient accuracy of estimated SVG vs. the number of iterations of algorithm 3*

B.2 Model-based control with perfect models

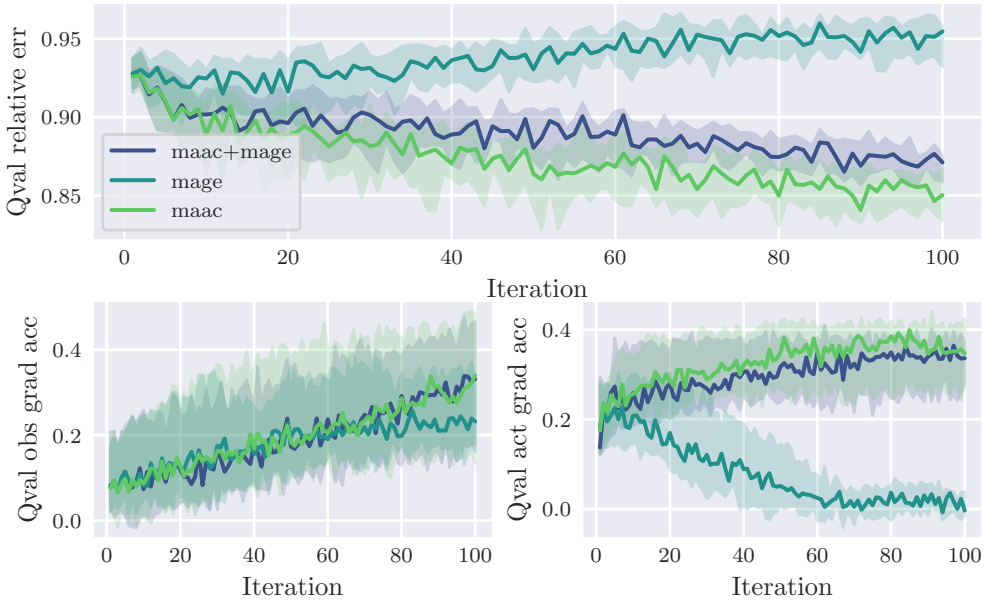

Figure B.4: *Top: relative error between learned and true Q-functions. Bottom left: state-gradient accuracy of learned Q-function. Bottom right: action-gradient accuracy of learned Q-function.*

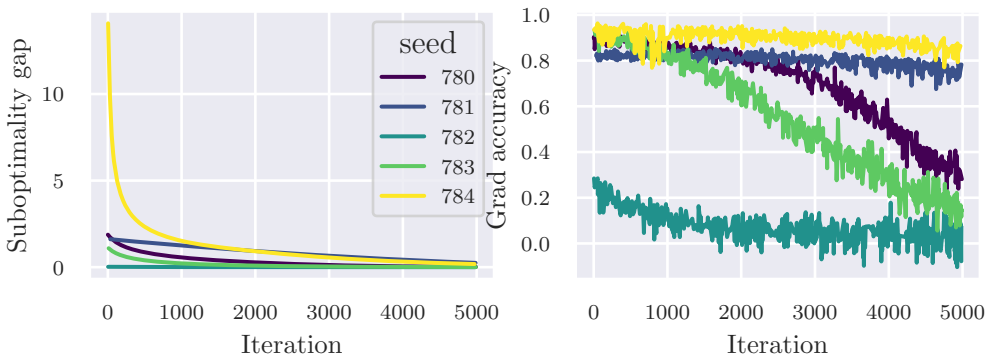

Figure B.5: *Metrics of model-based control with MAAC with perfect models (no learning). Suboptimality of target policy (left) and gradient accuracy (right) vs. number of policy improvement iterations.*

References

[Amos, Rodriguez, *et al.* 2018] Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J. Zico Kolter. "Differentiable MPC for End-to-end Planning and Control". In: *NeurIPS*. 2018, pp. 8299–8310 (cit. on p. 17).

[Amos, Stanton, *et al.* 2020] Brandon Amos, Samuel Stanton, Denis Yarats, and Andrew Gordon Wilson. "On the model-based stochastic value gradient for continuous reinforcement learning". In: *CoRR* abs/2008.1 (2020) (cit. on pp. 3, 7, 17, 33, 34, 39, 52, 55, 56, 58).

[Antos *et al.* 2008] András Antos, Csaba Szepesvári, and Rémi Munos. "Fitted Q-iteration in continuous action-space MDPs". In: *Advances in Neural Information Processing Systems 20*. Ed. by J C Platt, D Koller, Y Singer, and S T Roweis. Curran Associates, Inc., 2008, pp. 9–16. url: http://papers.nips.cc/paper/3233-fitted-q-iteration-in-continuous-action-space-mdps.pdf (cit. on p. 50).

[Asadi *et al.* 2018] Kavosh Asadi, Evan Cater, Dipendra Misra, and Michael L. Littman. "Equivalence Between Wasserstein and Value-Aware Loss for Model-based Reinforcement Learning". In: *CoRR* (June 2018). arXiv: 1806.01265 (cit. on p. 17).

[Brunton and Kutz 2019] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. doi: 10.1017/9781108380690 (cit. on p. 22).

[Buckman *et al.* 2018] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. "Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion". In: (2018), pp. 8224–8234. url: http://papers.nips.cc/paper/8044-sample-efficient-reinforcement-learning-with-stochastic-ensemble-value-expansion%20http://arxiv.org/abs/1807.01675 (cit. on p. 52).

[Byravan *et al.* 2019] Arunkumar Byravan *et al.* "Imagined value gradients: model-based policy optimization with tranferable latent dynamics models". In: *CoRL*. Vol. 100. Proceedings of Machine Learning Research. PMLR, 2019, pp. 566–589 (cit. on p. 3).

[Chan *et al.* 2020] Stephanie C Y Chan, Samuel Fishman, Anoop Korattikara, John Canny, and Sergio Guadarrama. "Measuring the Reliability of Reinforcement Learning Algorithms". In: *ICLR*. OpenReview.net, 2020 (cit. on p. 4).

[Charnes *et al.* 1976] A Charnes, E L Frome, and P L Yu. "The Equivalence of Generalized Least Squares and Maximum Likelihood Estimates in the Exponential Family". In: *Journal of the American Statistical Association* 71.353 (1976), pp. 169–171. doi: 10.1080/01621459.1976.10481508. url: https://www.tandfonline.com/doi/abs/10.1080/01621459.1976.10481508 (cit. on p. 44).

[Chen and Jiang 2019] Jinglin Chen and Nan Jiang. "Information-Theoretic Considerations in Batch Reinforcement Learning". In: *36th International Conference on Machine Learning, ICML 2019* 2019-June (May 2019), pp. 1792–1817. url: http://arxiv.org/abs/1905.00360 (cit. on p. 16).

[Chua *et al.* 2018] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *NeurIPS*. 2018, pp. 4759–4770 (cit. on p. 16).

[Clavera *et al.* 2020] Ignasi Clavera, Yao Fu, and Pieter Abbeel. "Model-Augmented Actor-Critic: Backpropagating through Paths". In: *ICLR*. OpenReview.net, 2020 (cit. on pp. 3, 33, 34, 38, 39).

[D'Oro and Jaskowski 2020] Pierluca D'Oro and Wojciech Jaskowski. "How to Learn a Useful Critic? Model-based Action-Gradient-Estimator Policy Optimization". In: *NeurIPS*. 2020 (cit. on pp. 7, 19, 34, 47, 52, 53).

[D'Oro, Metelli, *et al.* 2019] Pierluca D'Oro, Alberto Maria Metelli, Andrea Tirinzoni, Matteo Papini, and Marcello Restelli. "Gradient-Aware Model-based Policy Search". In: *CoRR* (Sept. 2019). arXiv: 1909.04115. url: http://arxiv.org/abs/1909.04115 (cit. on p. 17).

[Dayawansa 2001] Wijesuriya P. Dayawansa. "Mathematical control theory: deterministic finite dimensional systems [2nd edition] [book review]". In: *IEEE Trans. Autom. Control.* 46.4 (2001), pp. 673–675 (cit. on p. 23).

[Deisenroth and Rasmussen 2011] Marc Peter Deisenroth and Carl Edward Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*. Ed. by Lise Getoor and Tobias Scheffer. Omnipress, 2011, pp. 465–472. url: https://icml.cc/2011/papers/323_icmlpaper.pdf (cit. on p. 3).

[Dong *et al.* 2020] Kefan Dong, Yuping Luo, Tianhe Yu, Chelsea Finn, and Tengyu Ma. "On the expressivity of neural networks for deep reinforcement learning". In: *ICML*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 2627–2637 (cit. on p. 52).

[Donti *et al.* 2017] Priya L. Donti, J. Zico Kolter, and Brandon Amos. "Task-based end-to-end model learning in stochastic optimization". In: *NIPS*. 2017, pp. 5484–5494 (cit. on p. 17).

[Engstrom *et al.* 2020] Logan Engstrom *et al.* "Implementation Matters in Deep RL: A Case Study on PPO and TRPO". In: *ICLR*. OpenReview.net, 2020. url: https://github.com/implementation-matters/code-for-paper (cit. on p. 4).

[Fairbank and Alonso 2012] Michael Fairbank and Eduardo Alonso. "Value-gradient learning". In: *IJCNN*. IEEE, 2012, pp. 1–8 (cit. on pp. 19, 34).

[A. M. Farahmand *et al.* 2017] Amir Massoud Farahmand, André Barreto, and Daniel Nikovski. "Value-Aware Loss Function for Model-based Reinforcement Learning". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*. Ed. by Aarti Singh and Xiaojin (Jerry) Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1486–1494 (cit. on p. 17).

[A.-m. Farahmand 2018] Amir-massoud Farahmand. "Iterative Value-Aware Model Learning". In: *NeurIPS*. 2018, pp. 9090–9101. url: http://papers.nips.cc/paper/8121-iterative-value-aware-model-learning (cit. on pp. 9, 17).

[Feinberg *et al.* 2018] Vladimir Feinberg *et al.* "Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning". In: (Feb. 2018). url: http://arxiv.org/abs/1803.00101 (cit. on p. 52).

[François-Lavet *et al.* 2018] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends in Machine Learning* 11.3-4 (2018), pp. 219–354. doi: 10.1561/2200000071 (cit. on pp. 1, 2).

[Fujimoto *et al.* 2018] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1582–1591. url: http://proceedings.mlr.press/v80/fujimoto18a.html (cit. on pp. 47, 50, 53).

[Gibbs and Su 2002] Alison L Gibbs and Francis Edward Su. "On Choosing and Bounding Probability Metrics". In: *International Statistical Review* 70.3 (2002), pp. 419–435. doi: 10.1111/j.1751-5823.2002.tb00178.x. url: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1751-5823.2002.tb00178.x (cit. on p. 17).

[I. Goodfellow *et al.* 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 5).

[I. J. Goodfellow *et al.* 2016] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016 (cit. on p. 33).

[Hafner *et al.* 2020] Danijar Hafner, Timothy P Lillicrap, Jimmy Ba, and Mohammad Norouzi. "Dream to Control: Learning Behaviors by Latent Imagination". In: *ICLR*. OpenReview.net, 2020 (cit. on pp. 3, 33, 34, 39).

[Heess *et al.* 2015] Nicolas Heess *et al.* "Learning Continuous Control Policies by Stochastic Value Gradients". In: *NIPS*. 2015, pp. 2944–2952. url: http://papers.nips.cc/paper/5796-learning-continuous-control-policies-by-stochastic-value-gradients (cit. on pp. 3, 16, 31).

[Hein *et al.* 2017] Daniel Hein *et al.* "A benchmark environment motivated by industrial control problems". In: *SSCI*. IEEE, 2017, pp. 1–8. isbn: 9781538627259. doi: 10.1109/SSCI.2017.8280935. url: http://ieeexplore.ieee.org/document/8280935/ (cit. on p. 62).

[Henderson *et al.* 2018] Peter Henderson *et al.* "Deep reinforcement learning that matters". In: *AAAI*. AAAI Press, 2018, pp. 3207–3214 (cit. on p. 4).

[Hessel *et al.* 2019] Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. "On Inductive Biases in Deep Reinforcement Learning". In: (July 2019). url: http://arxiv.org/abs/1907.02908 (cit. on p. 6).

[Hochreiter 1998] Sepp Hochreiter. "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems* 6.2 (Nov. 1998), pp. 107–116. issn: 02184885. doi: 10.1142/S0218488598000094 (cit. on p. 32).

[Ilyas *et al.* 2020] Andrew Ilyas *et al.* "A Closer Look at Deep Policy Gradients". In: *ICLR*. OpenReview.net, 2020 (cit. on pp. 4, 29, 35, 37).

[Islam *et al.* 2017] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control". In: *CoRR* abs/1708.04133 (2017) (cit. on p. 4).

[Janner *et al.* 2019a] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. "When to trust your model: model-based policy optimization". In: *NeurIPS*. 2019, pp. 12498–12509 (cit. on p. 16).

[Janner *et al.* 2019b] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. "When to Trust Your Model: Model-Based Policy Optimization". In: *NeurIPS*. 2019, pp. 12498–12509 (cit. on pp. 32, 52).

[Kautsky *et al.* 1985] J Kautsky, N K Nichols, and P V A N Dooren. "Robust pole assignment in linear state feedback". In: *International Journal of Control* 41.5 (1985), pp. 1129–1155. doi: 10.1080/0020718508961188. url: https://doi.org/10.1080/0020718508961188 (cit. on p. 25).

[Le *et al.* 2019] Hoang M. Le, Cameron Voloshin, and Yisong Yue. "Batch Policy Learning under Constraints". In: *36th International Conference on Machine Learning, ICML 2019* 2019-June (Mar. 2019), pp. 6589–6600. url: http://arxiv.org/abs/1903.08738 (cit. on p. 16).

[Levine and Abbeel 2014] Sergey Levine and Pieter Abbeel. "Learning neural network policies with guided policy search under unknown dynamics". In: *NIPS*. 2014, pp. 1071–1079 (cit. on p. 16).

[Lillicrap *et al.* 2016] Timothy P Lillicrap *et al.* "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. url: http://arxiv.org/abs/1509.02971 (cit. on pp. 50, 56).

[Liu *et al.* 2021] Zhuang Liu, Xuanlin Li, Bingyi Kang, and Trevor Darrell. "Regularization Matters for Policy Optimization - An Empirical Study on Continuous Control". In: *International Conference on Learning Representations* (2021). url: https://github.com/xuanlinli17/iclr2021_rlreg (cit. on p. 4).

[Ângelo G. Lovatto *et al.* 2020] Ângelo G. Lovatto, Thiago P. Bueno, Denis D. Mauá, and Leliane N. de Barros. "Decision-aware model learning for actor-critic methods: when theory does not meet practice". In: *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops*. Vol. 137. Proceedings of Machine Learning Research. PMLR, Dec. 2020, pp. 76–86. url: http://proceedings.mlr.press/v137/lovatto20a.html (cit. on p. 4).

[Ângelo Gregório Lovatto *et al.* 2021] Ângelo Gregório Lovatto, Thiago Pereira Bueno, and Leliane Nunes de Barros. "Gradient estimation in model-based reinforcement learning: a study on linear quadratic environments". In: *Intelligent Systems*. Ed. by André Britto and Karina Valdivia Delgado. Cham: Springer International Publishing, 2021, pp. 33–47. isbn: 978-3-030-91702-9 (cit. on p. 6).

[Mnih *et al.* 2015] Volodymyr Mnih *et al.* "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533 (cit. on p. 15).

[Moerland *et al.* 2020] Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. "Model-based Reinforcement Learning: A Survey". In: *Proceedings of the International Conference on Electronic Business (ICEB)* 2018-Decem (June 2020), pp. 421–429. url: http://arxiv.org/abs/2006.16712 (cit. on p. 2).

[Mohamed *et al.* 2020] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. "Monte Carlo Gradient Estimation in Machine Learning". In: *J. Mach. Learn. Res.* 21 (2020), 132:1–132:62 (cit. on p. 31).

[Myung 2003] In Jae Myung. "Tutorial on maximum likelihood estimation". In: *Journal of Mathematical Psychology* 47.1 (Feb. 2003), pp. 90–100. issn: 00222496. doi: 10.1016/S0022-2496(02)00028-7 (cit. on p. 17).

[Pardo *et al.* 2018] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. "Time Limits in Reinforcement Learning". In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 4042–4051 (cit. on p. 10).

[Paszke *et al.* 2019] Adam Paszke *et al.* "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on pp. 27, 62).

[Polydoros and Nalpantidis 2017] Athanasios S Polydoros and Lazaros Nalpantidis. "Survey of Model-Based Reinforcement Learning: Applications on Robotics". In: *Journal of Intelligent and Robotic Systems* 86.2 (2017), pp. 153–173. DOI: 10.1007/s10846-017-0468-y. URL: https://doi.org/10.1007/s10846-017-0468-y (cit. on p. 2).

[Rajeswaran *et al.* 2017] Aravind Rajeswaran, Kendall Lowrey, Emanuel V Todorov, and Sham M Kakade. "Towards Generalization and Simplicity in Continuous Control". In: *Advances in Neural Information Processing Systems 30*. Ed. by I Guyon *et al.* Curran Associates, Inc., 2017, pp. 6550–6561. URL: http://papers.nips.cc/paper/7233-towards-generalization-and-simplicity-in-continuous-control.pdf (cit. on p. 25).

[Recht 2019] Benjamin Recht. "A Tour of Reinforcement Learning: The View from Continuous Control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 2.1 (May 2019), pp. 253–279. ISSN: 2573-5144. DOI: 10.1146/annurev-control-053018-023825. URL: http://arxiv.org/abs/1806.09460 (cit. on pp. 6, 22).

[Ruder 2016] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016) (cit. on pp. 15, 16).

[Santamaría *et al.* 1997] Juan Carlos Santamaría, Richard S Sutton, and Ashwin Ram. "Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces". In: *Adapt. Behav.* 6.2 (1997), pp. 163–217 (cit. on p. 5).

[Schulman, Heess, *et al.* 2015a] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. "Gradient estimation using stochastic computation graphs". In: *NIPS*. 2015, pp. 3528–3536 (cit. on pp. 2, 31).

[Schulman, Heess, *et al.* 2015b] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. "Gradient Estimation Using Stochastic Computation Graphs". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes, Neil D Lawrence, Daniel D Lee, Masashi Sugiyama, and Roman Garnett. 2015, pp. 3528–3536. URL: http://papers.nips.cc/paper/5899-gradient-estimation-using-stochastic-computation-graphs (cit. on pp. 18, 19).

[Schulman, Moritz, *et al.* 2016] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. "High-dimensional continuous control using generalized advantage estimation". In: *ICLR (Poster)*. 2016 (cit. on p. 63).

[Silver *et al.* 2014] David Silver, Guy Lever, Deepmind Technologies, G U Y Lever, and U C L Ac. "Deterministic Policy Gradient (DPG)". In: *Proceedings of the 31st International Conference on Machine Learning* 32.1 (Jan. 2014), pp. 387–395. issn: 1938-7228. url: http://proceedings.mlr.press/v32/silver14.html (cit. on pp. 6, 33, 47).

[Sutton and Barto 2018] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. url: http://incompleteideas.net/book/the-book-2nd.html (cit. on pp. 1, 63).

[Szepesvári 2010] Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010. doi: 10.2200/S00268ED1V01Y201005AIM009. url: https://doi.org/10.2200/S00268ED1V01Y201005AIM009 (cit. on pp. 1, 9, 11).

[Tits and Yang 1996] A L Tits and Yaguang Yang. "Globally convergent algorithms for robust pole assignment by state feedback". In: *IEEE Transactions on Automatic Control* 41.10 (1996), pp. 1432–1452. doi: 10.1109/9.539425 (cit. on p. 25).

[Todorov and Weiwei Li 2005] E Todorov and Weiwei Li. "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems". In: *Proceedings of the 2005, American Control Conference, 2005.* 2005, pp. 300–306. doi: 10.1109/ACC.2005.1469949 (cit. on p. 5).

[Emanuel Todorov 2006] Emanuel Todorov. "Optimal control theory". In: *Bayesian brain: probabilistic approaches to neural coding* (2006), pp. 269–298 (cit. on pp. 5, 11).

[Emanuel Todorov *et al.* 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *IEEE International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2012, pp. 5026–5033. isbn: 9781467317375. doi: 10.1109/IROS.2012.6386109. url: http://ieeexplore.ieee.org/document/6386109/ (cit. on pp. 5, 26).

[Tsiamis and Pappas 2021] Anastasios Tsiamis and George J Pappas. "Linear Systems can be Hard to Learn". In: *CoRR* abs/2104.0 (2021) (cit. on pp. 6, 62).

[Richard B Vinter and R. Vinter 2010] Richard B Vinter and RB Vinter. *Optimal control*. Springer, 2010 (cit. on p. 5).

[Zabczyk 1992] Jerzy Zabczyk. *Mathematical control theory - an introduction*. Systems & Control: Foundations & Applications. Birkhäuser, 1992 (cit. on p. 23).