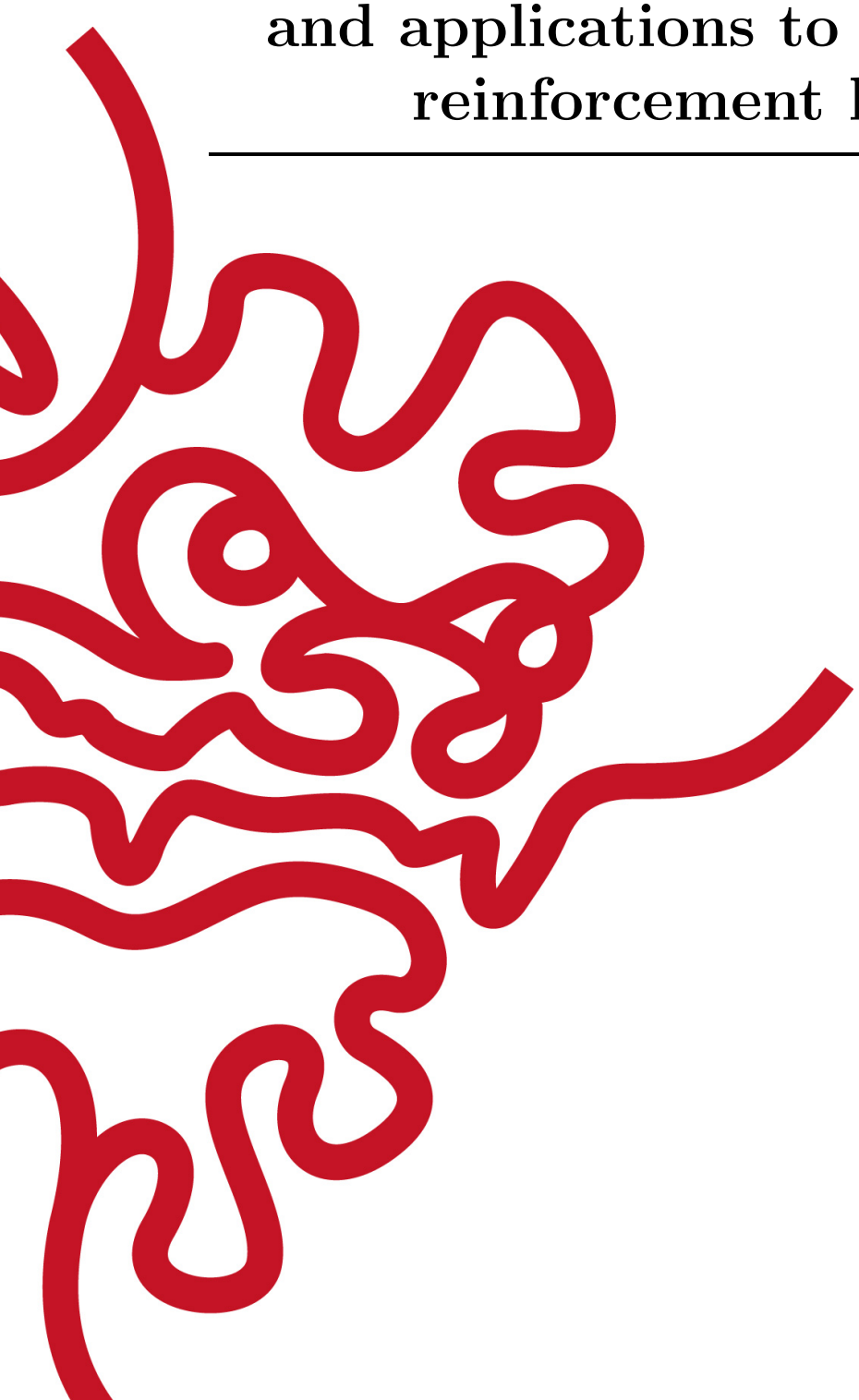
Okinawa Institute of Science and Technology
Graduate University

Thesis submitted for the degree

Doctor of Philosophy

Total stochastic gradient algorithms and applications to model-based reinforcement learning



by

Paavo Parmas

Supervisor: **Kenji Doya**

January, 2020

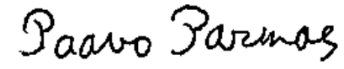
Declaration of Original and Sole Authorship

I, Paavo Parmas, declare that this thesis entitled *Total stochastic gradient algorithms and applications to model-based reinforcement learning* and the data presented in it are original and my own work.

I confirm that:

- No part of this work has previously been submitted for a degree at this or any other university.
- References to the work of others have been clearly acknowledged. Quotations from the work of others have been clearly indicated, and attributed to them.
- In cases where others have contributed to part of this work, such contribution has been clearly acknowledged and distinguished from my own work.
- None of this work has been previously published elsewhere, with the exception of the following:
  1. Parmas, P., Rasmussen, C. E., Peters, J., & Doya, K. (2018, July). PIPPS: Flexible Model-Based Policy Search Robust to the Curse of Chaos. In International Conference on Machine Learning (pp. 4062-4071).
  2. Parmas, P. (2018). Total stochastic gradient algorithms and applications in reinforcement learning. In Advances in Neural Information Processing Systems (pp. 10204-10214).

Date: January, 2020

Signature: 

Abstract

Total stochastic gradient algorithms and applications to model-based reinforcement learning

Optimizing via stochastic gradients is a powerful and flexible technique ubiquitously used in machine learning, reinforcement learning, control, operations research, etc. In many of these applications, the gradients are estimated through a stochastic sampling process, and the learning performance hinges on the accuracy of the estimated gradients. This thesis develops a collection of several novel statistical algorithms to acquire improved gradient estimation accuracy. The need to develop such algorithms was motivated from a model-based reinforcement learning (MBRL) scenario, where I observed that chaotic properties of the dynamics caused the gradient variance to explode when using standard gradient estimation techniques, such as reparameterization gradients. The new techniques sometimes improve the accuracy by 10^6 times and more. The methods rely on both new gradient estimators, as well as clever algorithms to take advantage of the graph structure of the computations to combine estimators in a statistically principled way. While the work started by trying to solve a specific problem related to MBRL, the proposed solutions are general and applicable to any other stochastic computation graph. The problems with chaos have recently been also observed in other tasks, such as meta-learning or protein folding software, and the solutions may prove useful in those domains as well. The main contributions are 1) an MBRL framework called PIPPS, which is similar to the PILCO algorithm, but lifts all of its restrictions by swapping the cumbersome moment-matching computations with a particle sampling approach while achieving the same learning performance with no down-sides, 2) the total propagation algorithm, which is a replacement for backpropagation that prevents the exploding gradient problem by combining gradient estimators in the backwards pass, 3) the probabilistic computation graph framework, which is an intuitive visual method to reason about total gradients on graphs, 4) new policy gradient estimators derived by using the probabilistic computation graph framework, 5) some theoretical discussion about control variates for gradients, a unified theory of reparameterization and likelihood ratio gradient estimators, and an optimal importance sampling scheme for reducing likelihood ratio gradient variance. I hope this work may lead towards new software frameworks that go beyond backpropagation, and implement more advanced methods for estimating gradients.

Acknowledgment

I am grateful to my supervisor Kenji Doya who trusted me enough to let me do whatever I wanted, and provided me with an excellent environment to pursue my research.

I would also like to thank Carl Edward Rasmussen at the University of Cambridge, Jan Peters at TU Darmstadt, Jun Morimoto at the Advanced Telecommunications Research Institute International (ATR), Masashi Sugiyama at the University of Tokyo and RIKEN-AIP for hosting me at their labs for several months. Jan Peters was also the external examiner for my thesis proposal exam, and gave good recommended reading. In particular the suggestion to write down all of the shortcomings of PILCO was useful, as it helped me identify that getting particle sampling methods to work would solve all of the problems, and is thus a highly impactful research topic. Carl Edward Rasmussen had some prior experience with particle methods. In particular he had supervised Andrew McHutchon attempting to use particles in PILCO-like approaches, and gave some good suggestions. For example, he suggested to plot the landscape and the gradients of the objective in Figure 4.2. What I did differently from McHutchon, was that I performed a more careful experimental design, adding errorbars on my plots and fixing the random number seeds, and this allowed me to elucidate the problems. Carl was also my supervisor during my undergraduate studies' final year project, and taught me a lot about machine learning. Masashi Sugiyama gave several useful pieces of advice on mathematical notation and clarity in writing.

I would also like to thank my external examiners Thomas Dietterich and Hidetoshi Shimodaira whose careful reading of the thesis helped me improve the clarity and style.

I am happy to have had the opportunity to do an internship at DeepMind Paris lead by Remi Munos. It was a pleasure to work with my collaborators Karl Tuyls, Daniel Hennes and Shayegan Omidshafiei.

I thank the other members of the Neural Computation Unit for a supportive environment. I greatly enjoyed our discussions during teatime. I thank Jiexin Wang, who built a mobile phone robot, which I used in experiments with PILCO in the early days of my PhD studies. I thank the unit administrators, Kikuko Matsuo, Emiko Asato and Misuzu Saito, without whose support nothing would get done.

I thank the members at the Fujita lab at the Tokyo Institute of Technology, where I spent my first month in Japan during an OIST gap program. I could not have asked for a warmer welcome to Japan.

Finally, I would like to thank all the staff at the Graduate School Office for their constant support. In particular, I am grateful for having been able to help with outreach activities such as the yearly Science Challenge workshops, which I enjoyed immensely.

Abbreviations

PILCO	Probabilistic Inference for Learning Control
PIPPS	Probabilistic Inference for Particle-based Policy Search
w.r.t.	with respect to
s.t.	such that
MBRL	model-based reinforcement learning
MM	moment matching
RL	reinforcement learning
RP	reparameterization
LR	likelihood ratio
SLRG	slice ratio gradient
BLR	likelihood ratio gradient for a Beta distribution
BRG	slice ratio gradient for a Beta distribution
TRRG	truncated ratio gradient
TP	total propagation
GR	Gaussian resampling
GS	Gaussian shaping
GLR	likelihood ratio Gaussian shaping gradient
GTP	total propagation Gaussian shaping gradient
DEL	density estimation likelihood ratio gradient
PCG	probabilistic computation graph
SCG	stochastic computation graph
ES	Evolution Strategies
Q.E.D.	quod erat demonstrandum (what was to be shown)
GP	Gaussian process
iff	if and only if
pdf	probability density function
cdf	cumulative distribution function
SGD	stochastic gradient descent
BFGS	Broyden-Fletcher-Goldfarb-Shanno (a quasi-Newton optimizer)

Glossary

Machine learning	A scientific discipline dealing with computers/machines that learn from data.
Reinforcement learning	A subfield of machine learning dealing with learning to take actions to maximize rewards.
Agent	An entity taking actions in an environment, trying to maximize the rewards it receives.
Model-based reinforcement learning	A type of reinforcement learning, where the agent concurrently learns to make predictions about what will happen in the environment when particular actions are taken.
Deep learning	A type of machine learning, where the model consists of multiple layers of information processing usually learning a representation of the data.
Exploding gradient problem	A phenomenon in deep learning where the magnitude of the gradient of a model becomes very large, leading to large gradient steps and problems in the optimization.
Curse of Chaos	Phenomenon in which the backpropagated gradient variance explodes due to chaotic dynamics.
Chaotic gradients	Another name for the curse of chaos phenomenon.
Moment matching	A method for approximating a distribution by a Gaussian by matching the mean and variance to the exact mean and variance of the true distribution.
Monte Carlo method	A method that uses sampling to solve problems.
Reparameterization gradient	One type of Monte Carlo gradient estimator of an expectation of a function using samples of the gradient of said function.
Likelihood ratio gradient	One type of Monte Carlo gradient estimator of an expectation of a function using samples of the value of said function.
Baseline	A method for reducing likelihood ratio gradient variance, by subtracting a value, the baseline, from the sampled points.

Nomenclature

$\mathbf{x}$	state of system
$\mathbf{u}$	control/action
A	matrix
$\mathbf{a}$	vector
a	scalar
$\hat{a}$	an estimator of a scalar
$\mathcal{N}(\mathbf{x}; \mu, \Sigma)$	Gaussian distribution over $\mathbf{x}$ with mean μ and covariance Σ
$\mathcal{GP}$	Gaussian process model
R	reward function
r_t	reward at time t
G_t	return $\sum_{h=t}^{T} r_t$ (sum of rewards until end of episode)
c	cost function
$\frac{\partial f(x,y)}{\partial x}$	partial derivative of f w.r.t. x
$\frac{\partial f(x,y)}{\partial x}\Big\vert_y$	partial derivative of f w.r.t. x while keeping y constant
$\frac{\mathrm{d} f(x,y)}{\mathrm{d} x}$	total derivative of f w.r.t. x: $\frac{\mathrm{d} f(x,y)}{\mathrm{d} x} = \frac{\partial f(x,y)}{\partial x}\Big\vert_y + \frac{\partial f(x,y)}{\partial y}\Big\vert_x \frac{\mathrm{d} y}{\mathrm{d} x}$
D	state dimension
F	action dimension
$\mathbb{E}[x]$	expectation of random variable x
$\mathbb{V}[x]$	variance of random variable x
$p(x; \theta)$	probability distribution of x with parameters θ
$p(x\vert y)$	probability distribution of x conditioned on random variable y

I dedicate my thesis to my family,
Ivar, Kaupo, Piret, Erika and Ivo.

Contents

List of Figures

List of Tables

Introduction

Learning with gradient descent

The deep learning revolution (LeCun et al., 2015; Schmidhuber, 2015a) was built using gradient descent and backpropagation (Werbos, 1974; Rumelhart et al., 1988). Gradient descent provided a simple, efficient, general and scalable learning algorithm that allowed training neural networks while taking advantage of the increasing computational resources. While this approach has worked great for supervised learning, credit assignment across long sequences of computations, such as those arising in decision making and control are still challenging avenues of research.

Might the same approach also allow optimizing the behavior of intelligent decision makers? Reinforcement learning (Sutton and Barto, 1998) is a general field, which deals with such decision makers, called agents trying to find the behavior that maximizes their rewards in some environment. In that scenario, standard backpropagation cannot be used to efficiently compute gradients, because the computations may not be differentiable, or there may be stochasticities in the computations, meaning that an estimator of the gradients will be necessary. Among reinforcement learning algorithms, there also exist gradient based methods, such as policy gradient algorithms (Sutton et al., 2000), which allow learning with gradient descent even when the computations are not directly differentiable; however, the typical gradient estimation schemes are not efficient. The main topic of my thesis is the investigation of better gradient estimators, hoping to create efficient algorithms for agents such as robots to learn optimal decision making and behavior in various environments. But first, in the next section I will explain why robots should learn at all.

Why should robots learn?

Robots have become an integral part of the modern world through increasing manufacturing accuracy and efficiency at factories. They can perform fast and precise motion to assemble products according to programmed guidelines. This approach works well in a controlled environment, where unexpected events are rare. However, if robots were to be put into the same diverse situations as people meet in their daily lives, the number of different cases to consider would be so large that it is not feasible to explicitly program a solution to every problem that the robot might encounter. Some examples are care robots for elderly people—these would ideally be as nimble as humans—or movement assisting devices, which would have to adapt to each individuals movement

patterns. If we desire practical service robots that can help humans in our daily lives, these robots need to be able to learn the correct motions by themselves from experience. The bottom line is that control theory has been around for a century, but I still do not have a robot in my kitchen. In my thesis, I investigate model-based reinforcement learning methods, which are a particularly data-efficient way of learning motion control. Moreover, they allow for more efficient forms of gradient computation by utilizing backpropagation (Jordan and Rumelhart, 1992).

Why should learning be model-based?

If you ask a model-free agent why it performed a particular action, it might say: "Because I thought I will get a large reward in the long-term if I perform this action." Such kind of primitive thought patterns could hardly be used to describe an advanced agent acting intelligently. For any agent to be called intelligent, it should be able to make better predictions about the future, to reason about the outcomes of its actions, and be able to predict outcomes without having to try every possible behavior. For any agent to be intelligent, it should use model-based learning algorithms.

There are many advantages of model-based learning algorithms: data-efficiency, safety, transfer learning. **Data-efficiency:** for example the model-based reinforcement learning algorithm PILCO (Deisenroth and Rasmussen, 2011) showed incredible data-efficiency solving tasks using several orders of magnitude less data than model-free alternatives. This algorithm is also a main target of my investigations. **Safety:** if an agent has a model, it is able to detect when the environment has changed, and it is thus possible to implement counter-measures to failures. **Interpretability:** the model predictions can be used to improve interpretability of the agent's behavior, as it is possible to predict exactly what the agent thinks will happen. **Transfer learning:** if there are multiple tasks in the same environment, but with a different reward function, the same learned dynamics model can be used among the different tasks, thus allowing for transfer learning.

There are no real downsides to model-based reinforcement learning, except for maybe added complexity, and the only question is how to get such algorithms to work effectively. In my thesis I elucidate several issues, which may have been preventing more wide spread use of model-based reinforcement learning.

Contributions

The main focus of my work was on trying to overcome the limitations of the PILCO algorithm caused by its inflexible use of moment matching predictions, which are not generally applicable, e.g., the predictions cannot be used with expressive neural network based models, severely limiting the applicability of such algorithms. To overcome such limitations, I attempted using flexible Monte Carlo particle sampling based predictions instead, which are generally applicable. By tackling this task, I stumbled on a more general problem of effective gradient estimation—even though using particle predictions is easy, computing good gradients is not straightforward. Through my work,

I was able to uncover that chaotic properties of long chains of computations can render backpropagation-like gradient estimators useless, and worked towards new better gradient estimation schemes.

The main contributions are:

- A unified view of likelihood ratio and reparameterization gradient estimators.
- An optimal importance sampling scheme to reduce likelihood ratio gradient variance.
- Analysis of baseline techniques for reducing likelihood ratio gradient variance.
- The probabilistic computation graph framework, which generalizes previous "policy gradient theorems", and provides a visual and intuitive method for deriving gradient estimators.
- New gradient estimators including the density estimation likelihood ratio gradient as well as the Gaussian shaping gradient method.
- An analysis of how chaotic properties of long chains of computations cause backpropagation gradients to become ill-behaved.
- The total propagation algorithm, which is an elegant solution to the issue with chaos, and allows combining gradient estimators during the backwards computation of gradient estimation.
- The Probabilistic Inference for Particle-based Policy Search (PIPPS) model-based reinforcement learning algorithm that uses particles for predictions instead of the cumbersome moment matching predictions in the PILCO algorithm.

Structure of thesis

The thesis starts with background information in Chapter 1 including typical explanations of gradient estimators and PILCO, as well as explanations of the shortcomings of PILCO. Chapters 2–4 give the main novel contributions. In Chapter 2 I explain novel views of gradient estimators through a single sampling operation, including explanations of likelihood ratio and reparameterization gradients. Chapter 3 discusses how to use the estimators in Chapter 2 for gradient estimation on graphs of computations. In particular, I discuss my new probabilistic computation graph framework. Finally, Chapter 4 utilizes these gradient estimators in model-based reinforcement learning using particle predictions. I discuss how the performance of gradient estimators is problem dependent, how chaotic properties can cause backpropagation-like gradient estimators to become ill-behaved, and how to effectively combine estimators using my total propagation algorithm.

Chapter 1

Background

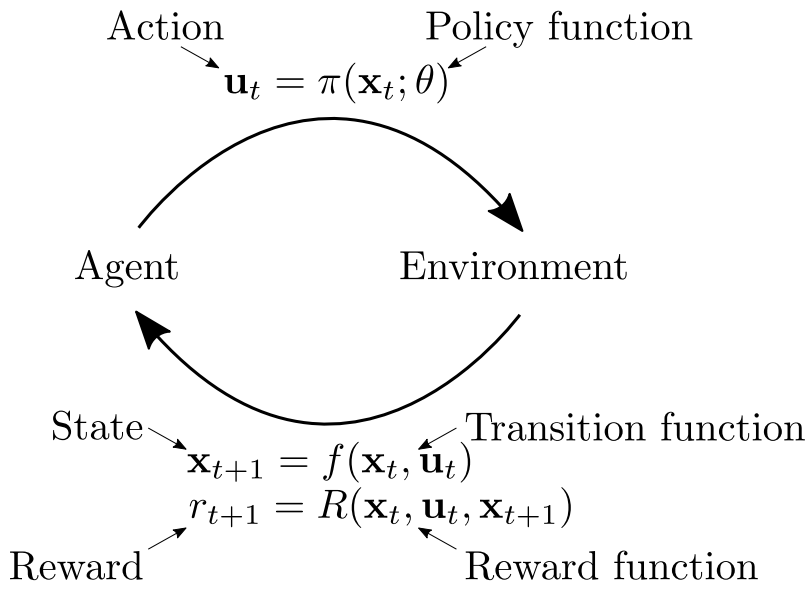


Figure 1.1: In reinforcement learning (RL), an agent is in a state $\mathbf{x}$ in an environment, and it can move around by picking actions $\mathbf{u}$ (although the agent does not necessarily know the outcomes of its actions). The agent's goal is to maximize long-term rewards, by picking the appropriate actions. The difficulty lies in the fact that actions have long-term consequences (an action taken now may lead one into a good situation in the future, where they can acquire lots of reward), but it is difficult to know which actions in the past contributed to the current situation.

1.1 Episodic policy search

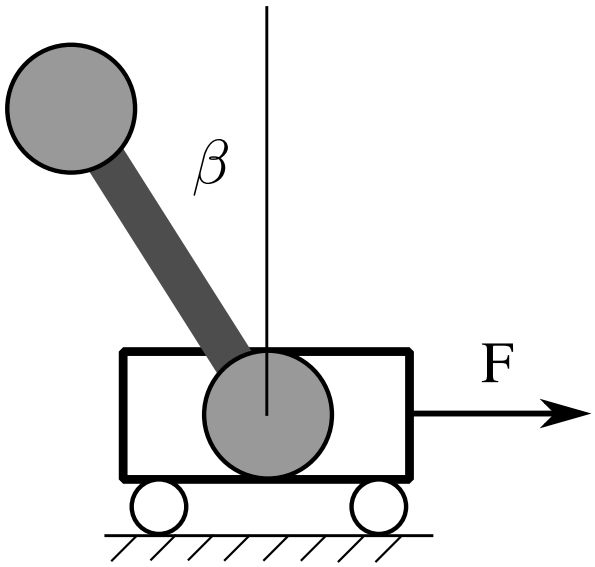


Figure 1.2: Cart-pole: common benchmark task in control and reinforcement learning

Here I give the background to how gradient based learning algorithms can be applied to train systems to improve their control policy π_θ (a function that determines what action to apply depending on the situation). The methods in my thesis are applicable to many tasks, such as stochastic variational inference (Hoffman et al., 2013), price sensitivity estimation in finance (Fu and Hu, 1995), etc. but I focus on robot control in my exposition.

In the episodic learning setting, an agent, such as a robot, is given multiple attempts at solving a task. Each attempt is given a numerical score, which I call the return G. The agent's goal is to find a behavior that gives the best expected return. For a concrete example, consider the cart-pole robot system in Figure 1.2. The task is to move the cart back and forth so as to swing up the pendulum and keep it balanced in the upright position. This is a common benchmark task in control and reinforcement learning, and I use it as a basic example in my explanations and many of my experiments.

I consider discrete time systems which at any time step t can be described by the state vector $\mathbf{x}_t$, for example the position of the cart s, angle of the pendulum β and the velocities $\dot{s}, \dot{\beta}$; and the applied action/control vector $\mathbf{u}_t$, for example the force applied on the cart F. An episode starts by sampling a state from a fixed initial state distribution $\mathbf{x}_0 \sim p(\mathbf{x}_0)$, for example with the cart in the center and the pendulum hanging downwards. The policy π_θ is a function parameterized by θ, and it determines what action is applied $\mathbf{u}_t \sim p(\mathbf{u}_t) = \pi(\mathbf{x}_t; \theta)$. Having applied an action, the state transitions to a new state according to an unknown dynamics function $\mathbf{x}_{t+1} \sim p(\mathbf{x_{t+1}}) = f(\mathbf{x}_t, \mathbf{u}_t)$. Both the policy and the dynamics may be stochastic and nonlinear. Actions and state transitions are repeated for up to T time-steps, giving rise to a trajectory τ, which is the sequence $(\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, ..., \mathbf{x}_T)$.

Each episode is scored according to the return function $G(\tau)$. Often, the return decomposes into a sum of rewards for each time-step $G(\tau) = \sum_{t=0}^{T} r_t$ in which case the goal is to optimize the policy parameters θ to maximize the expected return $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[G(\tau)]$. In my work I consider the equivalent problem of minimizing a cost $G(\tau) = \sum_{t=0}^{T} c(\mathbf{x}_t)$, where $c(\mathbf{x})$ is the cost function. I do not consider cost functions that explicitly depend on the actions $\mathbf{u}$, though they would be trivial to include. In the cart-pole case, there are many possible ways to define the cost, e.g. the cost may be to maximize the height of the pole, or to minimize the distance of the tip of the pendulum to the position of the tip of the pendulum when the cart-pole system is upright. Note that in the general case, the return function may not be differentiable and need not consist of a sum of terms for each time step, but could be a function of the whole trajectory. For example a robot might be given a task to jump as high as it can, and the return to maximize would be the height at the peak of the trajectory. Many of the previous approaches described in my thesis, such as PILCO, are not easily extended to work in such a setting, whereas the new methods I develop could be directly applied.

Learning proceeds by alternating between executing the policy once or multiple times on the system, then updating the parameters with the aim of improving the performance on the following attempts. There exist many different methods of updating the parameters, but perhaps the most obvious idea is to directly estimate the gradient of the objective function $\frac{\mathrm{d}}{\mathrm{d}\theta} J(\theta)$ and use it for optimization—so called policy gradient methods. In Sections 1.1.1 and 1.1.2 I explain model-free and model-based policy gradients respectively.

Remark: Policy search is not restricted to Markov Decision Processes (MDPs)

Often, RL is explained in terms of the formalism of Markov Decision Processes, which essentially just says that the next state $\mathbf{x}'$ and reward r depend only on the current state $\mathbf{x}$ and action $\mathbf{u}$, not on anything that happened earlier. But RL is a more general problem. The essence of RL is about an agent learning to obtain rewards in an environment, no matter what properties the environment has. Many RL algorithms, such as Q-learning (Watkins, 1989) require this MDP assumption; however, an advantage of episodic policy gradient algorithms is that they work directly even if the MDP assumption is lifted. The environment could be either only partially observable (POMDP), or the state transitions or rewards could depend also on past states and actions, but episodic policy search methods would still be applicable without major modifications.

1.1.1 Model-free policy gradient algorithms

To use policy gradient methods, one must somehow estimate the derivative of the expected return w.r.t. the policy parameters $\frac{\mathrm{d}}{\mathrm{d}\theta} J(\theta) = \frac{\mathrm{d}}{\mathrm{d}\theta} \mathbb{E}_{\tau \sim p(\tau;\theta)}[G(\tau)]$. Here I explain the likelihood ratio gradient method (Glynn, 1990; Williams, 1992)—a general technique to estimate the gradient of the expectation of an arbitrary function $\phi(x)$ with respect to the parameters of the sampling distribution $\frac{\mathrm{d}}{\mathrm{d}\theta} \mathbb{E}_{x \sim p(x;\theta)}[\phi(x)]$. This estimator is sometimes also called the score function estimator, but I prefer the name

"likelihood ratio gradient", because it is more informative about the mechanism behind the estimation technique. This technique relies on using a stochastic policy $\pi(\mathbf{u}|\mathbf{x};\theta)$, e.g. Gaussian noise could be added to the action. We will see that by doing so, it is possible to estimate the gradient without knowing anything about the environment.

Likelihood ratio gradient (LR): The desired gradient can be written as $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{x\sim p(x;\theta)}\left[\phi(x)\right] = \int \frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}\phi(x)\mathrm{d}x$. Multiplying and dividing by $p\left(x;\theta\right)$ we get:

$$\begin{aligned}\int \frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}\phi(x)\mathrm{d}x &= \mathbb{E}_{x\sim p}\left[\frac{\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}}{p(x)}\phi(x)\right] \\ &= \mathbb{E}_{x\sim p}\left[\frac{\mathrm{d}\log p(x;\theta)}{\mathrm{d}\theta}\phi(x)\right]\end{aligned}$$

The LR gradient often has a high variance, and has to be combined with variance reduction techniques known as control variates (Greensmith et al., 2004). A common approach subtracts a constant baseline b from the function values to obtain the estimator $\mathbb{E}_{x\sim p}\left[\frac{\mathrm{d}}{\mathrm{d}\theta}\left(\log p(x;\theta)\right)\left(\phi(x)-b\right)\right]$. If b is independent from the samples, this can greatly reduce the variance without introducing any bias. In practice, the sample mean is a good choice $b = \mathbb{E}\left[\phi(x)\right]$. When estimating the gradient from a batch, one can estimate leave-one-out baselines for each point to obtain an unbiased gradient estimator (Mnih and Rezende, 2016), i.e. $b_i = \sum_{j\neq i}^{P}\phi(x_j)/(P-1)$. In Section 2.4 I perform my own analysis of the effect of using baselines.

LR policy gradient in a trajectory setting: The probability density $p(\tau) = p(\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, ..., \mathbf{x}_T)$ of observing a particular trajectory can be written as $p(\mathbf{x}_0)\pi(\mathbf{u}_0|\mathbf{x}_0;\theta)p(\mathbf{x}_1|\mathbf{x}_0,\mathbf{u}_0)...p(\mathbf{x}_T|\mathbf{x}_{T-1},\mathbf{u}_{T-1})$.

To use LR gradients, note that $p(\tau)$ is a product, so $\log p(\tau)$ can be transformed into a sum. Denote $G_h(\tau) = \sum_{t=h}^{T} c(\mathbf{x}_t)$. Noting that (1) only the action distributions depend on the policy parameters, and (2) an action does not affect costs obtained at previous time steps leads to the gradient estimator: $\mathbb{E}\left[\sum_{t=0}^{T-1}\left(\frac{\mathrm{d}}{\mathrm{d}\theta}\log\pi(\mathbf{u}_t|\mathbf{x}_t;\theta)G_{t+1}(\tau)\right)\right]$ In particular, note that as the $p\left(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t\right)$ terms do not depend on θ, it is not necessary to know the transition dynamics to estimate the policy gradient.

Deterministic policy gradients: For completeness, I note that there exists an alternative model-free method of estimating the gradient, which takes advantage of function approximation, and can even be used with deterministic policies (Silver et al., 2014). The gradient estimator is given by: $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}\left[G\right] = \mathbb{E}\left[\sum_{t=0}^{H-1}\frac{\mathrm{d}\mathbf{u}_t}{\mathrm{d}\theta}\frac{\mathrm{d}\hat{Q}_t(\mathbf{u}_t,\mathbf{x}_t)}{\mathrm{d}\mathbf{u}_t}\right]$, where $\hat{Q}_t(\mathbf{u}_t,\mathbf{x}_t)$ is a function approximator for the Q-value defined as

$$Q_t(\mathbf{u},\mathbf{x}) = \mathbb{E}\left[G_t|\mathbf{u}_t=\mathbf{u},\mathbf{x}_t=\mathbf{x}\right].$$

Later in my thesis, in Chapter 3 I describe my probabilistic computation graph framework, which allows to intuitively explain the relationship between deterministic policy gradient methods and LR gradients.

1.1.2 Model-based policy gradients

Model-based methods differ from model-free methods, in the sense that one has access to either the model f or an approximation of it $\hat{f}$. A naive approach to model-based policy gradients may simply apply the same model-free methods in simulations using the model; however, having access to the model allows for fundamentally different policy gradient algorithms. In particular, consider the case when the dynamics are deterministic. In such a case the full trajectory could be differentiated by applying the chain rule, and $\frac{\mathrm{d}G}{\mathrm{d}\theta}$ could be computed by using backpropagation. In the general case though, the dynamics are not deterministic. What to do when the dynamics are stochastic? One approach is to differentiate through the stochastic sampling operations using the reparameterization trick explained next.

Reparameterization gradient (RP): Consider sampling from a univariate Gaussian distribution. One approach first samples with zero mean and unit variance $\epsilon \sim \mathcal{N}(0,1)$, then maps this point to replicate a sample from the desired distribution $x = \mu + \sigma\epsilon$. Now it is straightforward to differentiate the output w.r.t. the distribution parameters $\theta = [\mu, \sigma]$, namely $\frac{\mathrm{d}x}{\mathrm{d}\mu} = 1$ and $\frac{\mathrm{d}x}{\mathrm{d}\sigma} = \epsilon$. Averaging samples of $\frac{\mathrm{d}\phi}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}\theta}$ gives an unbiased estimate of the gradient of the expectation $\mathbb{E}_{x\sim p(x;theta)}\left[\phi(x)\right]$, for any function $\phi(x)$. This is the RP gradient for a normal distribution. For a multivariate Gaussian, the Cholesky factor $(L, \text{s.t.}\Sigma = LL^T)$ of the covariance matrix can be used instead of σ. See (Rezende et al., 2014) for non-Gaussian distributions.

Such reparameterization gradients often yield much more accurate gradient estimates compared to LR gradients (Rezende et al., 2014). In particular, the reparameterization gradient scales well with the dimension of the sampling space. For example, consider the case when $\phi(\mathbf{x})$ is a linear function. In this case, a single sample yields an exact gradient of the expectation w.r.t. μ with the RP trick irrespective of the dimension. In contrast, the LR gradient variance will grow linearly as the dimension is increased. It is thus desirable to incorporate gradients computed using the RP method into the RL learning framework.

1.2 Model-based policy search

Section 1.1.2 explained how one can use a model to estimate the policy gradient. But where does this model come from, and how does it fit together into a learning framework? One approach is to perform learning by iterating between attempting to solve the task on the real system, model learning and policy optimization in simulations as explained in Algorithm 1 and Figure 1.3. In the first episode, the agent has no data and acts almost randomly. Before the next episode, the agent uses all of the data it has accumulated until that point to learn a predictive model $\hat{f}$. Then, using the learned model, it can imagine the outcome of an episode for any policy parameters θ, and uses simulations to optimize θ, so that it could try out a new behavior in the next trial. In practice one can perform hundreds of simulated trials to optimize the policy per one real trial, which is used to learn the model.

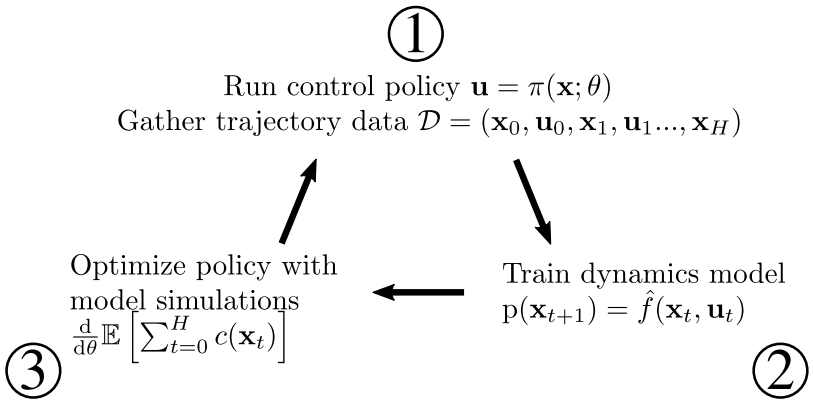


Figure 1.3: Illustration of an iterative model-based policy search framework

Algorithm 1 Episodic policy search with optimization using model-based gradients

Input: initial policy parameters θ_0, episode length T, initial state distribution $p(\mathbf{x}_0)$, number of optimization steps N, cost function $c(\mathbf{x})$, policy π, s.t. $\mathbf{u} \sim \pi(\mathbf{x}; \theta)$.
repeat
1. Execute policy on system, attempting to solve task, store data.
2. Train dynamics model $\hat{f}$, s.t. $p(\mathbf{x}_{t+1}) = \hat{f}(\mathbf{x}_t, \mathbf{u}_t)$.
3. Optimize policy parameters:

for $n = 1$ **to** N **do**
Predict: $\{p(\mathbf{x}_t) \mid t \in 1...T\}$ with $\pi(\mathbf{x}; \theta_n), \hat{f}, p(\mathbf{x}_0)$
Compute objective and gradient:
$J(\theta_n) = \sum_{t=0}^{T} \mathbb{E}\left[c(\mathbf{x}_t)\right], \quad \frac{\mathrm{d}}{\mathrm{d}\theta}\left(J(\theta_n)\right)$
Update: $\theta_{n+1} = function\left(\theta_n, J(\theta_n), \frac{\mathrm{d}J(\theta_n)}{\mathrm{d}\theta}\right)$
end for
until stop criterion, e.g. policy converges

1.3 PILCO

PILCO (Deisenroth and Rasmussen, 2011) is a model-based policy search method, which follows the framework in Section 1.2. It is a remarkable algorithm, which achieved incredible data-efficiency at learning continuous control problems from scratch, e.g. cart-pole swing-up and balancing in 20s. The success was based on two components:

1. A principled consideration of model uncertainty using Gaussian process (GP) models (Rasmussen and Williams, 2006). See also Appendix A for an explanation of GPs.

2. An analytic approximation of the trajectory distribution based on Gaussian moment matching, together with an analytic estimation of the expected cost and its gradient.

Algorithm 2 Analytic moment matching based trajectory prediction and policy evaluation (used in PILCO)

Input: policy π with parameters θ, episode length T, initial Gaussian state distribution $p(\mathbf{x}_0)$, cost function $c(\mathbf{x})$, learned dynamics model $\hat{f}$.
Requirements: if the input distribution is Gaussian, can analytically compute the expectations and variances of the outputs of $\pi(\mathbf{x})$, $\hat{f}(\mathbf{x}, \mathbf{u})$, $c(\mathbf{x})$, and differentiate them.
for $t = 0$ **to** $T - 1$ **do**
1. Using $p(\mathbf{x}_t)$ and π compute a Gaussian approximation to the joint state-action distribution:
$p(\tilde{\mathbf{x}}_t) = \mathcal{N}(\tilde{\mu}_t, \tilde{\Sigma}_t)$, where $\tilde{\mathbf{x}}_t = [x_t^T, u_t^T]^T$
2. Using $p(\tilde{\mathbf{x}}_t)$ and $\hat{f}$ compute a Gaussian approximation to the next state distribution:
$p(\mathbf{x}_{t+1}) = \mathcal{N}(\mu_{t+1}, \Sigma_{t+1})$
3. Using $p(\mathbf{x}_{t+1})$ and $c(\mathbf{x})$ compute the expected cost:
$\mathbb{E}\left[c(\mathbf{x}_{t+1})\right]$
end for
Gradient computation: $\frac{\mathrm{d}}{\mathrm{d}\theta}\left(\sum_{t=1}^{T} \mathbb{E}\left[c(\mathbf{x}_{t+1})\right]\right)$ is computed analytically during the for-loop by differentiating each computation separately, applying the chain rule, and accumulating the gradients.

The higher level view of PILCO follows Algorithm 1 and the moment matching based policy evaluation is detailed in Algorithm 2. As analytic gradients are obtained, any deterministic optimizer (Nocedal and Wright, 2006), such as BFGS (Broyden-Fletcher-Goldfarb-Shanno) can be used to optimize the policy parameters. In the following sections I provide additional details about the learned models, and the moment matching predictions. For other details refer to Algorithms 1 and 2, and the original paper (Deisenroth and Rasmussen, 2011).

1.3.1 Model learning

I follow the original PILCO, which uses Gaussian process dynamics models to predict the change in the state from one time step to the next, i.e. $p(\Delta x_{t+1}^a) = \mathcal{GP}(\mathbf{x}_t, \mathbf{u}_t)$, where $a \in \{1...D\}$, x^a denotes the a$^{\text{th}}$ dimension of the state-space, and $\Delta x_{t+1}^a = x_{t+1}^a - x_t^a$. A separate Gaussian process is learned for each dimension a. A squared exponential covariance function is used throughout $k_a(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}') = s_a^2 \exp(-(\tilde{\mathbf{x}} - \tilde{\mathbf{x}}')^T \Lambda_a^{-1} (\tilde{\mathbf{x}} - \tilde{\mathbf{x}}'))$, where s_a^2 and $\Lambda_a = \mathrm{diag}([l_{a1}, l_{a2}, ..., l_{aD+F}])$ are the function variance and length scale hyperparameters respectively. I use a Gaussian likelihood function with noise hyperparameter n_a^2. The hyperparameters are optimized to maximize the marginal likelihood. Note that in my experiments I use additive observation noise, and the models are trained to predict the next observation given the current observation. One could obtain more confident predictions by explicitly modeling the latent state using a state-space model—this approach is for example used by McAllister and Rasmussen (2016). The PILCO approach is not so much defined by what model is being used—any applicable probabilistic model is acceptable. The key distinct feature is the moment matching

Gaussian approximations of the state distributions explained in the next section.

1.3.2 Moment matching prediction

Moment matching refers to a method for approximating the output distribution when an input distribution is mapped through a function. In general, when a Gaussian distribution is mapped through a nonlinear function, the output distribution will be intractable and non-Gaussian; however, in some cases one can analytically evaluate the mean and variance of the output distribution. With Gaussian moment matching, one approximates the output distribution as a Gaussian distribution by matching the mean and variance to be the same as that of the true distribution. During each time step, moment matching is performed twice: once during evaluating the state-action joint distribution, the second time during evaluating the output of the dynamics function $\hat{f}$. The framework for predicting the trajectory and computing the gradient is explained in Algorithm 2. Note that even though the state-dimensions are modeled with separate functions $\hat{f}_a$, the moment matching is performed jointly, and the state distributions can include covariances (for example if two of the functions $\hat{f}_a$ were the same, their outputs would have a high covariance, because as the input point varies, the mean of the predictions would be the same for both functions).

Remark: Moment matching is perhaps the biggest issue in PILCO

While moment matching (MM) provided a deterministic method for computing trajectory distributions in PILCO, and hence allowed for efficient optimization, throughout my thesis I find that MM is restrictive, prevents one from building more advanced algorithms, and even gives overly conservative predictions.

1.3.3 Literature on PILCO

To give a flavor for what can be done with PILCO, and what sort of challenges it faces, I review some of the recent literature. PILCO was started in Deisenroth's PhD Thesis (Deisenroth, 2010). At that time, all of the key ingredients of PILCO were already present, and it achieved good results in problems such as cart-pole swing-up, a unicycle balancing task, etc. Later it was published in more compact form at the ICML conference (Deisenroth and Rasmussen, 2011) and in a journal (Deisenroth et al., 2014). After creation of the basic PILCO algorithm, various related works have been published.

PILCO was applied to a block-stacking task where a low-cost robotic arm manipulator was used to stack toy blocks on top of each other (Deisenroth et al., 2011). Feedback of the position was acquired using a cheap Kinect-style depth sensor. The paper included a straight forward introduction of constraints to the path planning problem, by adding penalties into the cost function in regions where a collision would occur. Adding this penalty not only greatly reduced collisions when moving the arm (close to 0), but also improved the block-stacking performance. Essentially, this is one of the many demonstrations that a good reward/cost function can improve performance.

The framework was applied to learn a single controller that would work for multiple targets (Deisenroth and Fox, 2011). In this case the controller was a function of both the state and the target. The method was successfully applied to the cart-pole swing-up problem as well as block-stacking. The authors noted that typically methods for blending local controllers together did not achieve the same performance as the presented method.

The multiple target version was applied to solving a challenging throttle valve control task (Bischoff et al., 2013). In addition to multiple targets, multiple start locations were also used, and a controller to perform trajectory tracking was learned. The work showed that PILCO can be applied to challenging real world problems in industry.

PILCO was applied to learn a controller for legged locomotion (Deisenroth et al., 2012). The experiments were done on a simulation of a biomechanical walker robot. The paper included dimensionality reduction to reduce the state-space so as to allow learning a controller more efficiently.

A mean function for the Gaussian process models was introduced to serve as a prior belief of the dynamics (Bischoff et al., 2014). In the paper, they made linear predictions of the position with the mean function, and used the deviation from this position as the target for the GP. They applied it to simulations of a throttle valve, as well as on a wheeled robot with a gripper performing an item fetching task, and achieved improved performance, although the difference did not seem large.

PILCO has been extended to imitation learning (Englert et al., 2013). Essentially in this work they set the cost function to the KL divergence between expert demonstrations, and the predicted trajectory based on the GP models. They were able to learn behavior to bounce a ball on a racket with a robotic arm.

A work which proves stability of controllers based on Gaussian process models was published (Vinogradska et al., 2016). Before this work, there was no guarantee of stability for controllers based on Gaussian process models. This may have deterred people in industry from using such controllers, especially in safety critical situations. This work made a step towards proving stability. In the work they dealt with uncertain inputs by using Gaussian quadrature to perform forward predictions, instead of the moment matching typically used in PILCO. They found that the trajectory distribution is found more accurately by using this method.

Two works were presented at the data-efficient machine learning workshop at ICML'16 (Gal et al., 2016; McAllister et al., 2016). The first replaces the Gaussian process models in PILCO with bayesian neural networks, and achieves still good, but slightly lower data-efficiency compared to PILCO (3–4 times slower). However, it achieved better control performance in the end (lower cost of the trajectories). The second work uses Bayesian optimization objectives such as expected improvement instead of the usual cost in PILCO. As PILCO lacks explicit exploration such objectives which trade off exploration and exploitation are an important area of research. The results in the paper online included a mistake (the authors told me at the conference they had not used enough sample trajectories); however, at the workshop they presented results that showed a slightly improved performance.

PILCO has also been extended to the partially observable case (McAllister and Rasmussen, 2016). The work implements a way to optimize a policy while assuming

that a filter will be applied during the trial. Typically control problems have noise, but by using a filter one can over time achieve a better state estimate. If a filter is going to be used during control, it is better to include this in the policy optimization as well, and this work does just that.

In addition to these works, I also previously used the algorithm under the supervision of Carl Rasmussen, one of the key authors of the software. I successfully applied it to a control problem involving swinging and balancing a pendulum on a cart while measuring the position of the system using an inexpensive camera. I did this work as part of my MEng Thesis, but the work has not been published. While Deisenroth and Rasmussen (2011) had previously applied PILCO on the same system, while measuring the state using the sensors in the apparatus, in my work, I had to overcome delayed measurements from the camera. This could be done in a straight-forward manner by appending previous control signals to the state space used for the model predictions. Knowing the previously applied control signals, there is sufficient information to predict what the observation at the next time step will be. While this state space worked for predictions, it was unsuitable for the controller. A controller could only be learned when the previous control signals were not a part of the input to the controller.

In my own work, as well as in the literature, every small change to PILCO has required extensive mathematical derivations, and modifications to the code—it is apparent that PILCO has shortcomings, which I will explain in the following section.

1.3.4 Shortcomings of PILCO

Long computational time: While PILCO can solve some problems, such as cart-pole swing-up using only 20s of data, the computational time between the trials can total a few hours. This contrast shows a trade-off between high data-efficiency, and computational efficiency. The main computational bottleneck is the optimization of the policy between trials. This computation scales with $O(N^2(E+F)^2E^2)$, where N is the number of data points, E is the dimensionality of the states that are predicted, and F is the dimensionality of the control signal (McAllister, 2017). The quadratic scaling with the number of data points is an issue, because with modern robots, which often operate at control frequencies of 500Hz, it does not take much time for the amount of data to become very large. This bad scaling can be lessened by using inducing point methods (Quiñonero-Candela and Rasmussen, 2005), such as a FITC sparse approximation of the Gaussian process, but this approximation has flaws (Bauer et al., 2016), and in practice still has limited scalability.

An even bigger issue is the 4th order scaling with the dimension, which in practice limits the use of PILCO to situations with fewer than around 20 dimensions. This bad scaling could be tackled with dimensionality reduction, but it is a difficult problem as of yet. Another option is to use only a part of the robot in a task—for example when learning a task that involves moving the arms of a robot, it may not be important to consider the state of the legs. However, none of these methods will help if the task is fundamentally high-dimensional.

Restrictions to reward functions: In the standard PILCO setup, the reward functions need to be known (although one could extend the method to learn the reward func-

tion as well). Furthermore, the reward function is restricted to functions for which the expectation can be computed analytically when the input is a Gaussian distributed random variable. Such functions include arbitrary polynomials, sums of Gaussians, sums of sinusoids—fortunately all of which are universal function approximators. While this class of functions will suffice in many cases, it can be a restriction in some situations.

Restrictions to policy: Similarly to the reward functions, the policy output also needs to be computable analytically when the input is a Gaussian distribution. This restriction, for example could prohibit the use of neural network controllers, which are very popular lately. As another example, differentiable model-predictive controllers could not be used (Amos et al., 2018). In general, the creativity in choice of policy is greatly restricted.

Restrictions to model and covariance functions: The model is also limited by the requirement for analytic moment matching. This limits the covariance function of the GP to Gaussians, polynomials, etc. PILCO has so far only been used with the Gaussian kernel, which assumes smoothness of the dynamics—this assumption may not be good in robotics in many cases. This restriction is a severe limitation as the choice of model should be based on which model is the most accurate, and not dictated by some analytic properties of the function.

Analytic moment matching can be a poor approximation: In PILCO, the state-control distribution at each time step is estimated as a Gaussian distribution. This moment matching can lead to compounding errors, and cause the estimation of the state distribution to be poor. Figure 1.4 shows an example comparison between trajectories predicted through Monte Carlo sampling, and through moment matching in the cart-pole task. Even though the policy solves the task, and balances the pendulum, the moment matching prediction-errors compound and the predicted trajectory diverges. Such conservative predictions may lead to underperformance. Other researches have also found that moment matching can be inaccurate: Vinogradska et al. (2016) used quadrature for predictions, and found that it differs significantly from MM; and Kupcsik et al. (2014) used particles, and also found that MM is inaccurate. For example, if the task requires multi modal trajectory distributions, then it would be impossible to solve if one uses moment matching approximations.

Lack of principled exploration: Finally, an issue with PILCO is its lack of a principled approach to exploration. Currently PILCO simply optimizes the controller to achieve the highest possible expected reward. In some situations it can happen that the trajectory with the highest expected reward passes through regions which are already completely explored. If the controller does not explore new regions in the state space, it will not gain new information, which will prevent it from improving the controller. It would be natural to use the uncertainty of the Gaussian process models in an exploration scheme, where the policy would be promoted to seek regions in the state space with a large variance. However, this is difficult to do in the original PILCO framework, because the state distribution includes not just uncertainty from the

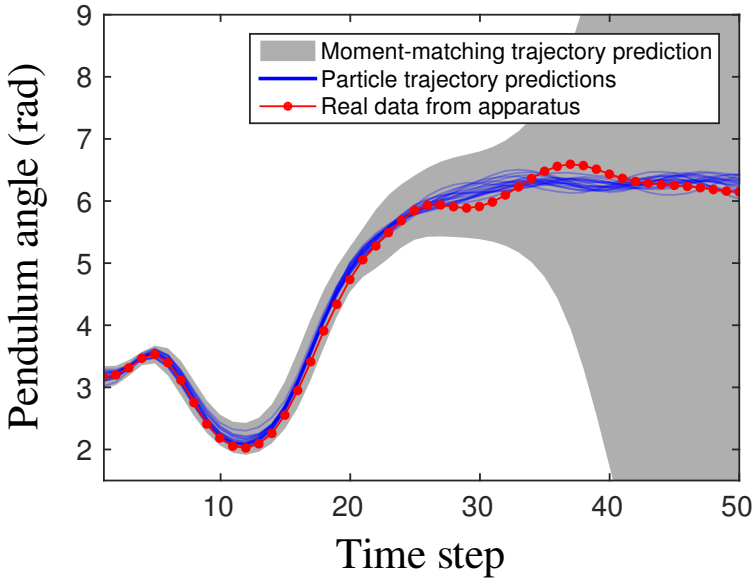


Figure 1.4: Moment-matching predictions can be vastly wrong. I compared the predicted trajectory distribution from moment matching against a Monte Carlo particle prediction for the same policy and model. The data was obtained from experiments on a real cart-pole apparatus while I was visiting Carl Rasmussen's lab at the University of Cambridge.

model predictions, but also the variance in the trajectory caused by the controller—for example even if the model were deterministic, a controller which inserts positive feedback could make an initial Gaussian distribution diverge arbitrarily in time. Increasing uncertainty in the state distribution may not be linked to exploring regions in state space where the model is uncertain, so it would not be meaningful exploration. The correct approach would be to decouple the different sources of uncertainty, and use only the model uncertainty to promote exploration. Such ideas were explored by McAllister et al. (2016); however, it is not straightforward with the analytic MM computations in PILCO.

Remark: Is moment matching really that bad?

Based on the above discussion, it seems that moment matching is a severe issue limiting what can be done with PILCO. To balance out this discussion, I present a few arguments for why moment matching may actually have been the key to PILCO's success. As previously argued by McHutchon (2014), in many robotics tasks it is probably not necessary to plan a trajectory distribution with different modes that are far apart (sometimes there can be multiple modes, such as when a pendulum can fall either left or right from the top balanced position, but the controller should not allow these modes to move far apart)—if the trajectory diverges, it means that the controller is probably not performing well. Enforcing unimodality may be providing a good prior to ease learning.
As another explanation: if multi modal distributions were allowed (e.g., the prediction is bimodal at each time step), then representing the trajectory distribution would require a number of parameters that is exponential in the length of the trajectory. This exponential growth in the number of parameters is halted by the moment matching, so it is not clear whether the moment matching is a problem, or whether it is actually an important component, without which learning would not be possible at the same data-efficiency

1.3.5 PILCO as a trajectory tracker

In this section I perform a derivation, which convinced me that moment matching is too much of a restriction: I show that properties of moment matching essentially imply that PILCO is learning linear trajectory tracking controllers. In particular, I show that a time-varying linear controller (a separate linear controller for each time step) can represent all achievable trajectories when moment matching is used. Basically, it comes down to the fact that the MM predictions of the next state depend on a Gaussian approximation of the state and control distribution at the current time step, and the only purpose of the policy is to simply set the parameters of the covariance at each time step. My claim can be stated more formally as:

Theorem 1 (PILCO is training trajectory trackers) *For every moment matching trajectory distribution prediction $p(\tau)$ achieved by a deterministic policy $\mathbf{u} = \pi(\mathbf{x};\theta)$, there exists a different time-varying affine policy $\mathbf{u} = \pi_{tracker}(\mathbf{x}, t;\theta) = A_t\mathbf{x}+\mu_t$, which achieves the exact same predicted trajectory distribution $p(\tau)$.*
Proof:

Let the control-state distribution at a time step be given as

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{u}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mu_x\\ \mu_u\end{bmatrix}, \begin{bmatrix}\Sigma_{xx} & s\\ s^T & \sigma_u^2\end{bmatrix}\right) \tag{1.1}$$

We can set s and σ_u to any values for which the covariance matrix stays a proper covariance matrix—it must be positive semi-definite. I will show that a linear controller can represent all of the admissible controllers (a Gaussian noise should be put on the

control as well to shift σ_u around, but I am not sure whether there is any advantage to such a non-deterministic controller in PILCO). The condition for positive semi-definiteness is equivalent to there being a Cholesky decomposition, and the proof follows easily from this idea. We can write

$$\Sigma = \begin{bmatrix} L_x & \mathbf{0} \\ l^T & n \end{bmatrix} \begin{bmatrix} L_x^T & l \\ \mathbf{0} & n \end{bmatrix} \tag{1.2}$$

where l and n are arbitrary parameters. We must find how these parameters relate to s and σ_u. A simple multiplication gives $s = L_x l$ and $\sigma_u^2 = n^2 + l^T l$.
Now, since we know the covariance, and the state, the control signal for a particular state can be found based on the conditional distribution $p(\mathbf{u}|\mathbf{x})$. For a Gaussian distribution we compute the mean of this conditional distribution as

$$\begin{aligned} \mathbf{u} &= \mu_u + s^T \Sigma_{xx}^{-1}(\mathbf{x} - \mu_x) \\ &= \left(\mu_u - s^T \Sigma_{xx}^{-1} \mu_x\right) + \left(s^T \Sigma_{xx}^{-1} \mathbf{x}\right) \end{aligned} \tag{1.3}$$

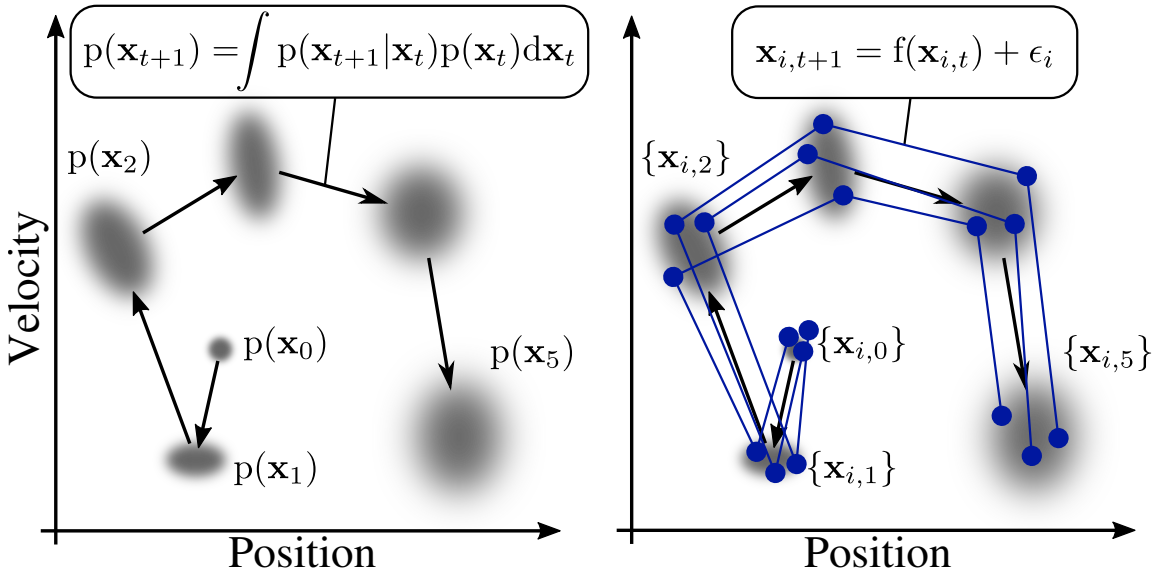
Notice that this is a linear controller $\mathbf{u} = A\mathbf{x} + \mathbf{a}$, where $A = s^T \Sigma_{xx}^{-1}$, and $\mathbf{a} = \mu_u - s^T \Sigma_{xx}^{-1} \mu_x$. Replacing s with $L_x l$ we see that there is a one to one mapping between $\mathbf{a}$ and l (because both Σ_s and L_s are positive-definite, hence invertible), and hence a time-varying affine controller can represent all admissible trajectory distributions, which concludes the proof.

Placing noise on the control action allows to also set n arbitrarily high. One could compute the variance of the controller as $\sigma^2 = \sigma_u^2 - s^T \Sigma_{xx}^{-1} s$. Replacing s with $L_x l$ and σ_u^2 with $n^2 + l^T l$, we see that $\sigma^2 = n^2$, and a deterministic controller corresponds to $n = 0$.

The control law is equivalent to a trajectory tracker: $\mathbf{u}_t = -A_t(\mathbf{z}_t - \mathbf{x}_t)$, where $-A_t \mathbf{z}_t = \mathbf{a}$, and $\mathbf{z}_t$ is the desired position. This derivation showed that PILCO is essentially a trajectory optimizer, and it is only learning how to track a trajectory, rather than learning the optimal actions in any given state. The most general form of the algorithm can be achieved by using a time-varying linear controller, but the number of parameters could be reduced by enforcing a shared controller between the different time steps. While this trajectory tracking property can be lessened by employing multiple start states in the optimization (Bischoff et al., 2013), the computational time of PILCO grows linearly with each new start location. As the computational time of PILCO is already long, this idea does not scale well.

1.3.6 How to overcome the challenges in PILCO?

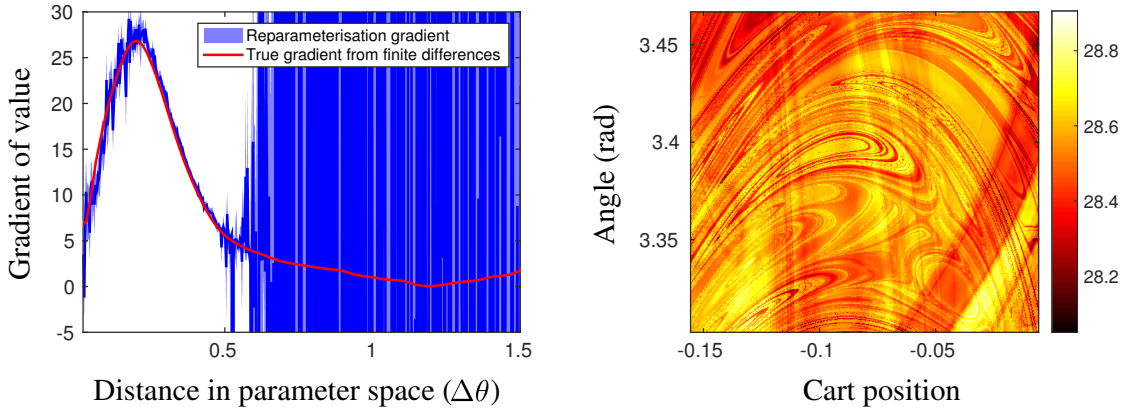
I explained that PILCO has several shortcomings, and is even conceptually flawed: see Section 1.3.5. All of these problems were linked to the moment matching requirement. A straightforward idea to overcome this problem would be to use Monte Carlo sampling instead of moment matching, which is illustrated in Figure 1.5. This has been attempted before in PILCO (McHutchon, 2014), and it did not work, with the poor performance attributed to local minima. In my thesis I found that the main problem was actually the method used for gradient computation. When using sampling it becomes necessary to differentiate through the stochasticities. It would be natural to



(a) Analytic approximation **(b)** Particle approximation

Figure 1.5: One can either perform extensive mathematical derivations to analytically predict an approximate trajectory distribution, as done in PILCO, or use a flexible particle approach to predict a stochastic approximation to the true trajectory distribution.

use reparameterization gradients, as was attempted by McHutchon (2014); however, I found that this did not work well. The problem is illustrated in Figure 1.6. There are regions in the policy parameter space, where the reparameterization gradient variance explodes (Fig. 1.6a), and this is caused by chaotic properties of the dynamics illustrated by fractal patterns in the input-output patterns (Fig. 1.6b). This kind of problem is inherent to the dynamics of the task. I discuss the problem in more detail in Chapter 4. The phenomenon highlighted that one has to consider which kinds of gradient estimators are suitable for which kinds of situations, and it is not sufficient to blindly apply backpropagation for computing gradients in every setting. In my thesis I started with this specific problem in model-based RL, but I stumbled on a more general question of appropriate gradient estimation. The next sections in my thesis will explain my new insights into gradient estimators, and how these lead towards solutions to the problem of poor gradients in model-based reinforcement learning. In Chapter 2 I will discuss gradient estimators through a single stochastic sampling operation, in Chapter 3 I will discuss how to put these gradient estimators together in a graph of computations, and finally in Chapter 4 I will return to this problem of gradients in model-based RL, and show that even in such highly erratic scenarios, well-behaved gradients can be estimated using my algorithms.



(a) Reparameterization gradients can fail **(b)** Chaotic properties in dynamics

Figure 1.6: Reparameterization gradients can have extremely erratic behavior in some regions of the policy parameter space (1.6a), and this is caused by inherent chaotic properties of the dynamics as illustrated by fractal input-output patterns (1.6b).

Chapter 2

Gradient estimators through a single sampling operation

The key task in policy gradient methods is estimating the gradient of an expectation $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x};\theta)}\left[\phi(\mathbf{x})\right]$ w.r.t. the parameters of the sampling distribution. In this section I discuss the foundations of estimators for such gradients in the setting when $\phi(\mathbf{x})$ is considered as a single step computation, i.e. we do not know what computations $\phi(\mathbf{x})$ performs internally; however, it may still be possible to query gradients of $\phi(\mathbf{x})$.

To give a concrete example of such algorithms, consider Evolution Strategies (ES) (Salimans et al., 2017). In ES, $\mathbf{x}$ are the policy parameters $\mathbf{w}$, and $\phi(\mathbf{w})$ is the fitness or the sum of rewards for one episode $\phi(\mathbf{w}) = \sum_{t=0}^{H} r(\mathbf{s}_t)$, where $\mathbf{s}$ is the state.[1] The algorithm samples a set of policy parameters $\{\mathbf{w}_i\}_{i=1}^{P}$, evaluates the performance of each $\mathbf{w}_i$, by running one episode with each $\mathbf{w}_i$, i.e. it computes $\phi(\mathbf{w}_i)$. Then, ES estimates the derivative of the expected fitness by using the likelihood ratio gradient, and optimizes by gradient ascent. As another example, the variational autoencoder (Kingma and Welling, 2013) estimates the derivative of the evidence lower bound by sampling and using the reparameterization gradient to differentiate through the stochastic operation.

In both of the above examples $\phi(\mathbf{x})$ may be deterministic, and the expectation is necessary only over a single stochastic computation $\mathbf{x} \sim p\left(\mathbf{x};\theta\right)$. Later, Chapter 3 will explain how the estimators introduced in this section can be combined in a graph of computations containing arbitrary stochastic and deterministic computations.

Previously (Sec. 1.1), I explained that the likelihood ratio and reparameterization tricks are two methods to estimate the required gradients. The LR gradient derivation was: $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{x\sim p(x;\theta)}\left[\phi(x)\right] = \int \frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}\phi(x)\mathrm{d}x = \int p\left(x;\theta\right)\frac{\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}}{p(x;\theta)}\phi(x)\mathrm{d}x = \int p\left(x;\theta\right)\frac{\mathrm{d}\log p(x;\theta)}{\mathrm{d}\theta}\phi(x)\mathrm{d}x = \mathbb{E}_{x\sim p(x;\theta)}\left[\frac{\mathrm{d}\log p(x;\theta)}{\mathrm{d}\theta}\phi(x)\right]$. On the other hand, the RP gradient was derived by defining a mapping $g(\epsilon;\theta) = x$, where ϵ comes from a fixed simple distribution, but x behaves as a sample from the desired distribution. For example, for a Gaussian distribution, $g(\epsilon;\theta) = \mu + \sigma\epsilon$, where $\epsilon \sim \mathcal{N}(0,1)$ and $\theta = [\mu,\sigma]$, then the RP gradient becomes $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{x\sim p(x;\theta)}\left[\phi(x)\right] = \frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}\left[\phi\left(g(\epsilon)\right)\right] = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}\left[\frac{\mathrm{d}\phi(g(\epsilon))}{\mathrm{d}\theta}\right] = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}\left[\frac{\mathrm{d}g}{\mathrm{d}\theta}\frac{\mathrm{d}\phi(g(\epsilon))}{\mathrm{d}g}\right]$. In this equation, $\frac{\mathrm{d}g}{\mathrm{d}\mu} = 1$, $\frac{\mathrm{d}g}{\mathrm{d}\sigma} = \epsilon$ and $\frac{\mathrm{d}\phi(g(\epsilon))}{\mathrm{d}g} = \frac{\mathrm{d}\phi(x)}{\mathrm{d}x}$.

[1] Note that the states $\mathbf{s}$ may also be sampled from some initial distribution, but this just manifests as additional noise in $\phi(\mathbf{w})$.

Such typical derivations are just mathematical tricks, and do not explain the mechanism behind the estimators. What is the meaning of these estimators? In this chapter, I give a physical interpretation of these two tricks leading to an improved insight and point towards methods to reduce LR gradient variance by importance sampling from an optimal distribution.[2]

Reducing the variance of these gradient estimates is a central problem in this line of work, because more accurate gradients lead to faster optimization. In addition to my discussion about importance sampling, I also discuss prior methods of gradient variance reduction, such as the use of control variates, known as baselines in the RL literature (Greensmith et al., 2004). I give a new analysis, and explain that simply trying to estimate the optimal baseline (Weaver and Tao, 2001) from the samples is not the optimal use of the samples for variance reduction, and attempt to derive better methods, which end up giving a slight improvement.

The chapter begins by explaining new interpretations of LR and RP estimators (Sec. 2.1), then explains optimal importance sampling schemes to reduce LR gradient variance (Sec. 2.2), evaluates importance sampling in experiments (Sec. 2.3) and finally discusses baseline techniques (Sec. 2.4). The reader only interested in importance sampling (Sec. 2.2) does not need to read the interpretations in Section 2.1 as the theory is not strictly necessary, though I think it adds intuition and was my primary motivation for deriving the importance sampling methods.

2.1 Interpretations of LR and RP gradients

2.1.1 A probability "boxes" view of LR and RP gradients

Here I give the first of my two explanations of the link between LR and RP gradients. The explanation relies on a first principles thinking about the effect that changing the parameters of a probability distribution θ has on infinitesimal "boxes" of probability mass (Fig. 2.1). Both LR and RP are trying to estimate $\frac{\mathrm{d}}{\mathrm{d}\theta}\int p\left(x;\theta\right)\phi(x)\mathrm{d}x$. A typical finite explanation of Riemann integrals is performed by discretizing the integrand into "boxes" of size Δx, and summing:

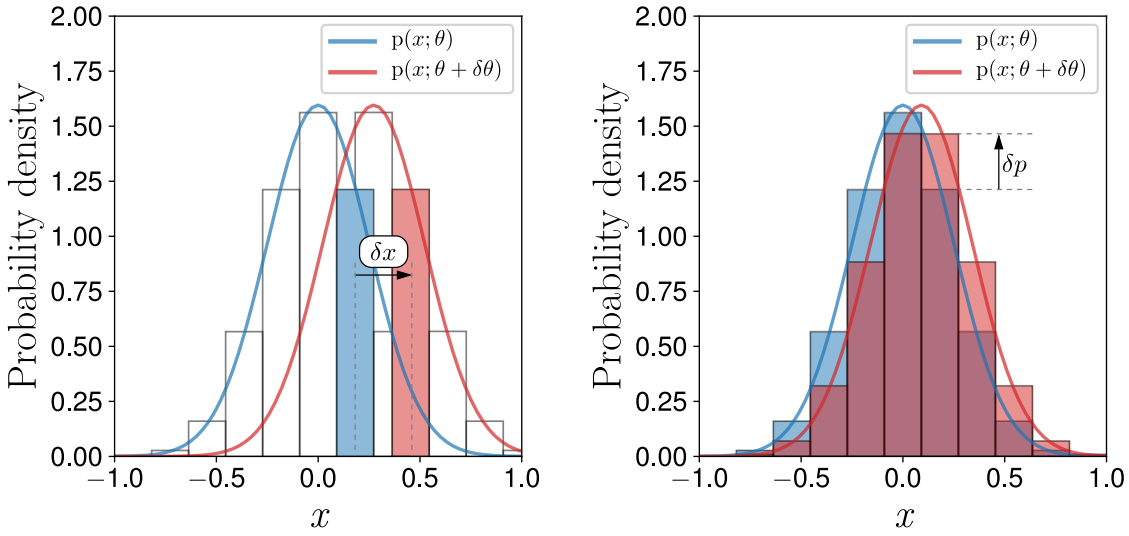
$$\frac{\mathrm{d}}{\mathrm{d}\theta}\sum_{i=1}^{N} p\left(x_i;\theta\right)\Delta x_i\phi(x_i). \tag{2.1}$$

Taking the limit as $N\to\infty$ recovers the true integral. In this equation,

$$\Delta p_i = p\left(x_i;\theta\right)\Delta x_i \tag{2.2}$$

is the amount of probability mass inside the "box", and $\phi(x_i)$ is the function value inside the "box". A finite approximation of the derivative w.r.t. θ can be performed by perturbing by $\delta\theta$, and estimating the change in the integral. When such a perturbation is performed, depending on how the "boxes" are defined, we will end up with either the RP gradient or the LR gradient.

[2]Note that my method differs from the typical use of importance sampling, where it is used for reusing samples from other policies. I instead modify the sampling distribution to reduce the gradient variance.



(a) RP box **(b)** LR box

Figure 2.1: Comparing what LR and RP views do to the probability boxes.

RP: Such a view can be used to explain RP gradients. In this case, the boundaries of the "box" are fixed with reference to the shape of the probability distribution, i.e. for each i. I define the center of the box as $x_i = g(\epsilon_i; \theta)$, and the boundaries as $g(\epsilon_i \pm \Delta\epsilon/2; \theta)$, where ϵ_i is the reference position on a fixed simple distribution $p(\epsilon)$. Now, note that as θ is perturbed, the amount of probability mass assigned to each "box" stays fixed at

$$\Delta p_i = p(\epsilon)\, \Delta\epsilon \; ; \tag{2.3}$$

however, the center of the "box" moves, so the function value $\phi(x_i)$ inside each "box" changes by

$$\delta\phi_i = \phi\left(g(\epsilon_i; \theta + \delta\theta)\right) - \phi\left(g(\epsilon_i; \theta)\right) = \phi(x_i + \delta x_i) - \phi(x_i). \tag{2.4}$$

The full derivative can then be expressed as

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{x\sim p(x;\theta)}\left[\phi(x)\right] \approx \frac{1}{\delta\theta}\sum_{i=1}^{N}\Delta p_i \delta\phi_i = \sum_{i=1}^{N}\Delta p_i \frac{\delta\phi_i}{\delta x_i}\frac{\delta x_i}{\delta\theta}. \tag{2.5}$$

Taking the infinitesimal limit as $N \to \infty$, and noting that $\Delta p_i = p(x_i; \theta)\, \Delta x_i$, we obtain the RP gradient estimator $\int p(x; \theta) \frac{\mathrm{d}\phi(x)}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}\theta}\, \mathrm{d}x$. We see that the RP gradient estimator essentially estimates the gradient by keeping the probability mass inside each "box" fixed, but estimating how the function value ϕ inside the "box" changes as the parameters θ are perturbed.

LR: The LR gradient, on the other hand, keeps the boundaries of the "boxes" fixed, i.e. the center of the box is at x_i, and the boundaries at $x_i \pm \Delta x_i/2$. Now, as the boundaries are independent of θ, the function value $\phi(x_i)$ inside the box stays fixed,

even as θ is perturbed by $\delta\theta$; however, the probability mass inside the box changes, because the density changes by

$$\delta p_i = p\left(x_i; \theta + \delta\theta\right) - p\left(x_i; \theta\right). \tag{2.6}$$

The full derivative can be expressed as

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{x\sim p(x;\theta)}\left[\phi(x)\right] \approx \frac{1}{\delta\theta}\sum_{i=1}^{N}\Delta x_i \delta p_i \phi(x_i) = \sum_{i=1}^{N} p\left(x_i;\theta\right)\Delta x_i \frac{\delta p_i/\delta\theta}{p\left(x_i;\theta\right)}\phi(x_i). \tag{2.7}$$

Where we have multiplied and divided by $p\left(x_i;\theta\right)$. Taking the infinitesimal limit recovers the LR gradient $\int p\left(x;\theta\right)\frac{\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}}{p(x;\theta)}\phi(x)\ \mathrm{d}x = \mathbb{E}_{x\sim p(x;\theta)}\left[\frac{\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}}{p(x;\theta)}\phi(x)\right]$. The transformation $p\left(x;\theta\right)\frac{\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}}{p(x;\theta)} = p\left(x;\theta\right)\frac{\mathrm{d}\log p(x;\theta)}{\mathrm{d}\theta}$ is known as the log-derivative trick, and it may appear to be the essence behind the LR gradient, but actually the multiplication and division by $p(x;\theta)$ is just a special case of the more general Monte Carlo integration principle. Any integral $\int f(x)\ \mathrm{d}x$ can be approximated by sampling from a distribution $q(x)$ as $\int f(x)\ \mathrm{d}x = \int q(x)\frac{f(x)}{q(x)}\ \mathrm{d}x = \mathbb{E}_{x\sim q(x)}\left[\frac{f(x)}{q(x)}\right]$. Rather than thinking of the LR gradient in terms of the log-derivative term, we think it is better to think of it as simply estimating the integral $\int \frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}\phi(x)\ \mathrm{d}x$ by applying the appropriate importance weights to samples from $p(x;\theta)$. Thus, we see that in the discretized case, the LR gradient picks $q(x) = p\left(x;\theta\right)$ (Jie and Abbeel, 2010) and performs Monte Carlo integration to approximate the integral $\frac{1}{\delta\theta}\sum_{i=1}^{N}\Delta x_i\delta p_i\phi(x_i)$ by sampling from $P(x_i) = \Delta x_i p\left(x_i;\theta\right)$. To summarize: LR estimates the gradient by keeping the boundaries of the boxes fixed, measuring the change in probability mass in each box, and weighting by the function value in the box: $\phi(x_i)\delta p$.

Sometimes, the LR gradient is described as being "kind of like a finite difference gradient" (Salimans et al., 2017; Mania et al., 2018), but here we see that it is a different concept, which does not rely on fitting a straight line between differences of ϕ, but estimates how probability mass is reallocated among different ϕ values via Monte Carlo integration by sampling from $p\left(x;\theta\right)$.

To highlight this point further, I show the concrete equations for the ES gradient estimator, which is a type of LR gradient used in evolutionary computation. The ES gradient estimator uses a Gaussian $p\left(x;\theta\right)$ and estimates $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{x\sim p(x;\theta)}\left[\phi(x)\right] = \mathbb{E}_{x\sim p(x;\theta)}\left[\frac{\mathrm{d}\log p(x;\theta)}{\mathrm{d}\theta}\phi(x)\right]$. The gradient estimator is derived as

$$\begin{aligned}
\log p\left(x;\theta\right) &= -\frac{1}{2}\log(2\pi) - \log(\sigma) - \frac{(x-\mu)^2}{2\sigma^2}, \\
\frac{\mathrm{d}\log p\left(x;\theta\right)}{\mathrm{d}\mu} &= \frac{x-\mu}{\sigma^2} = \frac{\epsilon}{\sigma}, \\
&\text{where} \quad x = \mu + \epsilon\sigma \text{ and } \epsilon \sim \mathcal{N}(0,1).
\end{aligned} \tag{2.8}$$

ES uses antithetic sampling, i.e. it samples points x in pairs opposite to each other, s.t. $x_+ = \mu + \sigma\epsilon$ and $x_- = \mu - \sigma\epsilon$. The gradient estimator w.r.t. the mean parameter averaged across the two samples becomes $\frac{1}{2}\left(\frac{\epsilon}{\sigma}\phi(x_+) + \frac{(-\epsilon)}{\sigma}\phi(x_-)\right)$, which is

$$\frac{\epsilon\left(\phi(x_+) - \phi(x_-)\right)}{2\sigma}. \tag{2.9}$$

Finite difference methods, on the other hand estimate the derivative by estimating the change in ϕ:

$$\frac{\mathrm{d}\phi(x)}{\mathrm{d}x} \approx \frac{\phi(x_+) - \phi(x_-)}{\Delta x}. \tag{2.10}$$

In the antithetic sampling case, $\Delta x = 2\sigma\epsilon$, so the estimator is

$$\frac{\mathrm{d}\phi(x)}{\mathrm{d}x} \approx \frac{\phi(x_+) - \phi(x_-)}{2\sigma\epsilon}. \tag{2.11}$$

This result is clearly different from the LR gradient estimator in Equation (2.9), as the ϵ is in the wrong place.

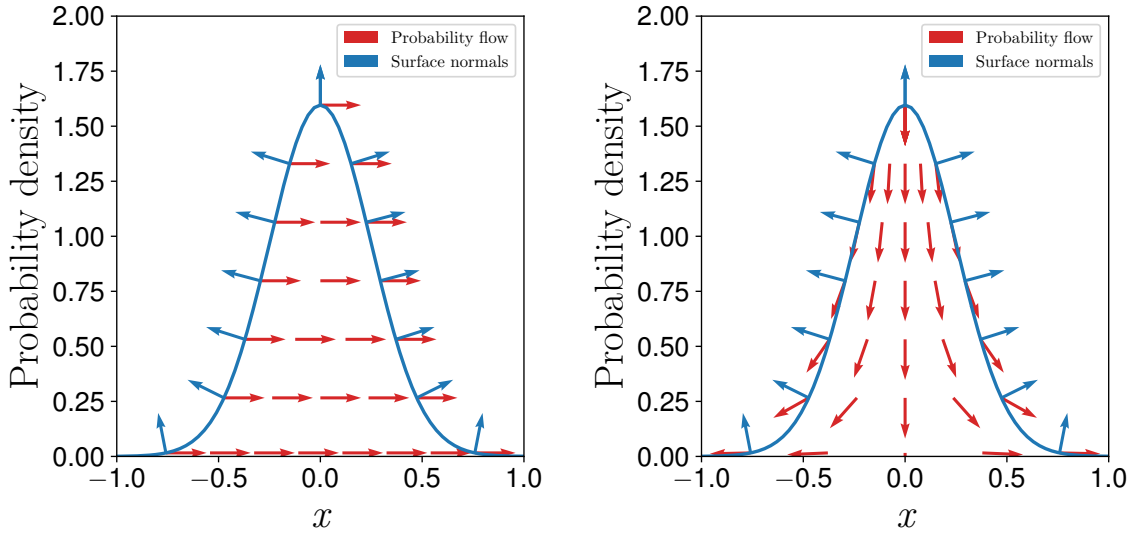
To highlight the difference further, note that if ϕ is truly linear $\phi(x) = ax$, then the finite difference gradient will always exactly estimate the derivative as a; however, the LR gradient has variance $\mathbb{V}\left[\epsilon\frac{2a\epsilon\sigma}{2\sigma}\right] = \mathbb{V}\left[\epsilon^2 a\right] = \mathbb{E}\left[\epsilon^4 a^2\right] = 3a^2$, which is non-zero. While certainly, the likelihood ratio and finite difference gradient estimators resemble each other, as both use the function values $\phi(x)$ to estimate the derivative, the mechanism they use to perform this estimation is completely different, and I hope that my explanation could lead to better gradient estimators combining the good properties of both.

2.1.2 A unified probability flow view of LR and RP gradients

In this section, I give my second explanation of LR and RP gradients. The appeal of this theory is that both LR and RP gradients come out of the same derivation, thus showing a link between the two. In particular, I define a virtual incompressible flow of probability mass imposed by perturbing the parameters of the probability distribution, which can be used to express the derivative of the expectation as an integral over this flow. LR and RP gradient estimators turn out to correspond to duals of this integral under Stokes' theorem.[3] See Appendix B for basics about vector calculus and fluid mechanics. Note that a similar theory was proposed by Jankowiak and Obermeyer (2018), but their derivation and motivation are different. They focused on deriving new RP gradient estimators, whereas I focused on LR gradient estimators, and the duality between RP and LR. Their derivation does not perform a height reparameterization (explained next), and instead requires that the flow continuity equation is satisfied. In general I think that their derivation is quite sleek and practical in terms of deriving new RP gradients; however, their boundary conditions can be confusing for non-physicists. My derivation is more visual, gives a bit more insight, and has good symbiosis with the importance sampling methods that I explain later in this chapter.

The main idea is similar to the RP trick, but in addition to reparameterizing the sampled $\mathbf{x}$ location, I sample a height h from the uniform distribution for each point: $h = \epsilon_h p\left(\mathbf{x};\theta\right)$, where $\epsilon_h \sim \mathrm{unif}(0,1)$. Thus, the sampling space is extended with an additional dimension for the height $\tilde{\mathbf{x}} = [\mathbf{x}^T, h]^T$, and I am uniformly sampling in the volume under the curve defined by $p\left(\mathbf{x};\theta\right)$. The transformation $g(\epsilon_x)$ is redefined to

[3] Stokes' theorem is the high-dimensional generalization of the divergence theorem, often used in electrodynamics and fluid mechanics. It relates the flux in and out of a volume to the change in density inside the volume.



(a) μ flow lines

(b) σ flow lines

Figure 2.2: Probability flow lines when μ and σ are perturbed.

$g(\epsilon_x, \epsilon_h) = [g(\epsilon_x)^T, \epsilon_h p(\mathbf{x}; \theta)]^T = \tilde{\mathbf{x}}$. Moreover, $\phi(\tilde{\mathbf{x}}) := \phi(\mathbf{x})$ does not depend on the height h. The expectation turns into:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\theta} \int p(\mathbf{x}; \theta)\, \phi(\mathbf{x}) \mathrm{d}\mathbf{x} &= \frac{\mathrm{d}}{\mathrm{d}\theta} \int_{\epsilon_x} \int_{\epsilon_h} p(\epsilon_x)\, p(\epsilon_h)\, \phi(g(\epsilon_x, \epsilon_h))\, \mathrm{d}\epsilon_x \mathrm{d}\epsilon_h \\ &= \int_V \nabla_{\tilde{\mathbf{x}}} \phi(\tilde{\mathbf{x}}) \nabla_\theta g(\epsilon_x, \epsilon_h) \mathrm{d}V. \end{aligned} \tag{2.12}$$

In Equation (2.12), V is the volume under the curve. The method somewhat resembles rejection sampling. The expectation in the reparameterized integral is uniformly sampling under the curve, so another way to express the probability density is $1/V$, where V is the total volume. However, as the volume is determined by a probability distribution, then $V = 1$, so the expectation is just the integral in the second line. Each column i of $\nabla_\theta g(\epsilon_x, \epsilon_h)$ corresponds to a vector field induced by perturbing the i^{th} component of θ. The red lines in Figure 2.2 show the induced flow fields for a Gaussian distribution as the mean and variance are perturbed. The other member of the integral $\nabla_{\tilde{\mathbf{x}}} \phi(\tilde{\mathbf{x}})$ is the grad of the scalar field $\phi(\tilde{\mathbf{x}})$. As ϕ does not depend on the height h, the grad will always be parallel to the $\mathbf{x}$ axes with magnitude $\frac{\mathrm{d}\phi}{\mathrm{d}\mathbf{x}}$.

According to Stokes' theorem, the *volume integral* in Equation (2.12) can be turned into a *surface integral* over the boundary of the volume $\mathbf{S}$ (I use $\mathrm{d}\mathbf{S}$ as a shorthand for $\hat{\mathbf{n}}\mathrm{d}S$, where $\hat{\mathbf{n}}$ is the surface normal vector). Stokes' theorem states:

$$\int_V \nabla \cdot \mathbf{F} \mathrm{d}V = \int_S \mathbf{F} \cdot \mathrm{d}\mathbf{S}. \tag{2.13}$$

In this equation, $\mathbf{F}$ is any vector field. A common corollary arises by picking $\mathbf{F} = \phi\mathbf{v}$, where ϕ is a scalar field, and $\mathbf{v}$ is a vector field. I choose $\mathbf{v} = \nabla_\theta g(\epsilon_x, \epsilon_h)\delta\theta$, where $\delta\theta$ is an arbitrary perturbation in θ, so that $\mathbf{F} = \phi(\tilde{\mathbf{x}})\nabla_\theta g(\epsilon_x, \epsilon_h)\delta\theta$, in which

case $\nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{F} = \nabla_{\tilde{\mathbf{x}}} \cdot (\phi(\tilde{\mathbf{x}}) \nabla_\theta g(\epsilon_x, \epsilon_h) \delta\theta) = \nabla_{\tilde{\mathbf{x}}} \phi(\tilde{\mathbf{x}}) \nabla_\theta g(\epsilon_x, \epsilon_h) \delta\theta + \phi(\tilde{\mathbf{x}}) \nabla_{\tilde{\mathbf{x}}} \cdot \nabla_\theta g(\epsilon_x, \epsilon_h) \delta\theta$. Note that the term $\nabla_\theta g(\epsilon_x, \epsilon_h)\delta\theta$ corresponds to an incompressible flow (because the probability density does not change at any point in the augmented space). As the div of an incompressible flow is 0, then $\nabla_{\tilde{\mathbf{x}}} \cdot \nabla_\theta g(\epsilon_x, \epsilon_h)\delta\theta = 0$, and the second term disappears. Noting that $\delta\theta$ can be cancelled, because it is arbitrary, we are left with the equation:

$$\int_V \nabla_{\tilde{\mathbf{x}}} \phi(\tilde{\mathbf{x}}) \nabla_\theta g(\epsilon_x, \epsilon_h) \mathrm{d}V = \int_S \phi(\tilde{\mathbf{x}}) \nabla_\theta g(\epsilon_x, \epsilon_h) \mathrm{d}\mathbf{S}. \tag{2.14}$$

Now I explain how the left-hand side of Equation (2.14) gives rise to the RP gradient estimator, while the right-hand side corresponds to the LR gradient estimator. The RP estimator follows quite easily, while the LR estimator is slightly more involved.

RP estimator: Consider the $\nabla_{\tilde{\mathbf{x}}} \phi(\tilde{\mathbf{x}}) \nabla_\theta g(\epsilon_x, \epsilon_h)$ term. As the scalar field $\phi(\tilde{\mathbf{x}})$ is independent of the height location h, the component of the grad in that direction is 0, and $\phi(\tilde{\mathbf{x}}) = \phi(\mathbf{x})$. As the h-component is 0, then the value of g in the h-direction is multiplied by 0, and is irrelevant for the product, so $\nabla_{\tilde{\mathbf{x}}} \phi(\tilde{\mathbf{x}}) \nabla_\theta g(\epsilon_x, \epsilon_h) = \nabla_{\mathbf{x}} \phi(\mathbf{x}) \nabla_\theta g(\epsilon_x)$, which is just the term used in the RP gradient estimator. Hence, the left-hand side of Equation (2.14) corresponds to the RP gradient.

LR estimator: I will show that the LR estimator tries to integrate $\int_S \phi(\tilde{\mathbf{x}}) \nabla_\theta g(\epsilon_x, \epsilon_h) \mathrm{d}\mathbf{S}$. First, note that $\mathrm{d}\mathbf{S} = \hat{\mathbf{n}} \mathrm{d}S$, and it is necessary to express the normalized surface vector $\hat{\mathbf{n}}$. A sufficient condition for the normal vector is that it should be perpendicular to two tangent vectors. I will find the vector perpendicular to two tangent vectors, one that points downhill, and another one that points along the contour line. To do so, I first express the tangent vector downhill $\mathbf{t}$, then change the height component of this vector to obtain a vector perpendicular to the tangent vector. This vector will also be perpendicular to the tangent vector along the contour by construction, and hence, it will be the normal vector.

The vector tangent and downhill to the surface is given by $\mathbf{t} = [-\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}, -\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)^T]$. The normal vector $\mathbf{n}$ is $[-\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}; h]$, such that $\mathbf{t} \cdot \mathbf{n} = 0$. Therefore, h must be such that $\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)^T - \left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)^T h = 0 \quad \Rightarrow \quad h = 1$. Finally, we normalize the vector:

$$\hat{\mathbf{n}} = {}^{[-\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}, 1]} \Big/ \sqrt{\left(\tfrac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)\left(\tfrac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)^T + 1}. \tag{2.15}$$

Next, we perform a change of coordinates from the surface elements $\mathrm{d}S$ to cartesian coordinates $\mathrm{d}\mathbf{x}$. When projecting a surface element $\mathrm{d}S$ with unit normal $\hat{\mathbf{n}}$ to a plane with unit normal $\hat{\mathbf{m}}$, the projected area is given by $\mathrm{d}\mathbf{x} = |\hat{\mathbf{n}} \cdot \hat{\mathbf{m}}| \, \mathrm{d}S$, therefore we need $\mathrm{d}\mathbf{x} = \mathrm{d}S \left| \frac{1}{\sqrt{\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)^T + 1}} [-\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}, 1] \cdot [\mathbf{0}, 1] \right| = {}^{\mathrm{d}S} \Big/ \sqrt{\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)^T + 1}$, from which we get

$$\mathrm{d}S = \sqrt{\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)^T + 1} \;\; \mathrm{d}\mathbf{x}. \tag{2.16}$$

Plugging Equations (2.15) and (2.16) into the right-hand side of Equation (2.14) we get

$$\int_X \phi(\tilde{\mathbf{x}}) \nabla_\theta g(\epsilon_x, \epsilon_h) \cdot \frac{[-\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}, 1]}{\sqrt{\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)^T + 1}} \sqrt{\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)\left(\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}\right)^T + 1} \ \mathrm{d}\mathbf{x} = \int_X \phi(\tilde{\mathbf{x}}) \nabla_\theta g(\epsilon_x, \epsilon_h) \cdot \left[-\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}, 1\right] \ \mathrm{d}\mathbf{x} \tag{2.17}$$

Recall that the last element of $g(\epsilon_x, \epsilon_h)$ is $\epsilon_h p\left(g(\epsilon_x); \theta\right)$, and that ϵ_h at the boundary surface is 1, then the $\nabla_\theta g(\epsilon_x, \epsilon_h) \cdot [-\frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}, 1]$ term turns into $-\nabla_\theta g(\epsilon_x) \cdot \frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}} + \frac{\partial \epsilon_h p(g(\epsilon_x);\theta)}{\partial \theta}\Big|_{\epsilon_x = const, \epsilon_h = 1}$. The last term $\frac{\partial p(g(\epsilon_x);\theta)}{\partial \theta}\Big|_{\epsilon_x = const}$ can be thought of as the rate of change of the probability density while following a point moving in the flow induced by perturbing θ. This quantity can be expressed with the material derivative $\frac{\partial p(g(\epsilon_x);\theta)}{\partial \theta}\Big|_{\epsilon_x = const} = \frac{\mathrm{d}p(\mathbf{x};\theta)}{\mathrm{d}\theta} + \nabla_\theta g(\epsilon_x) \cdot \frac{\mathrm{d}p}{\mathrm{d}\mathbf{x}}$. Finally, substituting into Equation (2.17):

$$\int_S \phi(\tilde{\mathbf{x}}) \nabla_\theta g(\epsilon_x, \epsilon_h) \ \mathrm{d}\mathbf{S} = \int_X \phi(\mathbf{x}) \frac{\mathrm{d}p\left(\mathbf{x}; \theta\right)}{\mathrm{d}\theta} \ \mathrm{d}\mathbf{x} \tag{2.18}$$

We have already seen that if one performs a Monte Carlo integration of the right-hand side of Equation (2.18) using samples from $p\left(\mathbf{x}; \theta\right)$, this gives rise to the LR gradient estimator. Thus, the RP gradient estimator and the LR gradient estimator are duals under Stokes' theorem. To further strengthen this claim we prove that the LR gradient estimator is the unique estimator that takes weighted averages of the function values $\phi(\mathbf{x})$.

Theorem 2 (Uniqueness of LR gradient estimator) *$g = p\left(\mathbf{x}; \theta\right) \frac{\mathrm{d} \log p(\mathbf{x};\theta)}{\mathrm{d}\theta}$ is the unique function g, s.t. $\int g(\mathbf{x})\phi(\mathbf{x}) \ \mathrm{d}\mathbf{x} = \frac{\mathrm{d}}{\mathrm{d}\theta} \int p\left(\mathbf{x}; \theta\right) \phi(\mathbf{x}) \ \mathrm{d}\mathbf{x}$ for any $\phi(\mathbf{x})$.*

Proof: Suppose there exist $g(\mathbf{x})$ and $q(\mathbf{x})$, s.t. $\int \phi(\mathbf{x}) g(\mathbf{x}) \ \mathrm{d}\mathbf{x} = \int \phi(\mathbf{x}) q(\mathbf{x}) \ \mathrm{d}\mathbf{x}$ for any $\phi(\mathbf{x})$. Rearrange the equation into $\int \phi(\mathbf{x}) \left(g(\mathbf{x}) - q(\mathbf{x})\right) \ \mathrm{d}\mathbf{x} = 0$, then pick $\phi(\mathbf{x}) = g(\mathbf{x}) - q(\mathbf{x})$ from which we get $\int \left(g(\mathbf{x}) - q(\mathbf{x})\right)^2 \ \mathrm{d}\mathbf{x} = 0$. Therefore $g = q$. Q.E.D.

From this result, we see that Equation (2.18) was immediately clear by inspection without having to go through the calculation, because we knew that it must equal $\frac{\mathrm{d}}{\mathrm{d}\theta} \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x};\theta)} \left[\phi(\mathbf{x})\right]$ by the construction based on reparameterization.

What happens if we perform the same kind of analysis for the RP gradient? Similarly, suppose that there exist $u(\mathbf{x})$ and $v(\mathbf{x})$, s.t. $\int \nabla\phi(\mathbf{x}) \cdot u(\mathbf{x}) \ \mathrm{d}\mathbf{x} = \int \nabla\phi(\mathbf{x}) \cdot v(\mathbf{x}) \ \mathrm{d}\mathbf{x}$ for any $\phi(\mathbf{x})$. Rearrange the equation into $\int \nabla\phi(\mathbf{x}) \cdot \left(u(\mathbf{x}) - v(\mathbf{x})\right) \ \mathrm{d}\mathbf{x} = 0$. Then, if we can pick $\nabla\phi(\mathbf{x}) = u(\mathbf{x}) - v(\mathbf{x})$ it would lead to $u = v$, which would show the uniqueness. However, it is not necessarily possible to pick such a $\phi(\mathbf{x})$. In particular, the integral of $\nabla\phi(\mathbf{x})$ over any closed path is 0, but this is not necessarily the case for $u - v$. Therefore, the same kind of analysis does not lead to a claim of uniqueness. Indeed, concurrent work (Jankowiak and Obermeyer, 2018) showed that there are an

infinite amount of possible reparameterization gradients, and the minimum variance is achieved by the optimal transport flow.[4]

2.2 Importance sampling for gradient estimators

2.2.1 Slice integral importance sampling

From Theorem 2 we saw that unlike the RP gradient case, the weighting g for function values $\phi(\mathbf{x})$ with $\mathbf{x} \sim p(\mathbf{x};\theta)$ to obtain an unbiased estimator for the gradient $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}[\phi(\mathbf{x})]$ is unique. The only option to reduce the variance by changing the weighting would then be to sample from a different distribution $q(\mathbf{x};\theta)$ via importance sampling. Motivated by the resemblance of the "boxes" theory in Section 2.1.1 to the Riemann integral, I propose to sample horizontal slices of probability mass resembling the Lebesgue integral. Such an approach appears attractive, because if the location of the slice is moved by modifying the parameters of the distribution (e.g., by changing the mean), then the derivative of the expected value of the integral over the slice will depend only on the value at the edges of the slice (because the probability density in the middle would not change). To obtain the gradient estimator, it will only be necessary to compute the probability density $p_L(\mathbf{x};\theta)$. I derive such a "slice integral" distribution corresponding to the Gaussian distribution. I call the new distribution the L-distribution, and it is plotted in Figure 2.3b. I chose the naming based on the first letter of Lebesgue due to the resemblance to the Lebesgue integral.

Derivation of the pdf of the L-distribution: One way to sample whole slices of a probability distribution would be to sample a height h between 0 and p_{max} proportionally to the probability mass at that height. The probability mass at a height h is just given by $2|x-\mu|$ where x is such that $p(x;\mu,\sigma) = h$, i.e. $2|x-\mu|$ is the distance between the edges of $p(x;\mu,\sigma)$. The probability mass corresponding to x is then given by $2|x-\mu|\mathrm{d}h$. Performing a change of coordinates to the x-domain, and splitting the mass between the two edges of the slice, we get $|x-\mu|\mathrm{d}h = |x-\mu|\left|\frac{\mathrm{d}p(x;\mu,\sigma)}{\mathrm{d}x}\right|\mathrm{d}x$. This gives a closed-form normalized pdf for the L-distribution:

$$\begin{aligned} p_L(x;\mu,\sigma) = |x-\mu|\left|\frac{\mathrm{d}p(x;\mu,\sigma)}{\mathrm{d}x}\right| &= |x-\mu|p(x;\mu,\sigma)\frac{|x-\mu|}{\sigma^2} \\ &= \frac{|x-\mu|^2}{\sqrt{2\pi}\,\sigma^3}\exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \end{aligned} \tag{2.19}$$

One can recognize that Equation (2.19) is actually just a Maxwell-Boltzmann distribution reflected about the origin with the probability mass split between the two sides.

Sampling from the L-distribution: To sample from this distribution, it is necessary to sample points proportionally to the length of the slices. It suffices to sample

[4] By minimum variance, I mean the minimum variance achievable if one does not assume any knowledge of $\phi(\mathbf{x})$, or alternatively that $\nabla\phi(\mathbf{x}) \approx \mathbf{1}$.

uniformly in the area under the curve in the space augmented with the height dimension h, then select the slice on which the sampled point lies. This can be achieved with the three steps: 1) sample a point from the base distribution: $x_s \sim p\left(x;\mu,\sigma\right)$, 2) sample a height: $h \sim \text{unif}\left(0, p\left(x_s;\mu,\sigma\right)\right)$, 3) sample x from one of the two edges of the slice at height h, i.e. sample from the set $p^{-1}(h;\mu,\sigma) = \{x : p\left(x;\mu,\sigma\right) = h\}$, where $p^{-1}(h)$ inverts the pdf, and computes the set of x that give a probability density h. For the L-distribution, this can be achieved by sampling $\epsilon_x \sim \mathcal{N}(0,1)$ and $\epsilon_h \sim \text{unif}(0,1)$ and transforming these by the equation:

$$x = \mu \pm \sigma\sqrt{-2\log(\epsilon_h) + \epsilon_x^2} \tag{2.20}$$

Derivation of sampling method for L-distribution: I first derive the inverse of the probability density $p^{-1}(h)$ as

$$\begin{aligned} h &= \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \\ \log(h) &= -\frac{1}{2}\log(2\pi) - \log(\sigma) - \frac{(x-\mu)^2}{2\sigma^2} \\ (x-\mu)^2 &= -2\sigma^2\left(\frac{1}{2}\log(2\pi) + \log(\sigma) + \log(h)\right) \\ x &= \mu \pm \sigma\sqrt{-\log(2\pi) - 2\log(\sigma) - 2\log(h)}\ . \end{aligned} \tag{2.21}$$

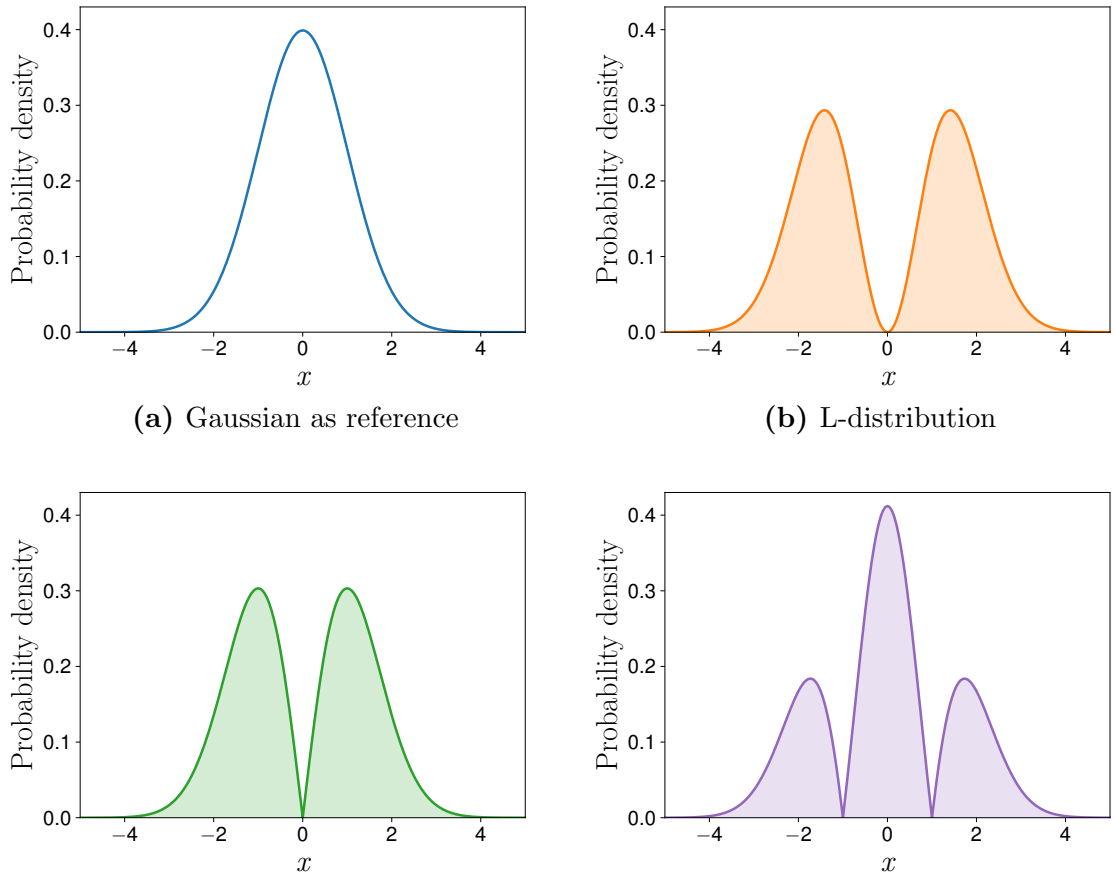
Now, noting $h = p(x_s)\epsilon_h$, where $\epsilon_h \sim \text{unif}(0,1)$ and $x_s = \mu + \sigma\epsilon_x, \quad \epsilon_x \sim \mathcal{N}(0,1)$, we end up with the sampling method:

$$\begin{aligned} x &= \mu \pm \sigma\sqrt{-\log(2\pi) - 2\log(\sigma) - 2\log\left(\frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{\sigma^2\epsilon_x^2}{2\sigma^2})\epsilon_h\right)} \\ &= \mu \pm \sigma\sqrt{-2\log(\epsilon_h) + \epsilon_x^2}. \end{aligned} \tag{2.22}$$

L-distribution LR gradient estimator: It is straightforward to obtain the LR gradient estimator for the L-distribution by applying the appropriate importance weight to $\frac{\mathrm{d}p}{\mathrm{d}\mu}$ based on the probability density. Note that as previously shown, the pdf of the L-distribution is $q_L(x) = |x-\mu|\left|\frac{\mathrm{d}p(x;\mu,\sigma)}{\mathrm{d}x}\right|$, and that $\frac{\mathrm{d}p(x)}{\mathrm{d}\mu} = -\frac{\mathrm{d}p(x)}{\mathrm{d}x}$, then we get

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\mu}\mathbb{E}_{x\sim p(x;\theta)}\left[\phi(x)\right] = \mathbb{E}_{x\sim q_L(x)}\left[\frac{\frac{\mathrm{d}p}{\mathrm{d}\mu}}{q_L(x)}\phi(x)\right] &= \mathbb{E}_{x\sim q_L(x)}\left[\frac{1}{x-\mu}\phi(x)\right] \\ &= \mathbb{E}_{x\sim q_L(x)}\left[\frac{\operatorname{sgn}(x-\mu)}{\sigma\sqrt{-2\log(\epsilon_h)+\epsilon_x^2}}\phi(x)\right]. \end{aligned} \tag{2.23}$$

It is interesting to note, that this gradient estimator, has the form $\frac{\phi(x)}{x-\mu}$, so if antithetic sampling is applied, it becomes $\frac{\phi(x_+)-\phi(x_-)}{\Delta x}$, which has the same form as the finite difference gradient estimator. Therefore, if $\phi(x)$ is linear, then this new estimator will always compute the exact derivative with a single pair of samples; however, the



(a) Gaussian as reference **(b)** L-distribution

(c) B-distribution **(d)** W-distribution

Figure 2.3: New importance sampling distributions to reduce likelihood ratio gradient variance, described in Sections 2.2.1 and 2.2.2. The shaded regions are histograms generated with the direct sampling methods, and the solid lines are the analytic probability densities. Both methods match, demonstrating the correctness of the derivations.

sampling distribution q_L is adjusted such that the gradient estimator is unbiased for arbitrary $\phi(x)$.

2.2.2 Slice ratio importance sampling

The derivation in Section 2.2.1 appeared quite ad hoc, and it is unclear whether it is a good distribution to sample from in general. In this section I derive an optimal sampling distribution to minimize the variance in the situation when we assume no knowledge about $\phi(x)$. The concept of sampling the height introduced in the previous section will prove useful in this derivation, and in fact the L-distribution turns out to be an optimal distribution for a 2-dimensional Gaussian base distribution (the L-distribution is also optimal in 1D if $\phi(x)$ is linear).

Optimal importance sampling for minimum variance: We seek a distribution $q(x)$, s.t. the variance of $\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}\phi(x)/q(x)$ is minimized. As $\phi(x)$ is not known *a priori*,[5] we minimize the variance of $\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}/q(x)$. The derivation is analogous to the standard result for optimal importance sampling in statistics (Owen, 2013). The variance can be expressed as $\int q(x)\left(\frac{\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}}{q(x)}\right)^2 \mathrm{d}x$. Adding in the constraint $\int q(x)\ \mathrm{d}x = 1$ with a Lagrange multiplier λ, and performing a variational optimization by setting the derivative w.r.t. q to 0 we have:

$$\begin{aligned} &\frac{\mathrm{d}}{\mathrm{d}q}\left(\int q(x)\left(\frac{\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}}{q(x)}\right)^2 \mathrm{d}x + \lambda\left(\int q(x)\ \mathrm{d}x - 1\right)\right) = 0 \\ &-\left(\frac{\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}}{q(x)}\right)^2 + \lambda = 0 \quad \Rightarrow \quad q(x) = \left|\frac{\mathrm{d}p\left(x;\theta\right)}{\mathrm{d}\theta}\right| / \sqrt{\lambda}\ . \end{aligned} \tag{2.24}$$

From Equation (2.24) we see that the optimal importance sampling distribution is proportional to the magnitude of the gradient of the base distribution. The questions are then how to normalize this distribution, and how to sample from it? In general it is difficult, but I derive methods for some special cases. In particular, for $\theta = \mu$ is a location parameter, I find a general method that works for any distribution $p\left(x;\theta\right)$ if it is parameterized by a mean shifting parameter (i.e. the location family).

For a Gaussian, one can derive two possible distributions: one for μ and one for σ. The derivative w.r.t. μ appears more important, so I derive it first. The derivation is easily extended to location family distributions. Note that $\frac{\mathrm{d}p(x)}{\mathrm{d}\mu} = -\frac{\mathrm{d}p(x)}{\mathrm{d}x}$ (this holds for all location family distributions), and recall that for the L-distribution in Section 2.2.1, a transformation from the h-coordinate to the x-coordinate caused a $\left|\frac{\mathrm{d}p(x)}{\mathrm{d}x}\right|$ term. These insights allow us to derive the distribution and a sampling method. Namely, to sample from the distribution: 1) sample $h \sim \mathrm{unif}(0, p_{max})$, where $p_{max} = p\left(\mu;\mu,\sigma\right)$ is the peak probability density, 2) sample x from one edge of the slice $p^{-1}(h) = \{x : p\left(x;\mu,\sigma\right) = h\}$. This process is illustrated in Figure 2.4 Putting these results together, one obtains the probability density function, as well as a sampling method:

$$\begin{aligned} p_B\left(x;\mu,\sigma\right) &= \frac{|x-\mu|}{2\sigma^2}\exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \\ x &= \mu \pm \sigma\sqrt{-2\log(\epsilon_h)} \quad \text{where} \quad \epsilon_h \sim \mathrm{unif}(0,1) \end{aligned} \tag{2.25}$$

I call the derived distribution the B-distribution, because the shape resembles a sideways B, and it is plotted in Figure 2.3c. Similarly to how the L-distribution was related to the Maxwell-Boltzmann distribution, the B-distribution is just the Rayleigh distribution symmetrized about the origin. The full derivation is below.

[5] Note that other methods, such as adaptive importance sampling could be used to also take into account for $\phi(x)$ when sampling multiple points, but here I focus on the non-adaptive case. Of course adaptive methods could be combined with my methods to achieve even better performance.

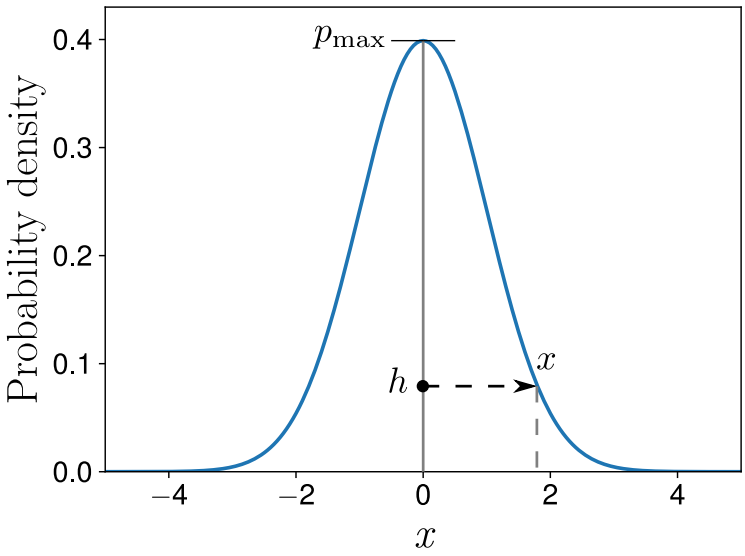


Figure 2.4: Illustration of slice ratio sampling method.

Slice ratio gradient derivation for a Gaussian base distribution: The pdf is

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{2.26}$$

The maximum probability density is at $x = \mu$:

$$p_{\max} = \frac{1}{\sqrt{2\pi}\sigma}. \tag{2.27}$$

The probability density for the slice ratio distribution can be derived by performing a change in coordinates from the h value to x. The probability mass at a slice $\mathrm{d}h$ split between two sides is ${}^{\mathrm{d}h}/_{2p_{\max}}$, so

$$\frac{1}{2p_{\max}}\mathrm{d}h = \frac{1}{2p_{\max}}\left|\frac{\mathrm{d}p}{\mathrm{d}x}\right|\mathrm{d}x. \tag{2.28}$$

From this we get

$$\begin{aligned} q(x) &= \frac{\sqrt{2\pi}\sigma}{2}\left|\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\frac{-(x-\mu)}{\sigma^2}\right| \\ &= \frac{|x-\mu|}{2\sigma^2}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \end{aligned} \tag{2.29}$$

which is the pdf in Equation (2.25).

To derive the sampling method, first, the inverse of the probability density $p^{-1}(h)$ was previously provided in Equation (2.21) as

$$x = \mu \pm \sigma\sqrt{-\log(2\pi) - 2\log(\sigma) - 2\log(h)}\,. \tag{2.30}$$

Now, noting $h = p_{\max}\epsilon_h$, where $\epsilon_h \sim \text{unif}(0,1)$, we end up with the sampling method:

$$\begin{aligned} x &= \mu \pm \sigma\sqrt{-\log(2\pi) - 2\log(\sigma) - 2\log\left(\frac{1}{\sqrt{2\pi}\sigma}\epsilon_h\right)} \\ &= \mu \pm \sigma\sqrt{-2\log\left(\epsilon_h\right)}\,, \text{where } \epsilon_h \sim \text{unif}(0,1). \end{aligned} \tag{2.31}$$

Optimal importance sampling distribution for $\frac{\mathrm{d}}{\mathrm{d}\sigma}$: For completeness, I also derive the optimal sampling distribution for the derivative w.r.t. σ. First note that $\frac{\mathrm{d}p(x)}{\mathrm{d}\sigma} = \sigma\frac{\mathrm{d}^2p(x)}{\mathrm{d}x^2}$ (this condition is specific to Gaussian distributions). This expression means that if we apply the same height sampling concept as used for μ on the distribution proportional to $\left|\frac{\mathrm{d}p(x)}{\mathrm{d}x}\right|$, we would obtain samples with probability density proportional to $\left|\frac{\mathrm{d}^2p(x)}{\mathrm{d}x^2}\right|$, and would hence be sampling from the desired distribution. The required base distribution is just the B-distribution (Eq. 2.25), so I can perform the required derivation. The result is given below:

$$\begin{aligned} p_W\left(x;\mu,\sigma\right) &= \frac{\sqrt{e}}{4\sigma}\left|\exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)\left(\frac{|x-\mu|^2}{\sigma^2} - 1\right)\right| \\ x &= \mu \pm \sigma\sqrt{W(-\epsilon_h^2/e)} \quad \text{where} \quad \epsilon_h \sim \text{unif}(0,1) \end{aligned} \tag{2.32}$$

In the above equation, $W(x)$ is the Lambert W function (Corless et al., 1996)—a function s.t. $z = W(ze^z)$. The solution for W is picked with equal probability from the -1 and 0 branches of W, and the $\pm$ is also sampled randomly with equal probability. Efficient implementations of W are available in common numerical computation packages, such as scipy (Jones et al., 01) or MATLAB. I call the result the W-distribution, because the sampling method includes the W function, and the pdf is plotted in Figure 2.3d. To the best of my knowledge, this distribution does not exist in the literature.

Derivation of W-distribution: I first derive the probability density $p_W\left(x\right)$, then the sampling scheme. The base distribution is $p_B\left(x\right)$, and I apply a transformation by which I sample the height h, and transform this to a point x by using the inverse $x \sim p_B^{-1}(h)$, and sampling uniformly between the x values that satisfy the equation, e.g. for the B-distribution in Figure 2.3c there are usually 4 points for each h value. Therefore $\mathrm{d}h = \frac{1}{4\max(p_B)}\left|\frac{\mathrm{d}p_B(x)}{\mathrm{d}x}\right|\mathrm{d}x$ and $p_W\left(x\right) = \frac{1}{4\max(p_B)}\left|\frac{\mathrm{d}p_B(x)}{\mathrm{d}x}\right|$. The required derivative is given by

$$\left|\frac{\mathrm{d}p_B\left(x\right)}{\mathrm{d}x}\right| = \frac{\text{sgn}(x-\mu)}{2\sigma^2}\exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) - \frac{|x-\mu|}{2\sigma^2}\exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)\frac{(x-\mu)}{\sigma^2} \tag{2.33}$$

Setting the derivative to 0 gives the locations of the peaks at $x = \mu \pm \sigma$. Evaluating $p_B\left(x\right)$ at these locations in Equation (2.25) gives the peak value as

$$\max(p_B) = \frac{1}{2\sigma}\exp\left(-1/2\right) \tag{2.34}$$

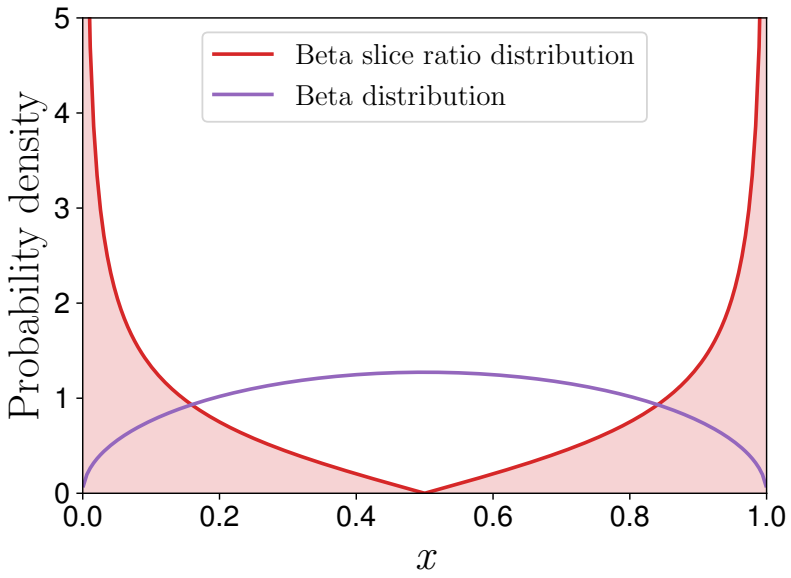


Figure 2.5: Slice ratio distribution for the Beta distribution with $\alpha = 1.5$.

Combining these results gives the density in Equation (2.32). Deriving the sampling method, requires inverting $p_B(x)$:

$$\begin{aligned} h &= \frac{|x-\mu|}{2\sigma^2}\exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right), \text{let} \quad t = \frac{(x-\mu)^2}{2\sigma^2}, \text{then} \\ h &= \frac{t^{1/2}}{\sigma\sqrt{2}}\exp(-t) \\ h^2 &= \frac{t}{2\sigma^2}\exp(-2t) \\ -4\sigma^2h^2 &= -2t\exp(-2t) \\ W\left(-4\sigma^2h^2\right) &= -2t \end{aligned} \tag{2.35}$$

Now recalling that $h = p_{max}\epsilon_h, \quad \epsilon_h \sim \text{unif}(0,1)$, where $p_{max} = \frac{1}{2\sigma}\exp(-1/2)$ from Equation (2.34), and plugging in the value of t, gives the sampling method in Equation (2.32).

Slice ratio sampling for the symmetric Beta distribution: The slice ratio sampling method is crucial in some situations. For example, consider a distribution such as the symmetric Beta distribution (Fig. 2.5):

$$p_\beta(x) = \frac{x^{\alpha-1}(1-x)^{\alpha-1}}{B(\alpha,\alpha)}. \tag{2.36}$$

When α tends to 1 from above, this distribution tends to the uniform distribution between 0 and 1. Consider a distribution with the same shape, but where the mean is shifted, s.t. it is symmetric about a parameter μ, instead of $x = 1/2$. In this case,

as α tends to 1, the variance of the gradient w.r.t. μ will tend to ∞, because $\frac{\mathrm{d}p_\beta(x)}{\mathrm{d}\mu}$ is around 0 in most of the sampling range, but very large at the edges of the distribution. I derived the optimal pdf, sampling method and gradient estimator:

$$\begin{aligned} p_{\beta R}(x) &= \frac{\alpha-1}{2\times 0.25^{\alpha-1}}(x-x^2)^{\alpha-2}|1-2x|, \\ x &= 0.5 \pm 0.5\sqrt{1-\epsilon_h^{1/(\alpha-1)}} \quad \text{where} \quad \epsilon_h \sim \text{unif}(0,1), \\ \frac{\mathrm{d}}{\mathrm{d}\mu}\mathbb{E}_{x\sim p_\beta(x)}\left[\phi(x)\right] &= \mathbb{E}_{x\sim q(x)}\left[\text{sgn}(x-0.5)\frac{2\times 0.25^{\alpha-1}}{B(\alpha,\alpha)}\phi(x)\right], \text{for } \alpha > 1. \end{aligned} \tag{2.37}$$

For a shifted, stretched and centered distribution, replace x with $k(x-0.5)+\mu$, and the gradient estimator needs to be scaled down by k. For the base distribution to have a variance σ^2, set k to $2\sigma\sqrt{2\alpha+1}$. Derivation: The variance of the Beta distribution is given by $\frac{\alpha^2}{(2\alpha)^2(2\alpha+1)} = \frac{1}{4(2\alpha+1)}$. We need $k^2\frac{1}{4(2\alpha+1)} = \sigma^2 \Rightarrow k = 2\sigma\sqrt{2\alpha+1}$. The full derivations for the gradient estimator are below.

Slice ratio gradient for a symmetric Beta distribution derivation: The pdf is

$$p_\beta(x) = \frac{x^{\alpha-1}(1-x)^{\alpha-1}}{B(\alpha,\alpha)} = \frac{(x-x^2)^{\alpha-1}}{B(\alpha,\alpha)}. \tag{2.38}$$

The maximum probability density is at $x = 0.5$:

$$p_{\max} = \frac{0.25^{\alpha-1}}{B(\alpha,\alpha)} \tag{2.39}$$

Similarly to Equation (2.29), the pdf of the slice ratio distribution is $q(x) = {}^{|\frac{\mathrm{d}p}{\mathrm{d}x}|}/_{2p_{\max}}$:

$$\begin{aligned} q(x) &= \frac{B(\alpha,\alpha)}{2\times 0.25^{\alpha-1}}\left|\frac{(x-x^2)^{\alpha-2}}{B(\alpha,\alpha)}(\alpha-1)(1-2x)\right| \\ &= \frac{\alpha-1}{2\times 0.25^{\alpha-1}}\left|(x-x^2)^{\alpha-2}(1-2x)\right|, \end{aligned} \tag{2.40}$$

which is the pdf in Equation (2.37).

To derive the sampling method, first derive the inverse of the probability density $p^{-1}(h)$ as

$$\begin{aligned} h &= \frac{(x-x^2)^{\alpha-1}}{B(\alpha,\alpha)} \\ h^{1/(\alpha-1)} &= \frac{(x-x^2)}{B(\alpha,\alpha)^{1/(\alpha-1)}} \\ x^2 - x + (hB(\alpha,\alpha))^{1/(\alpha-1)} &= 0 \\ x &= \frac{1}{2} \pm \frac{1}{2}\sqrt{1-4(hB(\alpha,\alpha))^{1/(\alpha-1)}}\,. \end{aligned} \tag{2.41}$$

Now, noting $h = p_{\max}\epsilon_h$, where $\epsilon_h \sim \text{unif}(0,1)$, we end up with the sampling method:

$$\begin{aligned} x &= \frac{1}{2} \pm \frac{1}{2}\sqrt{1 - 4\left(\epsilon_h \frac{0.25^{\alpha-1}}{B(\alpha,\alpha)} B(\alpha,\alpha)\right)^{1/(\alpha-1)}} \\ &= \frac{1}{2} \pm \frac{1}{2}\sqrt{1 - \epsilon_h^{1/(\alpha-1)}} \text{ , where } \epsilon_h \sim \text{unif}(0,1). \end{aligned} \tag{2.42}$$

Multidimensional Gaussian Slice ratio gradient In multiple dimensions the optimality equation in Equation (2.24) is still valid, but the method to derive the normalized distribution and sampling method have to be modified. For simplicity, I consider the case of optimal sampling for the derivative w.r.t. μ for a spherical Gaussian. Motivated from the derivation for a single dimension, consider a method which would sample a unit vector on a sphere for a direction $\hat{\mathbf{r}}$, as well as a height h, then invert the distribution s.t. $\mathbf{x} = p^{-1}(h, \hat{\mathbf{r}})$, where p^{-1} is a function s.t. $p(\mathbf{x}) = h$ and $\mathbf{x} = r\hat{\mathbf{r}}$, i.e., it picks $\mathbf{x}$ in the direction $\hat{\mathbf{r}}$, which gives the desired probability density. The conversion from the h-coordinate to the $\mathbf{x}$-coordinate would still give the desired $\left|\frac{\mathrm{d}p(\mathbf{x})}{\mathrm{d}\mathbf{x}}\right|$ term; however, due to the change in the surface area as the radius r is increased, there is an additional factor $r^{-(D-1)}$, where D is the dimensionality. In other words, the sampling method has to be modified to cancel out this new factor, and the required distribution must have the property: $q(\mathbf{x}) \propto r^{D-1}\left|\frac{\mathrm{d}p(\mathbf{x})}{\mathrm{d}\mathbf{x}}\right|$. For a Gaussian base distribution we get $q(\mathbf{x}) \propto r^D \exp(-\frac{r^2}{2\sigma^2})$. The required distribution is the chi distribution:

$$q(z;k) = \frac{1}{2^{(k/2-1)}\Gamma(k/2)} z^{k-1} \exp\left(-\frac{z^2}{2}\right) \quad \text{where} \quad r = \sigma z \quad \text{and} \quad k = D+1. \tag{2.43}$$

In fact, the Rayleigh distribution (corresponding to the B-distribution) is a special case of this distribution for $D = 1$, and the Maxwell-Boltzmann distribution (corresponding to the L-distribution) is the case for $D = 2$. Note that if one performs this sampling procedure, but while using $\tilde{D} = D - 1$, then the sample comes exactly from the original Gaussian distribution $p(\mathbf{x}; \mu, \Sigma)$. This remark highlights that there are diminishing returns to changing the sampling distribution as the dimensionality of the space is increased, because the optimal sampling distribution tends to the original Gaussian distribution. For this reason, I propose to sample each dimension separately from the B-distribution, potentially allowing for a bias, but while reducing the variance of the gradient estimator.

In general, I believe that such a technique will be necessary for other distributions as well if the dimension grows high. To see this, consider the importance weighted likelihood ratio gradient estimator for a factorized distribution $p(\mathbf{x}; \theta) = \prod_i p_i(x_i; \theta_i)$:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\theta_i}\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\phi(\mathbf{x})\right] &= \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[\frac{p\left(\mathbf{x}\right)}{q(\mathbf{x})}\frac{\mathrm{d}\log p}{\mathrm{d}\theta_i}\phi(\mathbf{x})\right] \\ &= \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[\frac{p_{\setminus i}\left(\mathbf{x}_{\setminus i}\right)p_i\left(x_i\right)}{q_{\setminus i}(\mathbf{x}_{\setminus i})q_i(x_i)}\frac{\mathrm{d}\log p_i\left(x_i\right)}{\mathrm{d}\theta_i}\phi(\mathbf{x})\right], \qquad (2.44) \\ &\qquad \text{where } p_{\setminus i} \text{ is } \prod_{j\neq i} p_j\left(x_j;\theta_j\right).\end{aligned}$$

While q can be modified to reduce the variance of $\frac{p_i(x_i)}{q_i(x_i)}\frac{\mathrm{d}\log p_i(x_i)}{\mathrm{d}\theta}$, this will increase the variance of the $\frac{p_{\setminus j}\left(\mathbf{x}_{\setminus j}\right)}{q_{\setminus j}(\mathbf{x}_{\setminus j})}$ terms for $j \neq i$. If each q_j is modified, then the variance of these terms grows exponentially with the dimension, and any decrease in variance from having modified q_i becomes negligible. My proposed solution is to replace $\frac{p_{\setminus j}\left(\mathbf{x}_{\setminus j}\right)}{q_{\setminus j}(\mathbf{x}_{\setminus j})}$ with its expected value, which is 1. In practice, such a scheme may introduce a small bias, but drastically reduce the variance. Next I show some fairly general conditions under which this method still gives an unbiased gradient estimator.

A few sufficient conditions for an unbiased gradient estimator, when ignoring importance weights from other dimensions: First, I consider functions of the form $\phi(\mathbf{x}) = \sum_{i=1}^{D}\phi_i(x_i)$, and show that ignoring the importance weights from dimension $j \neq i$ for the derivative w.r.t. θ_i, still gives an unbiased gradient estimator. Note that $\mathbb{E}_{x_i\sim q(x_i)}\left[\frac{p(x_i)}{q(x_i)}\frac{\mathrm{d}\log p(x_i;\theta_i)}{\mathrm{d}\theta_i}\mathbb{E}_{x_j\sim q(x_j)}\left[\phi_j(x_j)\right]\right] = \mathbb{E}_{x_i\sim p(x_i)}\left[\frac{\mathrm{d}\log p(x_i;\theta_i)}{\mathrm{d}\theta_i}\mathbb{E}_{x_j\sim q(x_j)}\left[\phi_j(x_j)\right]\right] = 0$, because $\mathbb{E}_{x_i\sim p(x_i)}\left[\frac{\mathrm{d}\log p(x_i;\theta_i)}{\mathrm{d}\theta_i}Y\right] = \frac{\mathrm{d}}{\mathrm{d}\theta_i}\mathbb{E}\left[Y\right] = 0$, for Y statistically independent from x_i, because the expected value of Y would not change if the distribution of x_i is changed. This result means that if the function ϕ has a structure, such that different dimensions affect the function value independently, then the gradient estimator will still be unbiased.

Next I show that even if the dimensions are not independent, in some cases the gradient estimator is unbiased. Notably, for a quadratic function $\phi(\mathbf{x}) = \mathbf{a}^T\mathbf{x} + \mathbf{x}^TQ\mathbf{x} + c$, the gradient estimator will be unbiased. First note that the diagonal terms in the quadratic function are independent, so the gradient of that portion of the cost will be unbiased based on the previous example. Next consider the off-diagonal terms of $\mathbf{x}^TQ\mathbf{x}$, which are $x_iQ_{ij}x_j$. Note that the distributions I considered, namely the B, W and L distributions were all symmetric about the mean value μ_j. Therefore $\mathbb{E}_{x_j\sim q(x_j)}\left[Q_{ij}x_j\right] = \mathbb{E}_{x_j\sim p(x_j)}\left[Q_{ij}x_j\right]$, and the derivative $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{x_i\sim p(x_i)}\left[x_i\mathbb{E}_{x_j\sim q(x_j)}\left[\frac{p(x_j)}{q(x_j)}Q_{ij}x_j\right]\right]$ remains unchanged even if one ignores the $p(x_j)/q(x_j)$ importance weights, because $\mathbb{E}_{x_j\sim q(x_j)}\left[\frac{p(x_j)}{q(x_j)}Q_{ij}x_j\right] = \mathbb{E}_{x_j\sim p(x_j)}\left[Q_{ij}x_j\right] = \mathbb{E}_{x_j\sim q(x_j)}\left[Q_{ij}x_j\right]$. This result implies that if the variance of the distribution σ^2 is small, and ϕ is smooth, such that ϕ is roughly quadratic in the range of the sampling distribution, then the gradient estimator will remain approximately unbiased.

Effect of greater variance of q_i: Finally, I point toward another issue with modifying q_i in Equation (2.44). The variance of q_i may be larger than the variance of

p_i, and this could manifest as a larger variance of $\phi(\mathbf{x})$, which would act as additional noise on the other dimensions $j \neq i$. My proposed solution is to optimize the reduction in gradient variance while constraining the variance of q. Assuming the mean $\mu = 0$, this can be performed using a variational optimization with an additional Lagrange multiplier for $\int q(x)x^2 \mathrm{d}x = k\sigma^2$ analogously to Equation (2.24). The general equation is derived below:

$$\frac{\mathrm{d}}{\mathrm{d}q}\left(\int q(x)\left(\frac{\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}}{q(x)}\right)^2 \mathrm{d}x + \lambda_1\left(\int q(x)\,\mathrm{d}x - 1\right) + \lambda_2\left(\int q(x)x^2\,\mathrm{d}x - k\sigma^2\right)\right) = 0$$
$$q(x) = \left|\frac{\mathrm{d}p\,(x;\theta)}{\mathrm{d}\theta}\right| \Big/ \sqrt{\lambda_1 + \lambda_2 x^2}. \tag{2.45}$$

For a Gaussian $p\,(x;\theta)$, this equation can be solved (derivation in next section). I call the result the truncated ratio gradient (TRRG). The pdf, sampling method and gradient estimator are below:

$$p_{tr}\,(x;c,\mu,\sigma) = \frac{\exp(-\frac{c^2}{2})}{1-\Phi(c)}\frac{1}{\sigma 2\sqrt{2\pi}}\frac{|x-\mu|}{\sqrt{(x-\mu)^2+\sigma^2c^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$
where $\Phi(c)$ is the cdf of a unit normal distribution,
$$x = \mu \pm \sigma\sqrt{\epsilon_c^2 - c^2} \quad \text{where} \quad \epsilon_c \sim \mathrm{truncG}(c,\infty)$$
and $\mathrm{truncG}(a,b)$ is the unit normal truncated[6]between a and b,
$$\frac{\mathrm{d}}{\mathrm{d}\mu}\mathbb{E}_{x\sim p_{tr}(x)}\left[\phi(x)\right] = \mathbb{E}_{x\sim q(x)}\left[\mathrm{sgn}(x-\mu)\frac{2\epsilon_c}{\sigma}\frac{1-\Phi(c)}{\exp(-\frac{c^2}{2})}\phi(x)\right]. \tag{2.46}$$

This distribution interpolates between a Gaussian distribution and the B-distribution. The interpolation is controlled by the c parameter: for $c = 0$ the distribution is Gaussian, and for $c \to \infty$ the distribution tends to the B-distribution. One half of the distribution is plotted in Figure 2.6 for several values of c.

Derivation of truncated ratio distribution and gradient: The pdf $p_{tr}\,(x;c,\mu,\sigma)$ in Equation (2.46) satisfies the optimality Equation (2.45), therefore, as long as it is a proper probability density, it is correct. I will show that the proposed sampling method corresponds to this pdf.

Without loss of generality, let $\mu = 0$. The pdf of ϵ_c is given by

$$p\,(\epsilon_c) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{\epsilon_c^2}{2}\right)\frac{1}{1-\Phi(c)}, \text{between } \epsilon_c \in [c,\infty]. \tag{2.47}$$

[6]By truncated I mean that the probability density is set to 0 outside these bounds, and the remaining probability distribution has to be renormalized. Such a distribution is implemented, e.g., in MATLAB and scipy (Jones et al., 01).

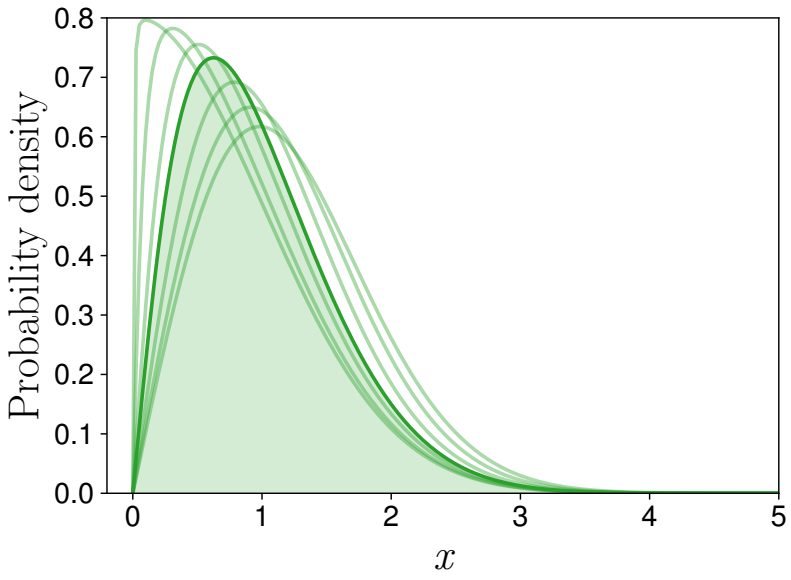


Figure 2.6: Truncated ratio distribution for $c \in [0.01, 0.1, 0.3, \mathbf{0.5}, 1.0, 2.0, 5.0]$

Perform a change of coordinates from ϵ_c to x and account for the stretching due to the Jacobian:

$$x = \sigma\sqrt{\epsilon_c^2 - c^2} \Rightarrow \mathrm{d}x = \sigma\frac{\epsilon_c}{\sqrt{\epsilon_c^2 - c^2}}\mathrm{d}\epsilon_c, \tag{2.48}$$

note that $\sigma\epsilon_c = \sqrt{x^2 + \sigma^2 c^2}$, so

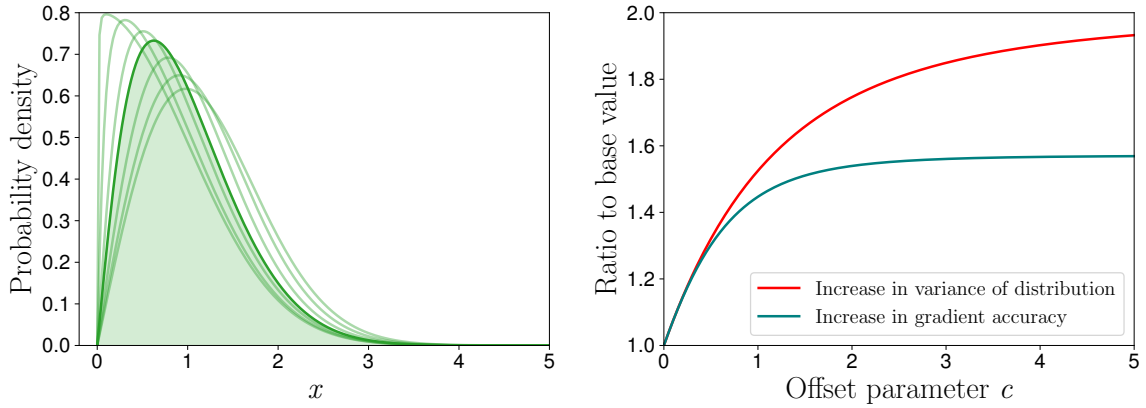
$$\frac{x/\sigma}{\sqrt{x^2 + \sigma^2 c^2}}\mathrm{d}x = \mathrm{d}\epsilon_c, \tag{2.49}$$

therefore

$$\begin{aligned} p\left(\epsilon_c\right)\mathrm{d}\epsilon_c &= \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{\epsilon_c^2}{2}\right)\frac{1}{1-\Phi(c)}\frac{x/\sigma}{\sqrt{x^2+\sigma^2c^2}}\mathrm{d}x \\ &= \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{x^2}{2\sigma^2}+\frac{c^2}{2}\right)\frac{1}{1-\Phi(c)}\frac{x/\sigma}{\sqrt{x^2+\sigma^2c^2}}\mathrm{d}x \\ &= \frac{\exp\left(-\frac{c^2}{2}\right)}{1-\Phi(c)}\frac{1}{\sigma\sqrt{2\pi}}\frac{x}{\sqrt{x^2+\sigma^2c^2}}\exp\left(-\frac{x^2}{2\sigma^2}\right)\mathrm{d}x. \end{aligned} \tag{2.50}$$

This result is the desired probability distribution on the half-plane $[0, \infty]$. It is normalized by construction. Symmetrizing the distribution about 0, and shifting by a mean parameter μ gives the desired result. The gradient estimator is easily derived by $\frac{\mathrm{d}p}{\mathrm{d}\theta}/q$.

Choosing the c parameter based on the expected accuracy: Figure 2.7b shows how the accuracy of $\frac{\mathrm{d}p}{\mathrm{d}\mu}/q$, and the variance of the distribution scale with c (I name these functions $t(c)$ and $v(c)$ respectively). These functions were computed analytically



(a) $c \in [0.01, 0.1, 0.3, \mathbf{0.5}, 1.0, 2.0, 5.0]$ **(b)** $t(c)$ in teal and $v(c)$ in red

Figure 2.7: Scaling of truncated ratio gradient accuracy with the dimension (2.7b) and the truncated ratio distribution for various c (2.7a).

(derivation in next section). How should one pick the parameter c? A simple choice may be to pick c around 0.5, where the accuracy starts increasing slower than the variance of the distribution. But is there a more principled method based on the dimensionality?

I give a short analysis of the effect of the variance and guidelines for picking c. For example, consider the case when $\phi(x)$ is linear with slope $\frac{\mathrm{d}\phi}{\mathrm{d}x} = a$ in every dimension, the dimensionality is D and the variance is scaled by v_c, then the variance of $\phi(x)$ would increase by a factor v_c to $a^2\sigma^2 D v_c$. The noise from the other dimensions would scale as roughly $v_c(D-1)a^2\sigma^2$. However, the increase in accuracy t_c counteracts this increase in noise, and the gradient variance of this noise scales as $({}^{v_c}/_{t_c})(D-1)a^2\sigma^2$. Now if we assume that the gradient signal has a variance around $a^2\sigma^2$, and we want to guarantee that the additional gradient noise from the other dimensions does not exceed the maximum decrease in variance, then we could pick c s.t. $({}^{v_c}/_{t_c} - 1)(D-1) \approx 1$. In Table 2.1, I show several values of $1/({}^{v_c}/_{t_c} - 1)$ and the expected increase in accuracy t_c, which can be used as a guideline for picking an appropriate c for the dimensionality of your problem. One could also estimate the reduction in gradient signal variance as $(1 - {}^1/_{t_c})a^2\sigma^2$ for a more conservative estimate of D, but in practice, the reduction in gradient signal variance is greater than ${}^1/_{t_c}$ because of structure in $\phi(x)$. Specifically, $\phi(x)$ tends to increase further away from the mean value of x. This increase means that more of the probability mass should be moved away from the center, and consequently, the increase in accuracy tends to be larger than when one does not assume anything about $\phi(x)$. In general, for deterministic problems it may be be better to be conservative and aim for a smaller increase in accuracy with a smaller c, whereas if $\phi(x)$ is stochastic, then the additional variance from other dimensions may be negligible and higher c values can be used.

Analytic variance and gradient accuracy derivations: The variance is most easily derived by working with the distribution on ϵ_c in Equation (2.47). Note that if we symmetrize the distribution about 0, then the mean will be 0, and the variance can

Table 2.1: Guidelines for choosing the offset parameter c for the truncated ratio gradient.

Suggested parameter c	0.1	0.2	0.3	0.4	0.5	0.6	0.8	1.0
Dimension $(D-1)$	4523	676	238	119	71	48	27	19
Exp. increase in accuracy t	1.076	1.144	1.204	1.257	1.302	1.341	1.402	1.447

be estimated as the expectation of x^2 when sampling from half the distribution:

$$\mathbb{V}[x] = \int x^2 p(\epsilon_c)\,\mathrm{d}\epsilon_c = \int \sigma^2(\epsilon_c^2 - c^2)p(\epsilon_c)\,\mathrm{d}\epsilon_c = \int \sigma^2\epsilon_c^2 p(\epsilon_c)\,\mathrm{d}\epsilon_c - \sigma^2 c^2. \tag{2.51}$$

So, we just need to find $\mathbb{E}[\epsilon_c^2] = \mathbb{V}[\epsilon_c] + \mathbb{E}[\epsilon_c]^2$. Denote $\mathcal{N}(x) = \mathcal{N}(x; 0, 1)$ the unit variance Gaussian distribution, and $\Phi(x)$ the cdf of the unit variance Gaussian, then the mean and variance of the 0 mean truncated Gaussian between $[c, \infty]$ can be written as $\mathbb{E}[\epsilon_c] = \frac{\mathcal{N}(c)}{1-\Phi(c)}$ and $\mathbb{V}[\epsilon_c] = 1 + \frac{c\mathcal{N}(c)}{1-\Phi(c)} - \left(\frac{\mathcal{N}(c)}{1-\Phi(c)}\right)^2$. Combining these two results:

$$\mathbb{E}\left[\epsilon_c^2\right] = 1 + \frac{c\mathcal{N}(c)}{1-\Phi(c)}. \tag{2.52}$$

Hence, the variance is

$$v(c) = \mathbb{V}[x] = \sigma^2\left(1 + \frac{c\mathcal{N}(c;0,1)}{1-\Phi(c)} - c^2\right). \tag{2.53}$$

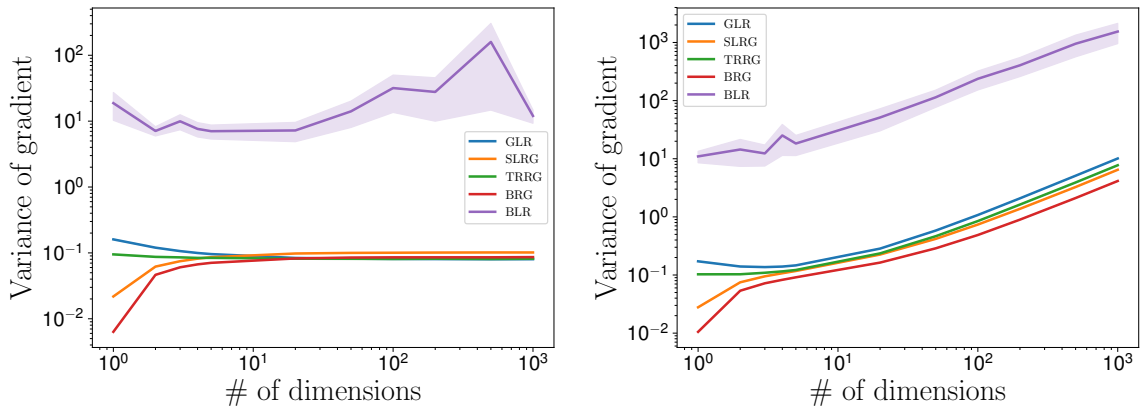
Next, I derive the variance of the gradient term $\frac{\mathrm{d}p}{\mathrm{d}\mu}/q$. Note that this term is given in Equation (2.46) as $\mathrm{sgn}(x-\mu)\frac{2\epsilon_c}{\sigma}\frac{1-\Phi(c)}{\exp(-\frac{c^2}{2})}$, so the variance is

$$\begin{aligned}\mathbb{V}\left[\frac{\mathrm{d}p}{\mathrm{d}\mu}/q\right] = \mathbb{E}\left[\left(\frac{2\epsilon_c}{\sigma}\frac{1-\Phi(c)}{\exp(-\frac{c^2}{2})}\right)^2\right] &= \mathbb{E}\left[\epsilon_c^2\right]\left(\frac{2}{\sigma}\frac{(1-\Phi(c))}{\exp(-\frac{c^2}{2})}\right)^2 \\ &= \left(1 + \frac{c\mathcal{N}(c;0,1)}{1-\Phi(c)}\right)\frac{4}{\sigma^2}\frac{(1-\Phi(c))^2}{\exp(-c^2)}.\end{aligned} \tag{2.54}$$

Finally, note that the gradient accuracy $t(c)$ is defined as $1/\mathbb{V}\left[\frac{\mathrm{d}p}{\mathrm{d}\mu}/q\right]$.

2.3 Experiments with slice ratio gradients

I conduct experiments both on a simple problem to verify the theoretical results, as well as on a larger evolution strategies problem to see the applicability of my methods. The main result is that for a Gaussian distribution at best a modest improvement is possible, so in practice it may not make much of a difference. However, for non-Gaussian distributions, the improvement can be arbitrarily large, e.g., for the Beta distribution with $\alpha = 1.5$, the gradient accuracy was improved by 100–1000 times.



(a) Deterministic **(b)** Noisy

Figure 2.8: Scaling of gradient estimator variance on a quadratic problem with deterministic and noisy function evaluations. The confidence intervals correspond to one standard deviation of the estimate.

Even though I believe that it would be naive to think that Gaussians are good enough for everything, non-Gaussian distributions have until now not found much use in reinforcement learning. I believe that my techniques will be necessary for using different distributions.

2.3.1 Experiments to verify theoretical results

I performed experiments on a quadratic $\phi(\mathbf{x})$ to verify the theory. In this experiment, I emphasize that the slice ratio gradient method is crucial for some non-Gaussian distributions, e.g., for my example with a Beta distribution.

Setup: $\phi(\mathbf{x})$ is a quadratic $(\mathbf{x}-\mathbf{a})^T Q(\mathbf{x}-\mathbf{a})$, where $\mathbf{a} = \mathbf{1}$ and $Q = \text{ones}(D, D)/D^2$ is a matrix of ones, which is scaled, such that $\phi(\mathbf{x})$ remains constant at $\mathbf{x} = \mathbf{0}$. I evaluate a deterministic case, as well as a case when Gaussian noise $\sigma_n^2 = 1$ is added on ϕ. I vary the dimension between 1–1000, and plot the variance of the gradient estimators: GLR—LR gradient with a Gaussian $p(x)$; SLRG—slice ratio gradient with a Gaussian $p(x)$; TRRG—truncated ratio gradient with $c = 0.5$; BRG—slice ratio gradient with a Beta $p(x)$, and $\alpha = 1.5$, plotted in Figure 2.5; BLR—LR gradient with a Beta $p(x)$, and $\alpha = 1.5$. The mean of the distributions was set to 0 and the variance was set to 1 (the Beta distributions were stretched by $k = 2\sigma\sqrt{2\alpha + 1}$ to achieve this). I used antithetic sampling, so that the effect of any baseline could be ignored. The gradient was estimated by averaging 100 samples, and this was repeated for a large number of times to estimate the variance of the gradient estimator. Bootstrapping was used to obtain confidence intervals. The results are plotted in Figure 2.8.

Results and analysis: The main result is that using the slice ratio method, the gradient accuracy for the Beta distribution could be increased by 100–1000 times (compare

BRG to BLR), showing that my method is necessary for some non-Gaussian distributions. In general, the increase in gradient accuracy would tend to ∞ as the α parameter tends to 1 from above; however, even for moderately curved cases, such as $\alpha = 1.5$ (Fig. 2.5) the improvement in accuracy can be drastic. This result arose from the fact that likelihood ratio gradients are not suitable for discontinuous $p(x;\theta)$, as they would give a biased gradient estimator. If $p(x;\theta)$ has sharp discontinuities, one could deal with this by sampling at the discontinuous points; however, it is not clear what to do when $p(x;\theta)$ is not completely discontinuous, but instead has sharp cliffs. Slice ratio gradients would automatically solve the problem in both scenarios.

The results confirm my theoretical analysis: in the deterministic case, the SLRG method outperforms the standard GLR method, but as the dimensionality is increased, this reverses; whereas in the high-noise case, SLRG always outperforms GLR. In the noisy case, the variances at $D = 1000$ are GLR: 10.10 ± 0.05, SLRG: 6.46 ± 0.03, TRRG: 7.73 ± 0.04, BRG: 4.14 ± 0.02 (the errorbars correspond to 1 standard deviation). The ratios $10.10/6.46 = 1.563$ and $10.10/7.73 = 1.307$ match the theoretical improvements in gradient accuracy for the SLRG gradient at large c in Figure 2.7b and for the TRRG gradient at $c = 0.5$ in Table 2.1. In the deterministic case, the gradient variances at $D = 1000$ are GLR: 0.0803 ± 0.0004, SLRG: 0.1015 ± 0.0005, TRRG: 0.0815 ± 0.0004, BRG: 0.0864 ± 0.0004, showing that TRRG is more robust than SLRG to problems arising from increasing the dimension, while it still allows reducing the variance in the stochastic ϕ setting. Interestingly, BRG achieved a lower gradient variance than SLRG in the deterministic setting, and was overall the best in the stochastic setting even though the variances of the base distributions were the same.

2.3.2 Experiments in evolution strategies

Evolution strategies are a technique based on sampling in the parameter space of a problem $p(\mathbf{w};\theta)$, and applying LR gradients to optimize the objective $\mathbb{E}_{p(\mathbf{w};\theta)}[\phi(\mathbf{w})]$. For example $\mathbf{w}$ may be the parameters of a neural network policy in reinforcement learning, and the objective is to find the distribution $p(\mathbf{w};\theta)$ over the parameters $\mathbf{w}$ that gives the behavior with the largest expected reward. In this case, $\phi(\mathbf{w})$ would be the return function for a particular parameter set $\mathbf{w}$. One would first sample parameters $\mathbf{w}$, these would be kept fixed for one episode of the agent's behavior, the behavior would be evaluated based on a reward function, and the sum of the reward would be returned to the algorithm as $\phi(\mathbf{w})$. LR gradients can be used to evaluate $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{\mathbf{w}\sim p(\mathbf{w};\theta)}[\phi(\mathbf{w})]$, and the objective can be optimized directly using gradient ascent. I implemented my new importance sampling schemes into David Ha's Evolution Strategies code available from https://github.com/hardmaru/estool (Ha, 2017) (note that my methods are not available from the link yet), and tested my methods on cartpole swing-up and biped walker tasks illustrated in Figure 2.9.

General setup: In all experiments I used spherical Gaussian base distributions $p(\mathbf{w};\theta)$ for the GLR, SLRG and TRRG methods, while the sampling distributions $q(\mathbf{w})$ varied based on the importance sampling scheme. For BRG, I used a Beta base distribution, and applied the Beta slice ratio gradient method. I used antithetic sampling, i.e. I always sampled $\mathbf{w}$ in pairs, which are located opposite of each other in

the distribution. If such a scheme is used, then any constant baseline b (Greensmith et al., 2004), which is subtracted from the $\phi(\mathbf{w})$ values will cancel out from the opposite pairs, and the effect of such baselines can be ignored. I did not use weight decay. For TRRG, $c = 0.5$, and for BRG, $\alpha = 1.1$ in all cases. I used the RAIDEN CPU cluster at RIKEN-AIP for the experiments. Biped tasks were run on 33 cores, and cartpole tasks were run on 5 cores. All tasks were run for 2000 policy improvement iterations (gradient steps), and repeated for several different random number seeds (details in tables). Because the samples from $q(x)$ do not correspond to the objective $\mathbb{E}_{p(x;\theta)}[\phi(x)]$, I separately evaluated the performance by sampling from $p(x;\theta)$ after every 10 iterations. Note that this was done only for evaluation purposes, and did not have any effect on the learning.

Cartpole setup: State dimension: 5; Action dimension: 1; Policy: neural network with one hidden layer with 10 neurons and tanh activations, total number of parameters $\mathbf{w}$: 71; Optimizer: basic stochastic gradient ascent with one learning rate parameter; Number of samples per iteration: 32; Std σ of Gaussian: 0.5. In addition to the standard cartpole task, I considered a setting where I artificially add noise onto the $\phi(\mathbf{w})$ values to simulate a setting where the rewards can only be observed stochastically, and test how my importance sampling methods cope with such noisy measurements. There are additional details in the table and figure captions.

Biped walker setup: State dimension: 24; Action dimension: 4; Policy: neural network with two hidden layers with 40 neurons each and tanh activations, total number of parameters $\mathbf{w}$: 2804; Optimizer: Adam (Kingma and Ba, 2014) with $\beta_1 = 0.99$ and $\beta_2 = 0.999$; Number of samples per iteration: 256; Std σ of Gaussian: 0.04. I used reward normalization (Mania et al., 2018), which is a technique to ensure that the scale of the rewards stays roughly constant by normalizing these with the standard deviation of the sampled returns ϕ, i.e. $\phi(w_i)$ is replaced with $\phi(w_i)/\mathbb{V}[\phi(w)]$, where $\mathbb{V}[\phi(w)]$ is the sample variance of the ϕ values. This appeared to perform better for GLR than rank standardization as used in (Salimans et al., 2017).[7]

Results: The results are in the tables and figures. The errorbars in the tables correspond to the sample standard deviation (so divide by the square root of the sample size to obtain a confidence interval), while the errorbars in the figures are already the standard deviation of the mean. The results act as a sanity check and show that my methods do work, while as expected the difference with standard GLR is small, because the improvement in accuracy is only modest (up to around 1.56 times), especially for high-dimensional sampling spaces. The experiments also show that SLRG can indeed have trouble with systems with a low stochasticity, e.g., the cartpole. The results also confirm that my methods reduce gradient variance in stochastic settings.

[7]Rank standardization computes the ranks of each sample, e.g. the largest value is changed to 1, the second largest to 2 and so on. Then the ranks are divided by the total number of samples and shifted such that the mean is zero.

Table 2.2: Cart-pole swing-up and balancing, no added noise, 32 samples per batch, SGD optimizer, average reward over whole training run, 10 experimental runs for each setting

Learning rate	0.001	0.003	0.005	0.008	0.01
GLR	492.7 ± 15.6	694.4 ± 43.2	716.9 ± 77.4	792.6 ± 31.9	782.9 ± 43.1
SLRG	439.0 ± 12.2	580.3 ± 51.4	664.9 ± 55.2	763.2 ± 52.1	747.4 ± 72.7
TRRG	464.1 ± 5.6	676.6 ± 52.4	771.2 ± 31.7	809.5 ± 21.7	747.9 ± 102.1

Table 2.3: Cart-pole swing-up and balancing, no added noise, 32 samples per batch, SGD optimizer, average reward of last 100 parameter values, 10 experimental runs for each setting

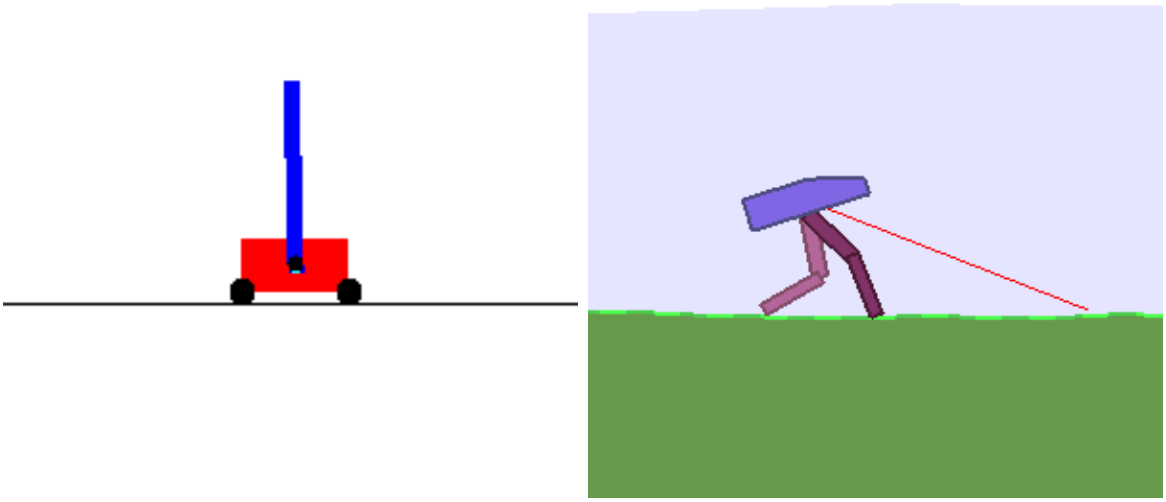
Learning rate	0.001	0.003	0.005	0.008	0.01
GLR	596.7 ± 39.7	881.2 ± 15.3	840.1 ± 103.2	904.4 ± 3.4	889.6 ± 47.4
SLRG	548.8 ± 7.6	723.1 ± 133.8	845.4 ± 82.0	887.0 ± 57.0	895.8 ± 24.7
TRRG	561.8 ± 16.0	867.1 ± 67.4	901.1 ± 3.5	905.8 ± 1.4	840.0 ± 136.0

Table 2.4: Cart-pole swing-up and balancing, 90 added noise standard deviation, 32 samples per batch, SGD optimizer, average reward over whole training run, the number of experimental runs for GLR, SLRG, TRRG were [10,10,50,50,10] for the learning rates from left to right respectively, and 20 runs for BRG in all cases.

Learning rate	0.001	0.003	0.005	0.008	0.01
GLR	519.4 ± 36.9	668.1 ± 72.6	702.7 ± 66.3	690.3 ± 60.8	637.4 ± 42.8
SLRG	459.7 ± 10.4	608.5 ± 51.2	668.9 ± 70.8	710.5 ± 62.6	696.1 ± 64.9
TRRG	485.5 ± 8.0	658.0 ± 64.7	708.1 ± 64.8	699.2 ± 73.7	682.8 ± 71.4
BRG	409.1 ± 12.5	531.3 ± 16.4	600.8 ± 54.4	662.7 ± 70.6	723.0 ± 67.6

Table 2.5: Cart-pole swing-up and balancing, 90 added noise, 32 samples per batch, SGD optimizer, average reward of last 100 parameter values, the number of experimental runs for GLR, SLRG, TRRG were [10,10,50,50,10] for the learning rates from left to right respectively, and 20 runs for BRG in all cases.

Learning rate	0.001	0.003	0.005	0.008	0.01
GLR	639.2 ± 81.6	807.6 ± 101.2	834.0 ± 82.8	810.2 ± 91.2	729.2 ± 103.8
SLRG	552.0 ± 5.9	771.0 ± 115.8	810.4 ± 108.9	845.1 ± 74.2	815.0 ± 83.3
TRRG	574.1 ± 21.4	818.1 ± 107.8	837.0 ± 86.0	814.0 ± 99.1	794.6 ± 121.0
BRG	535.8 ± 8.9	626.6 ± 69.9	736.8 ± 123.0	792.7 ± 115.0	873.5 ± 66.3



(a) Cartpole swing-up task **(b)** Biped walker task

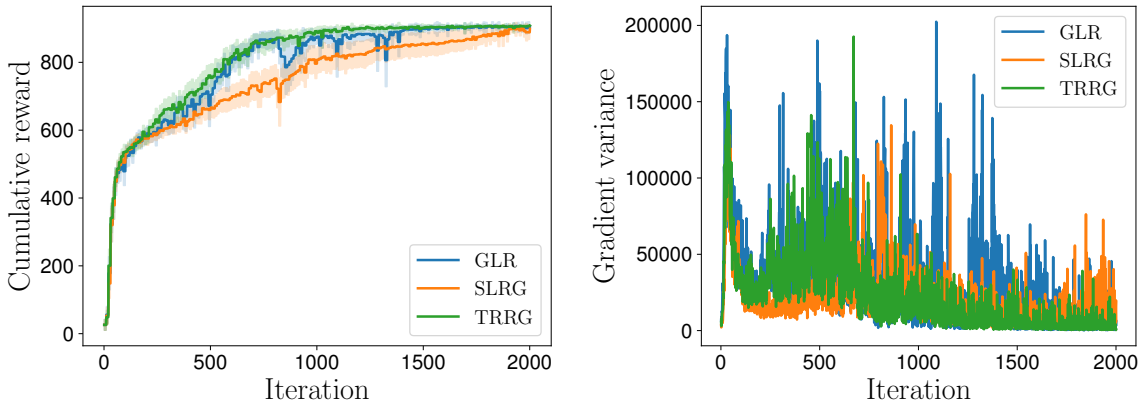
Figure 2.9: Environments used in evolution strategies experiments.

Table 2.6: Biped walker, 256 samples per batch, each parameter sample averaged over 4 episodes, Adam optimizer, reward scaled by standard deviation of rewards, average reward of whole training run, the number of experimental runs were [20,60,40,40,20] for the learning rates from left to right respectively

Learning rate	0.005	0.01	0.015	0.02	0.04
GLR	14.6 ± 30.0	223.2 ± 62.1	264.2 ± 45.7	253.0 ± 51.6	260.9 ± 56.6
TRRG	31.6 ± 42.7	230.0 ± 39.5	250.2 ± 43.9	257.4 ± 46.0	251.2 ± 45.5

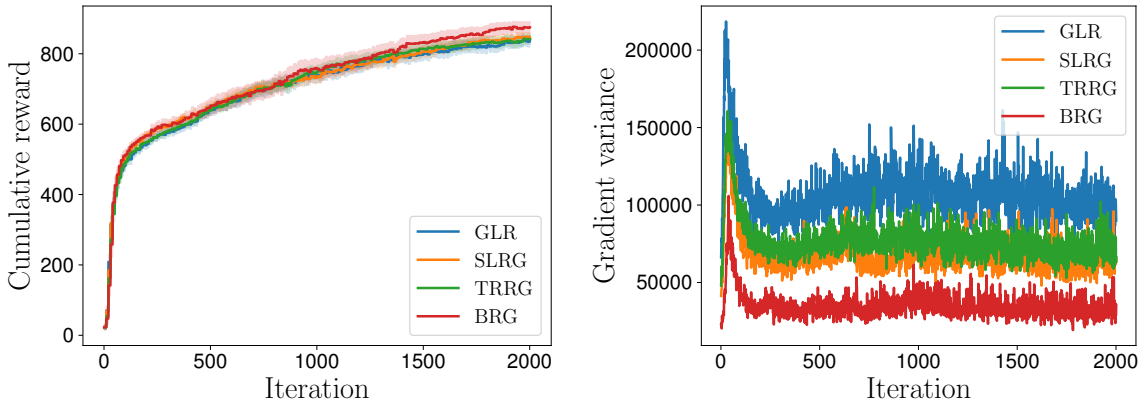
Table 2.7: Biped walker, 256 samples per batch, each parameter sample averaged over 4 episodes, Adam optimizer, reward scaled by standard deviation of rewards, average reward of last 100 parameter values, the number of experimental runs were [20,60,40,40,20] for the learning rates from left to right respectively

Learning rate	0.005	0.01	0.015	0.02	0.04
GLR	38.4 ± 92.2	390.4 ± 61.5	376.6 ± 48.6	345.5 ± 63.4	347.9 ± 63.0
TRRG	104.1 ± 154.2	394.0 ± 34.1	363.3 ± 58.3	353.8 ± 62.3	352.5 ± 63.1



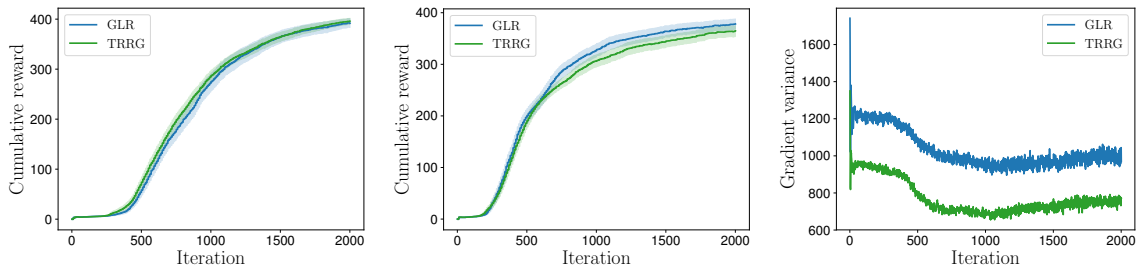
(a) Learning performance **(b)** Gradient variance

Figure 2.10: Cart-pole swingup, no noise, learning rate: 0.008 for all methods, errorbars show 1 standard deviation of the mean, TRRG's $c = 0.5$.



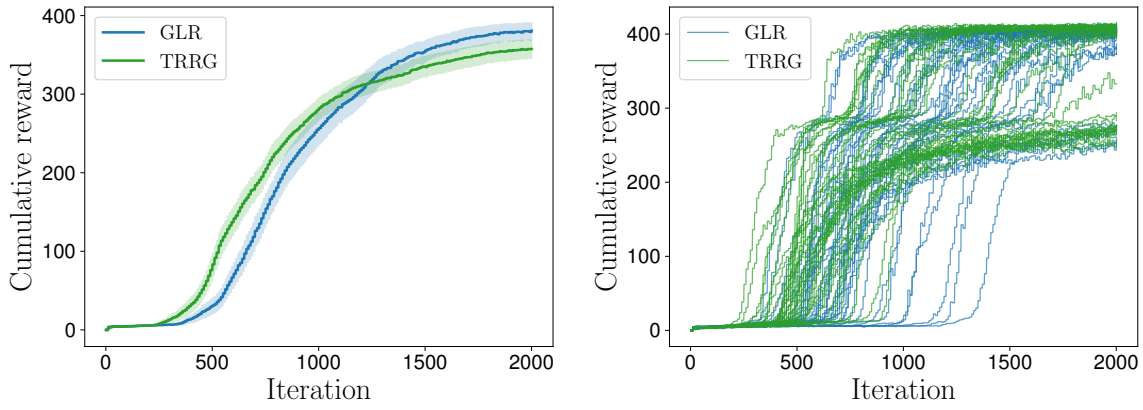
(a) Learning performance **(b)** Gradient variance

Figure 2.11: Cart-pole swingup; noise with std 90 added on cumulative reward; learning rates: GLR: 0.005, SLRG: 0.008, TRRG:0.005, BRG:0.01; errorbars show 1 standard deviation of the mean, BRG's $\alpha = 1.1$, TRRG's $c = 0.5$.



(a) Learning rate: 0.01 **(b)** Learning rate: 0.015 **(c)** Learning rate: 0.01

Figure 2.12: Biped walker; errorbars show 1 standard deviation of the mean, each parameter sample averaged over 4 episodes, TRRG's $c = 0.5$.



(a) Average results **(b)** Raw data

Figure 2.13: Biped walker; learning rate: 0.01, errorbars show 1 standard deviation of the mean, each parameter sample from 1 episode, 40 random number seeds, TRRG's $c = 0.5$; the final reward was bimodal, and while TRRG learned faster, more experiments converged to the lower local minimum.

2.4 Baseline techniques for variance reduction

In this section I discuss another more typical method for likelihood ratio gradient variance reduction: baseline techniques. The basic idea is that subtracting a baseline b from the samples ϕ reduces the gradient variance without introducing any bias. I discuss two common fallacies in some prior works. Both fallacies arise from the fact that often b is estimated from data. The first fallacy is that if b is estimated from data, then the gradient estimator may in fact become biased. The second fallacy is about optimal baselines (Weaver and Tao, 2001)—the process of simply trying to estimate the optimal baseline from data does not lead to the optimal gradient estimating technique, because of variance in estimating the baseline itself.

2.4.1 Preliminaries: Optimal baseline

The LR gradient estimator on its own has a large variance, and techniques must be used to stabilize it. A common technique is to subtract a constant baseline b from the $\phi(x)$ values, so that the gradient estimator becomes

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{x\sim p(x;\theta)}\left[\frac{\mathrm{d}\log p\left(x;\theta\right)}{\mathrm{d}\theta}\left(\phi(x)-b\right)\right]. \tag{2.55}$$

In practice, using $b=\mathbb{E}_{x\sim p(x;\theta)}\left[\phi(x)\right]$ works well, but one can also derive an optimal baseline (Weaver and Tao, 2001). I outline the derivation below. The gradient variance when a baseline is used can be expressed as

$$\begin{aligned}&\mathbb{V}_{x\sim p(x;\theta)}\left[\frac{\mathrm{d}\log p\left(x;\theta\right)}{\mathrm{d}\theta}\left(\phi(x)-b\right)\right]=\\&\mathbb{E}_{x\sim p(x;\theta)}\left[\left(\frac{\mathrm{d}\log p\left(x;\theta\right)}{\mathrm{d}\theta}\phi(x)\right)^2\right]\\&-2\mathbb{E}_{x\sim p(x;\theta)}\left[\left(\frac{\mathrm{d}\log p\left(x;\theta\right)}{\mathrm{d}\theta}\right)^2\phi(x)b\right]+\mathbb{E}_{x\sim p(x;\theta)}\left[\left(\frac{\mathrm{d}\log p\left(x;\theta\right)}{\mathrm{d}\theta}b\right)^2\right].\end{aligned} \tag{2.56}$$

Taking the derivative of Equation (2.56) w.r.t. b and setting to zero gives the optimal baseline as

$$b_{opt}=\frac{\mathbb{E}_{x\sim p(x;\theta)}\left[\left(\frac{\mathrm{d}\log p(x;\theta)}{\mathrm{d}\theta}\right)^2\phi(x)\right]}{\mathbb{E}_{x\sim p(x;\theta)}\left[\left(\frac{\mathrm{d}\log p(x;\theta)}{\mathrm{d}\theta}\right)^2\right]}. \tag{2.57}$$

In practice, for example if $\phi(x)$ is linear and $p\left(x;\theta\right)$ is a Gaussian distribution then $b_{opt}=\mathbb{E}_{x\sim p(x;\theta)}\left[\phi(x)\right]$, so the gain from trying to use an optimal baseline is often small. What would happen to the optimal baseline derivation for my importance sampling case (Sec. 2.2.2)? The derivation in Equation (2.56) has to be modified such that the

sampling distribution $p(x;\theta)$ is changed to $q(x)$, and $\frac{\mathrm{d}\log p(x;\theta)}{\mathrm{d}\theta}$ is changed to $\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}/q(x)$, giving an analogous optimal baseline (Jie and Abbeel, 2010) as

$$b_{opt} = \frac{\mathbb{E}_{x\sim q(x)}\left[\left(\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}/q(x)\right)^2\phi(x)\right]}{\mathbb{E}_{x\sim q(x)}\left[\left(\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}/q(x)\right)^2\right]}. \tag{2.58}$$

Note that if the slice ratio distribution is used, then the optimal baseline takes a particularly convenient form: $q = \left|\frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}\right|/\sqrt{\lambda}$, and $b_{opt} = \mathbb{E}_{x\sim q(x)}[\phi(x)]$.

2.4.2 Bias in gradient estimator with an estimated baseline

It is well known that a constant baseline b can be subtracted from the $\phi(x)$ values without affecting the expected value, while reducing the variance of the estimate. To see this, note that $\mathbb{E}\left[\frac{\mathrm{d}\log p(x;\theta)}{\mathrm{d}\theta}b\right] = \frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}[b] = 0$. However, an often neglected fact is that if the baseline b is estimated from the data, this may add a bias. Consider the case of using a baseline $b = \mathbb{E}[\phi(x)]$. The estimator $E(X)$, where X is a vector of the samples x_i, for $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}[\phi(x)]$ while using N samples is given below.

$$E(X) = \frac{1}{N}\sum_{i=1}^{N}\frac{\mathrm{d}\log p(x_i;\theta)}{\mathrm{d}\theta}\left(\phi(x_i) - \sum_{j=1}^{N}\frac{\phi(x_j)}{N}\right). \tag{2.59}$$

It is easy to see that this estimator will be biased for finite N. For example if one considers $N = 1$, the estimator will always be 0! Now note that $\mathbb{E}\left[\frac{\mathrm{d}\log p(x_i)}{\mathrm{d}\theta}\phi(x_j)\right] = 0$, for $i \neq j$, if we assume independent samples, and $\mathbb{E}\left[\frac{\mathrm{d}\log p(x_i)}{\mathrm{d}\theta}\phi(x_i)\right] = \frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}[\phi(x)]$ is an unbiased estimate of the gradient. Then the expected value of this estimator can be written with the equation:

$$\begin{aligned}
\mathbb{E}[E(X)] &= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}\left[\frac{\mathrm{d}\log p(x_i)}{\mathrm{d}\theta}\left(\phi(x_i)\frac{N-1}{N} + \sum_{j\neq i}^{N}\frac{\phi(x_j)}{N}\right)\right] \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\mathbb{E}\left[\frac{\mathrm{d}\log p(x_i)}{\mathrm{d}\theta}\left(\phi(x_i)\frac{N-1}{N}\right)\right] + \mathbb{E}\left[\frac{\mathrm{d}\log p(x_i)}{\mathrm{d}\theta}\left(\sum_{j\neq i}^{N}\frac{\phi(x_j)}{N}\right)\right]\right) \\
&= \frac{N-1}{N}\frac{\mathrm{d}\mathbb{E}[\phi(x)]}{\mathrm{d}\theta}.
\end{aligned} \tag{2.60}$$

The derivation in Equation (2.60) shows that one can obtain an unbiased estimator by simply rescaling the estimator by $N/(N-1)$. In other words, using the mean of the samples as a baseline changes the expected magnitude of the gradient estimate, but does not change the expected direction of the gradient. For example, previous work suggested estimating two sets of samples, and using one set as a baseline for the other (Tangkaratt et al., 2014), to obtain an unbiased gradient estimator, but my analysis shows that such a method is inefficient, because it wastes half of the samples, and the

gradient estimator has an unbiased direction even if the whole set is used. Such a bias in the magnitude of the gradient is equivalent to a change in the learning rate of the stochastic gradient procedure, so it is unimportant.

What happens if we use a non-uniform weighting for the returns when estimating the baseline? Consider only the contribution to the estimator from the baseline:

$$\mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\frac{\mathrm{d}\log p(x_i)}{\mathrm{d}\theta}\left(-\frac{\sum_{j=1}^{N}w(x_j)\phi(x_j)}{\sum_{j=1}^{N}w(x_j)}\right)\right]. \tag{2.61}$$

Now, take a step back, and consider just the expected value of the baseline, not the gradient. Define $p(X) = p(x_1)p(x_2)...p(x_N)$, $p_i(X_{-i}) = p(x_1)...p(x_{i-1})p(x_{i+1})...p(x_N)$ and $w_i(X) = w(x_i)/\sum_{j=1}^{N} w(x_j)$. One can write the expected value as below.

$$\begin{aligned}
\mathbb{E}\left[\frac{\sum_{j=1}^{N}w(x_j)\phi(x_j)}{\sum_{j=1}^{N}w(x_j)}\right] &= \int_X p(X)\sum_{j=1}^{N} w_j(X)\phi(x_j)dX \\
&= N\int_X p(X)w_1(X)\phi(x_1)\ \mathrm{d}X \\
&= N\int_x p(x_1)\phi(x_1)\left(\int_{X_{-1}} p_1(X_{-1})w_1(X)\ \mathrm{d}X_{-1}\right)\ \mathrm{d}x_1 \\
&= N\int_x p(x_1)\phi(x_1)f(x_1)\ \mathrm{d}x_1\ .
\end{aligned} \tag{2.62}$$

The second line follows from symmetry, and the remaining lines integrate out variables other than x_1. The function $f(x_1)$ is the expected normalized weight assigned to a sample at $x = x_1$. One can see that this will equal $\mathbb{E}[\phi(x)]$ for a general $\phi(x)$ iff $f(x_1) = 1/N$ (for example setting $\phi(x) = \delta(x - x_1)$ to the Dirac delta function will give this result).[8] Note that the expected weight is independent of x_i, so $w(x_i) = const$, which corresponds to the mean baseline. For $f(x_i) \neq 1/N$ this will be the expectation of a different function $\mathbb{E}[\phi(x)f(x)]$. What would this imply about the gradient? Before giving an answer let me introduce a useful trick for explaining such situations.

[8]The Dirac delta function $\delta(z)$ is a function, s.t. $\delta(z) = 0$ for $z \neq 0$, and $\int_a^b \delta(z)\ \mathrm{d}z = 1$ iff 0 lies between a and b.

Remark: Fixing trick

Consider evaluating the gradient of an expectation in the form $\frac{\mathrm{d}}{\mathrm{d}\theta}\int p(x;\theta)f(x;\theta)\ \mathrm{d}x$. Usually this will equal $\int \frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}f(x;\theta)\ \mathrm{d}x + \int p(x;\theta)\frac{\mathrm{d}f(x;\theta)}{\mathrm{d}\theta}\ \mathrm{d}x$, and this leaves it unclear what the component $\int \frac{\mathrm{d}p(x;\theta)}{\mathrm{d}\theta}f(x;\theta)\ \mathrm{d}x$ is equal to. The fixing trick allows to rewrite this term, as a partial derivative of an expectation. We proceed as follows:

1. Instead of $f(x;\theta)$, consider $f(x;\theta')$, where $\theta' = \theta$.
2. Take the partial derivative, while keeping θ' constant.
3. Write $\frac{\partial}{\partial\theta}\big|_{\theta'}\int p(x;\theta)f(x;\theta')dx = \int \frac{\partial p(x;\theta)}{\partial\theta}\Big|_{\theta'} f(x;\theta')\ \mathrm{d}x + \int p(x;\theta)\ \frac{\partial f(x;\theta')}{\partial\theta}\Big|_{\theta'}\ \mathrm{d}x = \int \frac{\partial p(x;\theta)}{\partial\theta}\Big|_{\theta'} f(x;\theta)\ \mathrm{d}x$, where the last line follows because $\frac{\partial f(x;\theta')}{\partial\theta}\Big|_{\theta'} = 0$, as θ' is kept constant, and because $f(x;\theta') = f(x;\theta)$.

This trick is useful in some of the derivations that I have provided.

Next, I apply the fixing trick to explain which derivative the biased gradient will correspond to. Write $w(x_i) = w(x_i;\theta_2)$ with $\theta_2 = \theta$. Then the likelihood ratio gradient is given by:

$$\begin{aligned}
&\mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\frac{\mathrm{d}\log p(x_i)}{\mathrm{d}\theta}\left(\frac{\sum_{j=1}^{N}w(x_j)\phi(x_j)}{\sum_{j=1}^{N}w(x_j)}\right)\right] \\
&= \mathbb{E}\left[\frac{1}{N}\frac{\mathrm{d}\log p(X)}{\mathrm{d}\theta}\left(\frac{\sum_{j=1}^{N}w(x_j)\phi(x_j)}{\sum_{j=1}^{N}w(x_j)}\right)\right] \\
&= \frac{\partial}{\partial\theta}\Big|_{\theta_2=const}\left(\mathbb{E}\left[\frac{1}{N}\left(\frac{\sum_{j=1}^{N}w(x_j)\phi(x_j)}{\sum_{j=1}^{N}w(x_j)}\right)\right]\right) \\
&= \int p(x_1)\frac{\mathrm{d}\log p(x_1)}{\mathrm{d}\theta}\phi(x_1)f(x_1;\theta_2)\ \mathrm{d}x_1\ .
\end{aligned} \tag{2.63}$$

This is the gradient of the expected value of the function $\phi(x)f(x;\theta_2)$, while keeping $\theta_2 = const$. Similarly to the case with the expectation, one can show that this will equal the gradient of $\mathbb{E}[\phi(x)]$ for arbitrary $\phi(x)$ if and only if $f(x) = 1$, and the direction will be unbiased only if $f(x)$ does not depend on x, which would imply that $w(x) = const$. The derivation shows that if one estimates a baseline by a weighted average of all of the samples, the only baseline that will keep the direction of the gradient unbiased for general reward functions is the mean of the sample returns.

However, there is an easy fix to this problem. Consider estimating a separate baseline b_i for each point x_i, using all of the other points $x_j|j \neq i$. Let Z be an arbitrary random variable, then as $\mathbb{E}\left[\frac{\mathrm{d}\log p(x_i)}{\mathrm{d}\theta}Z\right] = 0$ if $\frac{\mathrm{d}\log p(x_i)}{\mathrm{d}\theta}$ and Z are statistically independent, the method of estimating the baseline for one sample using other statistically independent samples will provide a baseline without biasing the gradient.

The additional computational cost will usually be negligible. Estimating N separate baselines for each of the N samples can be evaluated efficiently using roughly 3 times the amount of operations as is needed to estimate a single baseline by first evaluating the sums $A = \sum_{i=1}^{N} w(x_i)$ and $B = \sum_{i=1}^{N} w(x_i)\phi(x_i)$, then evaluating each baseline $b_i = (B - w(x_i)\phi(x_i))/(A - w(x_i))$. This debiasing technique was previously given by Mnih and Rezende (2016), and I independently rediscovered it. One difference is that they did not provide the analysis of the magnitude of the bias or the other analyses I performed.

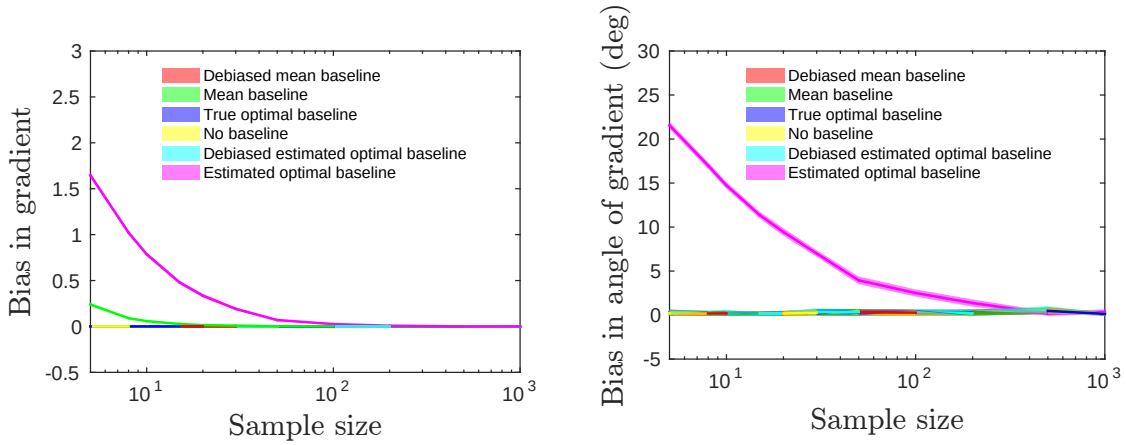
The optimal baseline is given by $\frac{\mathbb{E}\left[(\frac{\mathrm{d}\log p(x_i)\theta}{\mathrm{d})}^2\phi(x_i)\right]}{\mathbb{E}\left[(\frac{\mathrm{d}\log p(x_i)}{\mathrm{d}\theta})^2\right]}$, so one might be tempted to think that the optimal way to estimate a baseline would be to use the above trick of using all but one sample to estimate a separate "optimal baseline" for each sample. However, I will show next that this is suboptimal.

2.4.3 Effect of the variance of the baseline estimator

The previous derivations have ignored the fact that b is estimated from data, hence it is itself also a random variable. I give a derivation that considers this below. In the below derivation, I assume that the leave-one-out technique is used, so that b can be considered statistically independent from $\frac{\mathrm{d}\log p(x;\theta)}{\mathrm{d}\theta}$. Moreover, I ignore covariance terms between different samples. The derivation is illustratory to highlight the concept.

$$
\begin{aligned}
&\mathbb{V}\left[\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\left(\phi(x)-b\right)\right] = \\
&\quad\mathbb{E}\left[\left(\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\left(\phi(x)-b\right)\right)^2\right] - \mathbb{E}\left[\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\left(\phi(x)-b\right)\right]^2 = \\
&\quad\mathbb{E}\left[\left(\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\right)^2\phi(x)^2\right] - 2\mathbb{E}\left[\left(\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\right)^2\phi(x)\right]\mathbb{E}\left[b\right] + \mathbb{E}\left[\left(\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\right)^2\right]\mathbb{E}\left[b^2\right] \\
&\qquad - \mathbb{E}\left[\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\phi(x)\right]^2 = \\
&\quad\mathbb{E}\left[\left(\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\right)^2\phi(x)^2\right] - 2\mathbb{E}\left[\left(\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\right)^2\phi(x)\right]\mathbb{E}\left[b\right] \\
&\qquad + \mathbb{E}\left[\left(\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\right)^2\right]\left(\mathbb{E}\left[b\right]^2 + \mathbb{V}\left[b\right]\right) - \mathbb{E}\left[\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\phi(x)\right]^2 = \\
&\quad\mathbb{V}\left[\left(\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\right)\phi(x)\right] - 2\mathbb{E}\left[\left(\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\right)^2\phi(x)\right]\mathbb{E}\left[b\right] \\
&\qquad\qquad + \mathbb{E}\left[\left(\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\right)^2\right]\left(\mathbb{E}\left[b\right]^2 + \mathbb{V}\left[b\right]\right)
\end{aligned}
\tag{2.64}
$$

The above derivation assumes that b is statistically independent from $\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}$, which



(a) Bias in estimator **(b)** Bias in angle of estimator

Figure 2.14: The mean baseline leads to an unbiased gradient direction, but a biased magnitude.

means one can ignore the b in the last component, i.e., swapping the $\phi(x) - b$ to $\phi(x)$ in the last term between the second to third line is valid. Compared to the standard derivation, my derivation includes an additional $\mathbb{E}\left[\left(\frac{\mathrm{d}\log p(x)}{\mathrm{d}\theta}\right)^2\right]\mathbb{V}[b]$ term, showing that the variance of the baseline estimator itself also matters. Note that the above derivation is not fully complete because it ignores the covariance terms between separate members of the gradient samples, i.e., it ignores the covariance terms between $\frac{\mathrm{d}\log p(x_i)}{\mathrm{d}\theta}(\phi(x_i) - b_i)$ and $\frac{\mathrm{d}\log p(x_j)}{\mathrm{d}\theta}(\phi(x_j) - b_j)$ for $i \neq j$. But the derivation conveys the main idea that the variance of the baseline estimation technique should also be taken into account. In the experimental results in the next section, I show that indeed simply estimating the optimal baseline from the samples can even lead to a worse gradient estimator than when the mean baseline is estimated from the samples. I have also performed some preliminary work on obtaining gradient estimators that would outperform both the mean and optimal baseline estimation techniques, but the work is preliminary, and the improvement not so significant, so I leave it out of my thesis.

2.5 Toy experiments to test theory

Testing bias when using baselines: To test the theory of the bias when using baselines introduced in Section 2.4.2, I performed a simple toy experiment. I chose $\phi(\mathbf{x})$ to be a mixture of two Gaussian basis functions with centers at $\mathbf{m}_1 = [0, 1]$ and $\mathbf{m}_2 = [5, 0]$, and covariances of $\Sigma = 0.5I$, where I is the identity matrix. The weights assigned to the basis functions were 10 and 2000 respectively. The sampling distribution $p(\mathbf{x}; \theta)$ was a Gaussian with its center at $\mu = [0, 0]$ and a covariance $\Sigma = I$ equal to the identity matrix. I estimated the bias by computing the exact gradient analytically, and estimating the expected deviation squared from this exact gradient by repeatedly sampling the estimators for a large number of times. I estimated confidence intervals

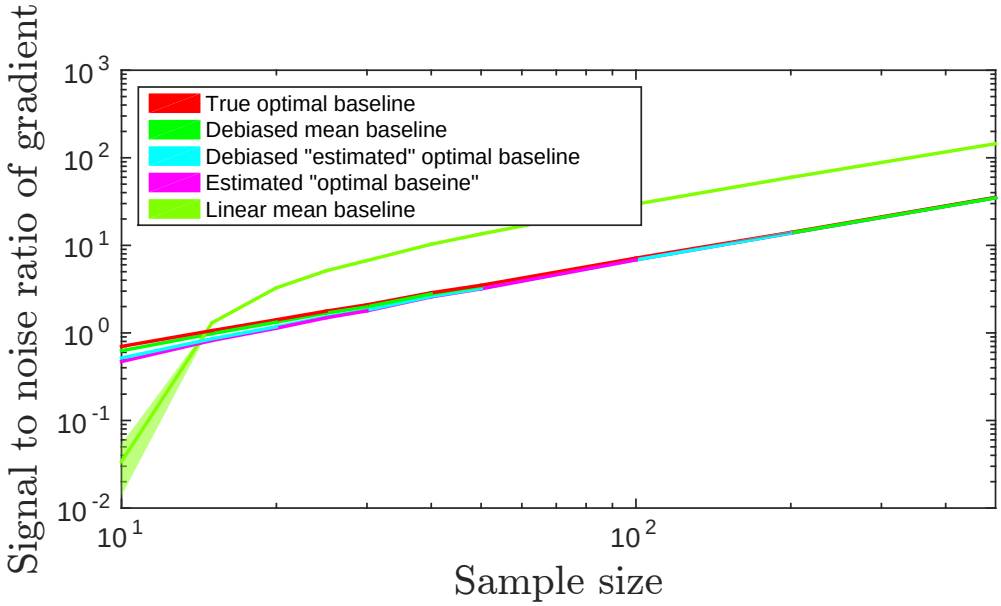


Figure 2.15: Comparing signal-to-noise ratios of LR gradient variance using different baselines on a 10-dimensional quadratic problem

using bootstrapping. The result is in Figure 2.14. As can be seen, the mean baseline has a biased magnitude, but an unbiased direction of the gradient. When adding my $N/(N-1)$ correction factor, the bias in the magnitude also disappears. On the other hand, the optimal baseline has both a biased magnitude and direction, but when the leave-one-out bias removal is applied, it becomes unbiased.

Signal-to-noise ratio of optimal baseline estimator: To test the theory about the variance of the baseline affecting the result in Section 2.4.3, I performed a simple toy experiment, where I estimated the signal-to-noise ratio of the gradient estimators by repeatedly sampling the estimator many times, and using bootstrapping to estimate the confidence intervals. $\phi(\mathbf{x})$ was a quadratic with dimension $D = 10$. The function was given by $\phi(\mathbf{x}) = \mathbf{x}^T B\mathbf{x} + A\mathbf{x} + b$, where B was generated as $Q + Q^T + 5I$, where Q was a matrix of size $D \times D$, where each entry was sampled from a unit variance Gaussian; A was a vector of length D, where each member was sampled from a unit variance Gaussian; and $b = 10$. The sampling distribution was Gaussian with $\mu = 3 \times \mathbf{1}$, and $\Sigma = 2I + 0.2\epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$. The results are in Figure 2.15. As can be seen, using the mean baseline actually outperforms the estimated optimal baseline in terms of signal-to-noise ratio at low sample sizes. Moreover, it is not much worse than using the true optimal baseline, when this is perfectly estimated. I also compared to a linear baseline fit onto the samples by linear regression, and this performed better, so it seems that such more advanced baselines are a good topic for further research.

Chapter 3

Probabilistic computation graphs for gradient estimation

In Chapter 2, I explained methods for estimating gradients of an expectation through a single sampling operation: $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}_{x\sim p(x;\theta)}\left[\phi(x)\right]$. However, usually gradient estimators are needed not just through a single computation, but through a computation graph containing arbitrary stochastic and deterministic computations. Previously, I assumed that the details of the computations performed by $\phi(x)$ are not known. However, typically such information is available. So we should take advantage of this information to construct better gradient estimators. The main topic of this chapter is an intuitive visual framework for reasoning about gradients on graphs of computations to enable deriving such better new gradient estimators.

An overview of application areas of gradient estimators on graphs of computations is given by (Schulman et al., 2015). The same authors also provided a method to obtain gradient estimators on stochastic computation graphs (SCG) by differentiating a surrogate loss. While the work provided an elegant method to obtain gradient estimators using automatic differentiation, the resulting framework has formal rules, which uniquely define one specific type of estimator, and it is not suitable for describing general gradient estimation techniques. For example, deterministic policy gradients (Silver et al., 2014) or total propagation (Parmas et al., 2018) are not covered by the framework. While later, shortly after my work described in this chapter, the framework was extended to include also some other estimators (Weber et al., 2019), each new estimator requires a new derivation, and the notation in the graphs becomes cumbersome compared to the original framework. In contrast, in probabilistic inference, the successful probabilistic graphical model framework (Pearl, 2014) only describes the structure of a model, while there are many different choices of algorithms to perform inference. In this chapter, I aim for a similar framework for gradient computation, which I call *probabilistic computation graphs*. My framework uses the total derivative rule $\frac{\mathrm{d}f}{\mathrm{d}a} = \frac{\partial f}{\partial a} + \frac{\partial f}{\partial b}\frac{\mathrm{d}b}{\mathrm{d}a}$ to decompose the gradient into a sum of partial derivatives along different computational paths, while leaving open the choice of estimator for the partial derivatives.

I begin with explaining my framework in Section 3.1, then discuss relationships to previous gradient estimation techniques in the literature, and finally show how my method can be used to derive new gradient estimators by giving two examples:

1. Gaussian shaping gradients (GS), 2. density estimation likelihood ratio gradients (DEL). Experimental results will be postponed until Chapter 4.

3.1 Total stochastic gradient theorem

Here I explain how my framework helps with decomposing the gradient of a complicated graph of computations into smaller sections, which can be readily estimated using the methods for gradient estimation in Chapter 2. In my framework, I work with the gradient of the marginal distribution—instead of looking just at how the expectation $\mathbb{E}[\phi(\mathbf{x})]$ changes as some parameters in the computation graph are perturbed, I consider the change in the whole distribution of $\phi(\mathbf{x})$. This more general problem directly gives one the gradient of the expectation as well, as the expectation is just a function of the marginal distribution.

3.1.1 Explanation of framework

I define *probabilistic computation graphs* (PCG). The definition is exactly equivalent to the definition of a standard directed graphical model, but it highlights my methods better, and emphasizes my interest in computing gradients, rather than performing inference. The main difference is the explicit inclusion of the *distribution parameters* ζ, e.g. for a Gaussian, the mean μ and covariance Σ. Previously, θ served the role of the distribution parameters, but here I introduce a new notation to emphasize that each node in the PCG will have some parameterized distribution and to highlight the difference from the policy parameters in reinforcement learning, which are usually denoted by θ.

Definition 1 (Probabilistic computation graph (PCG)) *A directed acyclic graph with nodes/vertices V and edges E, which satisfy the following properties:*

1. *Each node $i \in V$ corresponds to a collection of random variables with marginal joint probability density $p(\mathbf{x}_i; \zeta_i)$, where ζ_i are the possibly infinite parameters of the distribution. Note that the parameterization is not unique, and any parameterization is acceptable.*

2. *The probability density at each node is conditionally dependent on the parent nodes: $p(\mathbf{x}_i|\mathbf{Pa}_i)$ where $\mathbf{Pa}_i$ are the random variables at the direct parents of node i.*

3. *The joint probability density satisfies: $p(\mathbf{x}_1, ..., \mathbf{x}_n) = \prod_{i=1}^{n} p(\mathbf{x}_i|\mathbf{Pa}_i)$*

4. *Each ζ_i is a function of its parents: $\zeta_i = f(\mathbf{Pz}_i)$ where $\mathbf{Pz}_i$ are the distribution parameters at the parents of node i. In particular:*
   $p(\mathbf{x}_i; \zeta_i) = \int p(\mathbf{x}_i|\mathbf{Pa}_i)p(\mathbf{Pa}_i; \mathbf{Pz}_i)\mathrm{d}\mathbf{Pa}_i$

I emphasize that there is nothing stochastic in my formulation. Each computation is deterministic, although they may be analytically intractable. I also emphasize that this definition does not exclude deterministic nodes, i.e., the distribution at a node may be a Dirac delta function (a point mass). Later I will use this formulation to derive stochastic estimates of the gradients.

3.1.2 Derivation of theorem

I am interested in computing the total derivative of the distribution parameters at one node ζ_i w.r.t. the parameters at another node $\mathrm{d}\zeta_i/\mathrm{d}\zeta_j$, e.g., nodes i and j could correspond to ϕ and $\mathbf{x}$ in the expectation equation $\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x};\theta)}[\phi(\mathbf{x})]$. In this case, $\zeta_i = \theta$, and ζ_ϕ are the parameters describing the full distribution of ϕ, and is obtained by pushing the distribution of $\mathbf{x}$ through the mapping $\phi(\mathbf{x})$ (Fig. 3.1). Note that the expectation of ϕ is one possible choice of parameters in the vector ζ_ϕ. Hence, the gradient of an expectation reduces to my explanation, and I am describing a more general problem. By the total derivative rule: $\frac{\mathrm{d}\zeta_i}{\mathrm{d}\zeta_j} = \sum_{\zeta_m \in \mathbf{Pz}_i} \frac{\partial \zeta_i}{\partial \zeta_m}\frac{\mathrm{d}\zeta_m}{\mathrm{d}\zeta_j}$. Iterating this equation on the $\mathrm{d}\zeta_m/\mathrm{d}\zeta_j$ terms leads to a sum over paths from node j to node i:

Figure 3.1: Graph for $\phi(\mathbf{x})$

$$\frac{\mathrm{d}\zeta_i}{\mathrm{d}\zeta_j} = \sum_{Paths(j\to i)} \prod_{Edges(k,l)\in Path} \frac{\partial \zeta_l}{\partial \zeta_k} \tag{3.1}$$

This equation holds for any deterministic computation graph and is also well known in, e.g., the optimal Jacobian accumulation community (Naumann, 2008). This equation trivially leads to the *total stochastic gradient theorem*, which states that the sum over paths from A to B can be written as a sum over paths from A to intermediate nodes and from the intermediate nodes to B. Figure 3.2 provides examples of the paths in Equation (3.2) below.

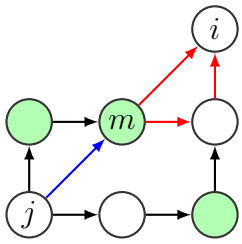
Theorem 3 (Total stochastic gradient theorem) *Let i and j be distinct nodes in a probabilistic computation graph, and let IN be any set of intermediate nodes, which block the paths from j to i, i.e., IN is such that there does not exist a path from j to i that does not pass through a node in IN. We denote, $\{a \to b\}$ the set of paths from a to b, and $\{a \to b\}\backslash c$ the set of paths from a to b, where no node along the path except for b is allowed to be in set c. Then the total derivative $\mathrm{d}\zeta_i/\mathrm{d}\zeta_j$ can be written with the equation:*

$$\frac{\mathrm{d}\zeta_i}{\mathrm{d}\zeta_j} = \sum_{m\in IN}\left(\left(\sum_{s\in\{m\to i\}}\prod_{(k,l)\in s}\frac{\partial\zeta_l}{\partial\zeta_k}\right)\left(\sum_{r\in\{j\to m\}\backslash IN}\prod_{(p,t)\in r}\frac{\partial\zeta_t}{\partial\zeta_p}\right)\right) \tag{3.2}$$

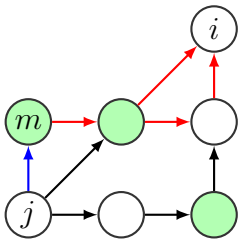
Equations (3.1) and (3.2) can be combined to give:

$$\frac{\mathrm{d}\zeta_i}{\mathrm{d}\zeta_j} = \sum_{m\in IN}\left(\left(\frac{\mathrm{d}\zeta_i}{\mathrm{d}\zeta_m}\right)\left(\sum_{r\in\{j\to m\}\backslash IN}\prod_{(p,t)\in r}\frac{\partial\zeta_t}{\partial\zeta_p}\right)\right) \tag{3.3}$$

Note that an analogous theorem could be derived by swapping $r \in \{j \to m\}\backslash IN$ and $s \in \{m \to i\}$ with $r \in \{j \to m\}$ and $s \in \{m \to i\}\backslash IN$ respectively. This leads to



(a) $\{j \to m\}$ paths may not pass through green nodes.



(b) $\{m \to i\}$ paths may pass through green nodes.

Figure 3.2: Example paths in Equation (3.2). The green nodes correspond to the intermediate nodes IN.

the equation below:

$$\frac{\mathrm{d}\zeta_i}{\mathrm{d}\zeta_j} = \sum_{m \in IN} \left(\left(\sum_{r \in \{m \to i\} \setminus IN} \prod_{(p,t) \in r} \frac{\partial \zeta_t}{\partial \zeta_p} \right) \left(\frac{\mathrm{d}\zeta_m}{\mathrm{d}\zeta_j} \right) \right) \tag{3.4}$$

I will refer to Equations (3.3) and (3.4) as the second and first half *total gradient equations* respectively.

3.1.3 Gradient estimation on a graph

Here I clarify one method how the partial derivatives through the nodes $m \in IN$ in the previous section can be estimated. Before explaining how to combine the estimators on a graph, I introduce the categorization of *pathwise gradient estimators* and *jump gradient estimators*. The pathwise gradient estimator naming is well-known (Schulman et al., 2015), whereas the term jump gradient estimator is new. The categorization is based on the following properties of the estimators:

- *Pathwise derivative estimators* compute partial derivatives through a single edge, e.g., $\frac{\partial \zeta_m}{\partial \zeta_j}$ (Fig. 3.2). For example, RP gradients belong to this category.

- *Jump gradient estimators* sum the gradients across all computational paths between two nodes and directly compute total derivatives, e.g., $\frac{\mathrm{d}\zeta_i}{\mathrm{d}\zeta_m}$ (Fig. 3.2). For example, LR gradients or deterministic policy gradients (Sec. 1.1.1) belong to this category.

By appropriately combining these estimators, various different gradient estimation schemes can be created. Below, I outline just one possibility. The task is to estimate the derivative of the expectation at a distal node i w.r.t. the parameters at an earlier node j: $\frac{\mathrm{d}}{\mathrm{d}\zeta_j} \mathbb{E}_{\mathbf{x}_i \sim p(\mathbf{x}_i;\zeta_i)} [\mathbf{x}_i]$, through an intermediate node m. Note that $\mathbb{E}[\mathbf{x}_i]$ can be picked as one of the distribution parameters in ζ_i. The true ζ are intractable, so I perform an ancestral sampling based estimate $\hat{\zeta}$, i.e., I sample sequentially from each $p(\mathbf{x}_*|\mathrm{P}_*)$ to get a sample through the whole graph, then $\hat{\zeta}_*$ will simply be the parameters of $p(\mathbf{x}_*|\mathrm{Pa}_*)$. I refer to one such sample as a *particle*. I use a batch of P such particles $\hat{\zeta}_* = \{\hat{\zeta}_{*,c}\}_c^P$ to

obtain a mixture distribution as an approximation to the true distribution. Such a sampling procedure has the properties $p(\mathbf{x};\zeta) = \int p\left(\mathbf{x};\hat{\zeta}\right) p\left(\hat{\zeta}\right) \mathrm{d}\hat{\zeta}$ and $\mathbb{E}_{\mathbf{x}_i \sim p(\mathbf{x}_i;\zeta_i)}[\mathbf{x}_i] = \mathbb{E}_{\hat{\zeta}_i \sim p\left(\hat{\zeta}_i;\zeta_j\right)}\left[\mathbb{E}_{\mathbf{x}_i \sim p\left(\mathbf{x}_i;\hat{\zeta}_i\right)}[\mathbf{x}_i]\right]$. For simplicity in the explanation, I further assume that the sampling is reparameterizable, i.e. $p\left(\hat{\zeta}_m;\zeta_j\right) = \int f(\hat{\zeta}_m;\zeta_j,\epsilon_m) p(\epsilon_m)\,\mathrm{d}\epsilon_m$. One can write $\frac{\mathrm{d}}{\mathrm{d}\zeta_j}\mathbb{E}_{\hat{\zeta}_i \sim p\left(\hat{\zeta}_i;\zeta_j\right)}\left[\mathbb{E}_{\mathbf{x}_i \sim p\left(\mathbf{x}_i;\hat{\zeta}_i\right)}[\mathbf{x}_i]\right] = \mathbb{E}_{\epsilon_m \sim p(\epsilon_m)}\left[\frac{\partial \hat{\zeta}_m}{\partial \zeta_j}\frac{\mathrm{d}}{\mathrm{d}\hat{\zeta}_m}\mathbb{E}_{\mathbf{x}_i \sim p\left(\mathbf{x}_i;\hat{\zeta}_i\right)}[\mathbf{x}_i]\right]$. The term $\frac{\partial \hat{\zeta}_m}{\partial \zeta_j}$ will be estimated with a pathwise derivative estimator. The remaining term $\frac{\mathrm{d}}{\mathrm{d}\hat{\zeta}_m}\mathbb{E}_{\mathbf{x}_i \sim p\left(\mathbf{x}_i;\hat{\zeta}_i\right)}[\mathbf{x}_i]$ will be estimated with any other estimator, e.g., a jump estimator could be used.

I summarize the process for creating gradient estimators from j to i on the graph:

1. Choose a set of intermediate nodes IN, which block the paths from j to i.
2. Construct pathwise derivative estimators from j to the intermediate nodes IN.
3. Construct total derivative estimators from IN to i, and apply Equation (3.3) to combine the gradients.

3.2 Relationship to various gradient estimators

My framework is a general scheme to decompose the total derivative into partial derivatives in any scenario. It is thus not surprising that most other gradient estimators in the literature can be cast into my framework. Indeed, I have not been able to find a gradient estimator that could not be described by my framework. In this section, I discuss several connections to existing estimators in prior work.

3.2.1 Relationship to policy gradient theorems

Recall the typical model-free RL scenario (Sutton and Barto, 1998): an agent performs actions $\mathbf{u}_t \sim \pi(\mathbf{u}|\mathbf{x}_t;\theta)$ according to a stochastic policy π, transitions through states $\mathbf{x}_t$, and obtains costs c_t (or conversely rewards). The agent's goal is to find the policy parameters θ that optimize the expected return $G = \sum_{t=0}^{H} c_t$ for each episode. The probabilistic computation graph corresponding to this scenario is provided in Figure 3.5a.

In the literature, two "gradient theorems" are widely applied: the policy gradient theorem (Sutton et al., 2000), and the deterministic policy gradient theorem (Silver et al., 2014). These two are equivalent in the limit of no noise (Silver et al., 2014).

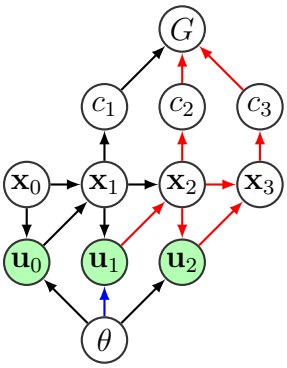


Figure 3.3: Probabilistic computation graph for model-free LR gradient estimation.

Policy gradient theorem

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}\left[G\right] = \mathbb{E}\left[\sum_{t=0}^{H-1} \frac{\mathrm{d}\log\pi(\mathbf{u}_t|\mathbf{x}_t;\theta)}{\mathrm{d}\theta}\hat{Q}_t(\mathbf{u}_t,\mathbf{x}_t)\right]. \tag{3.5}$$

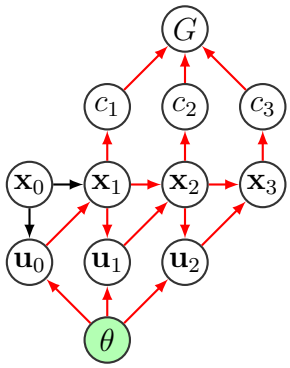
Deterministic policy gradient theorem

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}\left[G\right] = \mathbb{E}\left[\sum_{t=0}^{H-1} \frac{\mathrm{d}\mathbf{u}_t}{\mathrm{d}\theta}\frac{\mathrm{d}\hat{Q}_t(\mathbf{u}_t,\mathbf{x}_t)}{\mathrm{d}\mathbf{u}_t}\right]. \tag{3.6}$$

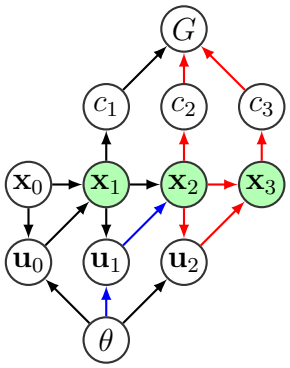
$\hat{Q}_t$ corresponds to an estimator of the remaining return $\sum_{h=t}^{H-1} c_{h+1}$ from a particular state $\mathbf{x}$ when choosing action $\mathbf{u}$. For Equation (3.5) any estimator is acceptable, even a sample based estimate could be used. For Equation (3.6), $\hat{Q}$ is usually a differentiable surrogate model. Figure 3.3 shows how these two theorems correspond to the same probabilistic computation graph. The intermediate nodes are the actions selected at each time step. The difference lies in the choice of jump estimator to estimate the total derivative following the intermediate nodes—the policy gradient theorem uses an LR gradient, whereas the deterministic policy gradient theorem directly differentiates a surrogate model that approximates the remaining cost. I believe that the derivation based on a PCG is more intuitive than previous proofs (Sutton et al., 2000; Silver et al., 2014), which consist of a page of algebra.

3.2.2 Parameter-space sampling based methods

Previously, I introduced the evolution strategies algorithm (Salimans et al., 2017), which is essentially the same but a scaled up version of a prior work, parameter exploring policy gradients (Sehnke et al., 2010). These algorithms sample in the parameter space, rather than in the action space. The PCG is given in Figure 3.4a. Basically, the methods directly estimate the total derivative from θ to the objective node G, by applying the likelihood ratio gradient with samples of θ.



(a) PGPE/ES



(b) Model-based policy gradients

Figure 3.4: Probabilistic computation graphs for policy gradients with parameter exploration (PGPE)/evolution strategies (ES) (3.4a), as well as for model-based state space policy gradient estimation (3.4b).

3.2.3 Model-based gradient estimators

In Section 3.2.1, I showed how the PCG framework can be used to explain model-free policy gradient algorithms. In this section, I show that if one has access to a differentiable model of the dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, then a different type of model-based policy gradient estimator can be created. The PCG is in Figure 3.4b. As can be seen, it is necessary to differentiate through the model—one requires the terms $\frac{\mathrm{d}\log p(\mathbf{x}_{t+1}|\mathbf{x}_t,\pi(\mathbf{x}_t;\theta))}{\mathrm{d}\theta} = \frac{\mathrm{d}\log p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t)}{\mathrm{d}\mathbf{u_t}}\frac{\mathrm{d}\mathbf{u_t}}{\mathrm{d}\theta}$, where the action $\mathbf{u}_t = \pi(\mathbf{x}_t;\theta)$ is given by the policy function. Here, I have assumed that the policy is deterministic, but it would be straight forward to also use a stochastic policy and use reparameterization to differentiate through the sampling. The full gradient estimator becomes

Model-based policy gradient theorem

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}\left[G\right] = \mathbb{E}\left[\sum_{t=1}^{H}\frac{\mathrm{d}\log p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{u}_{t-1})}{\mathrm{d}\mathbf{u}_{t-1}}\frac{\mathrm{d}\mathbf{u}_{t-1}}{\mathrm{d}\theta}\hat{V}_t(\mathbf{x}_t)\right]. \tag{3.7}$$

In the above equation $\hat{V}_t(\mathbf{x}_t)$ is the value function, which is an estimator for the remaining return $G_t = \sum_{h=t}^{H} c_h$ from a particular state $\mathbf{x}_t$. Notice that unlike the model-based case, the return is not conditioned on the action. Similarly to the model-free case, a version of the gradient estimator that directly differentiates a surrogate model $\hat{V}_t(\mathbf{x}_t)$ can be constructed:

Deterministic model-based policy gradient theorem

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}\left[G\right] = \mathbb{E}\left[\sum_{t=1}^{H}\frac{\mathrm{d}\hat{V}_t(\mathbf{x}_t)}{\mathrm{d}\mathbf{x}_t}\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}\mathbf{u}_{t-1}}\frac{\mathrm{d}\mathbf{u}_{t-1}}{\mathrm{d}\theta}\right]. \tag{3.8}$$

In the above equation, it is again necessary to differentiate through the dynamics model to obtain the gradient $\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}\mathbf{u}_{t-1}}$. This kind of gradient estimator has been considered

previously in the literature as a *value gradient method* (Fairbank and Alonso, 2012), and it is for example used in the stochastic value gradients algorithm (Heess et al., 2015), which combines model-free trajectories with a model-based gradient estimator by conditioning the actions on the true trajectory.

3.2.4 Relationship to "Generalized policy gradient theorem"

In this section, I discuss how my framework can be used to interpret another recent attempt at generalizing policy gradient theorems (Ciosek and Whiteson, 2017). While, the work discussed the concept of Rao-Blackwellizing the gradient estimator, i.e. rather than sampling a single action at each state, the quantity $\frac{\mathrm{d}\log\pi(\mathbf{u}_t|\mathbf{x}_t;\theta)}{\mathrm{d}\theta}\hat{Q}_t(\mathbf{u}_t,\mathbf{x}_t)$ can be integrated out across the action at each time step in the trajectory. If $\hat{Q}_t(\mathbf{u}_t,\mathbf{x}_t)$ is a function approximator, this marginalization can be performed locally, without having to sample a separate trajectory for each choice of action, and the resulting gradient estimator is strictly better than the version that does not integrate the action. While this concept makes sense, and is useful, the work contained a "Generalized policy gradient theorem", which I would like to criticize in this section.

The generalized policy gradient theorem is written as:

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}\left[G\right] = \int_s \mathrm{d}\rho(s)\left(\frac{\mathrm{d}V}{\mathrm{d}\theta} - \int_a \mathrm{d}\pi(a|s)\frac{\mathrm{d}Q}{\mathrm{d}\theta}\right), \tag{3.9}$$

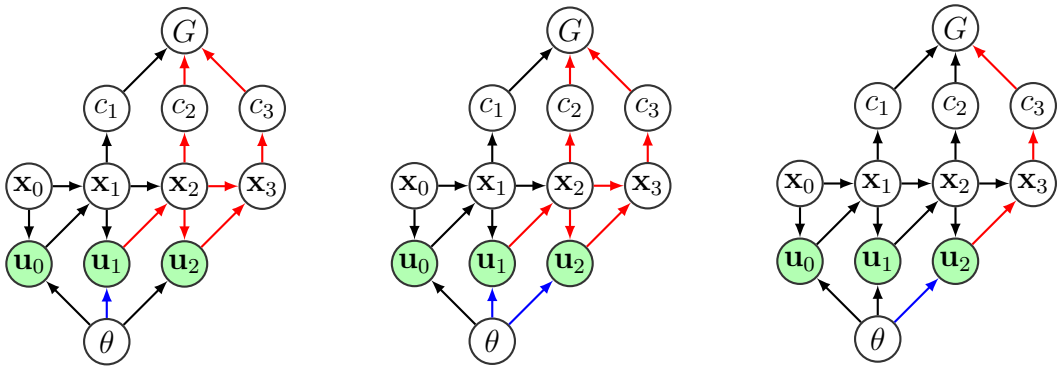
where s is the state, a the action, and $\rho(s)$ the state distribution over the trajectory, and where the authors denoted $\mathrm{d}\rho(s)$ to mean integration with respect to the probability measure $\rho(s)$. One could equivalently write $\rho(s)\mathrm{d}s$. I can explain this theorem in terms of my total stochastic gradient theorem.

First note that $\frac{\mathrm{d}Q}{\mathrm{d}\theta} = \frac{\mathrm{d}}{\mathrm{d}\theta}\int \mathrm{d}p(s'|s,a)\ \ (r + V(s')) = \int \mathrm{d}p(s'|s,a)\frac{\mathrm{d}V(s')}{\mathrm{d}\theta}$, where s' is the next state. One can then write

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}\left[G\right] = \int_s \mathrm{d}\rho(s)\int_a \mathrm{d}\pi(a|s)\int_{s'} \mathrm{d}p(s'|s,a)\left(\frac{\mathrm{d}V}{\mathrm{d}\theta} - \frac{\mathrm{d}V(s')}{\mathrm{d}\theta}\right). \tag{3.10}$$

It is easy to check that this equation is correct: if you sum for all time steps, everything after the first time step cancels out, and one is left with the term $\int_s \mathrm{d}p_0(s)\frac{\mathrm{d}V(s)}{\mathrm{d}\theta}$, which is just the gradient of the expected value over the initial state distribution, which is the objective function. Note that a difference with previous equations is that in the case here, V is not just an approximator, but the true value function, and when it is differentiated w.r.t. θ, one does not simply look at how the state distribution changes, but also considers how the value function itself changes due to the change in the policy at future time steps.

I have explained Equation (3.10) using probabilistic computation graphs in Figure 3.5. To me it seems that the theorem simply re-expresses the computational paths in the previous policy gradient theorems as a subtraction between two different sums of computational paths, and the additional insight appears limited. As far as I have understood, as of now, the theorem has yet to lead to any new algorithm or method either.



(a) Classical policy gradient **(b)** $\frac{\mathrm{d}V}{\mathrm{d}\theta}$ **(c)** $\frac{\mathrm{d}V(s')}{\mathrm{d}\theta}$

Figure 3.5: The policy gradient theorem and deterministic policy gradient theorems both correspond to the graph in Figure 3.5a, whereas Equation (3.10) appears to simply rearrange terms and write the computational paths of Figure 3.5a by subtracting the paths in Figure 3.5c) from the paths in Figure 3.5b. It thus appears to me that rather than the theorem generalizing the previous policy gradient theorems, it is just a mathematical trick that rearranges the terms.

In this section, I wanted to highlight that my probabilistic computation graph framework provides an intuitive visual framework for reasoning about gradient estimators. I believe that such a framework gives a clarity of thought, which can help interpret algebraic equations and avoid pitfalls.

3.3 New gradient estimators

In Section 3.1.3, I explained how a particle-based mixture distribution is used for creating gradient estimators. In the following sections, I instead take advantage of these particles to estimate a different parameterization Γ, directly for the marginal distribution. Although the algorithms have general applicability, to make a concrete example, I explain them in reference to model-based policy gradients using a differentiable model considered in my previous work (Parmas et al., 2018), and discussed in Chapter 4, for which the PCG is given in Figure 3.4b. Experimental results will be postponed until Chapter 4.

3.3.1 Density estimation likelihood ratio gradient (DEL)

Following the explanation in Section 3.3, one could attempt to estimate the distribution parameters Γ from a set of sampled particles, then apply the LR gradient using the estimated distribution $q(\mathbf{x};\Gamma)$. In particular, I will approximate the density as a Gaussian by estimating the mean $\hat{\mu} = \sum_i^P \mathbf{x}_i/P$ and variance $\hat{\Sigma} = \sum_i^P (\mathbf{x}_i - \hat{\mu})^2/(P-1)$. Then, using the standard LR trick, one can estimate the gradient $\sum_i^P \frac{\mathrm{d}\log q(\mathbf{x}_i)}{\mathrm{d}\theta}(G_i - b)$, where $q(\mathbf{x}) = \mathcal{N}(\hat{\mu}, \hat{\Sigma})$. To use this method, one must compute derivatives of $\hat{\mu}$ and $\hat{\Sigma}$ w.r.t. the particles $\mathbf{x}_i$, then carry the gradient to the policy parameters using the chain rule while differentiating through the model, which is straightforward. I refer to the

new method as the DEL estimator. Importantly, note that while $q(\mathbf{x})$ is used for estimating the gradient, it is not in any way used for modifying the trajectory sampling.
Advantages of DEL: One can use LR gradients even if no noise is injected into the computations.
Disadvantages of DEL: The estimator is biased, and density estimation can be difficult.
Experimental evaluation: The experimental results are given in Section 4.5.4, and show that DEL gives a non-trivial success rate, but is still a crude estimator, and may have to be developed further for practical gains.

3.3.2 Distributional/Gaussian shaping gradients (GS)

Until now, all RL methods have used the second half total gradient equation (Eq. 3.3). Might one create estimators that use the first half equation (Eq. 3.4)? Figure 3.6 gives an example of how this might be done. I propose to estimate the density at $\mathbf{x}_m$ by fitting a Gaussian to the particles. Then $\mathrm{d}\mathbb{E}\left[c_m\right]/\mathrm{d}\Gamma_m$ (the pink edges) will be estimated by sampling from this distribution (or by any other method of integration). This leaves the question of how to estimate $\mathrm{d}\Gamma_m/\mathrm{d}\theta$ (all paths from θ to $\mathbf{x}_m$). One option would be to use the RP method, which is straightforward. However, here I consider using the LR method, which leads to a more interesting estimator. I first apply the second half total gradient equation on $\mathrm{d}\Gamma_m/\mathrm{d}\theta$ to obtain terms $\sum_{r\in\{\theta\to x_k\}/IN}\prod_{(p,t)\in r}\frac{\partial\zeta_t}{\partial\zeta_p}$ (blue edges) and $\frac{\mathrm{d}\Gamma_m}{\mathrm{d}\zeta_{x_k}}$ (red edges). In the scenarios I consider, the first of these terms is a single path, and will be estimated using RP. The second term is more interesting, and I will estimate this using an LR method.

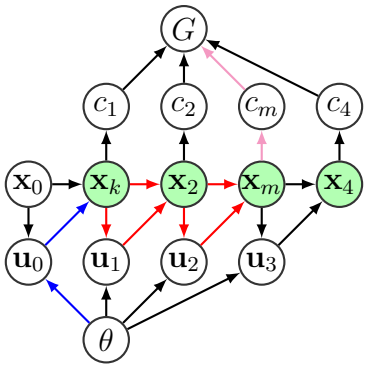


Figure 3.6: Computational paths in Gaussian shaping gradient

As I am using a Gaussian approximation, the distribution parameters Γ_m are the mean and variance of $\mathbf{x}_m$, which can be estimated with samples as $\mu_m = \mathbb{E}\left[\mathbf{x}_m\right]$ and $\Sigma_m = \mathbb{E}\left[\mathbf{x}_m\mathbf{x}_m^T\right] - \mu_m\mu_m^T$. One can obtain LR gradient estimates of these terms:

$$\frac{\mathrm{d}}{\mathrm{d}\zeta_{x_k}}\mathbb{E}\left[\mathbf{x}_m\right] = \mathbb{E}_{\mathbf{x}_k\sim p(\mathbf{x}_k;\zeta_{x_k})}\left[\frac{\mathrm{d}\log p(\mathbf{x_k};\zeta_{x_k})}{\mathrm{d}\zeta_{x_k}}(\mathbf{x}_m - \mathbf{b}_\mu)\right],$$

$$\frac{\mathrm{d}}{\mathrm{d}\zeta_{x_k}}\mathbb{E}\left[\mathbf{x}_m\mathbf{x}_m^T\right] = \mathbb{E}_{\mathbf{x}_k\sim p(\mathbf{x}_k;\zeta_{x_k})}\left[\frac{\mathrm{d}\log p(\mathbf{x_k};\zeta_{x_k})}{\mathrm{d}\zeta_{x_k}}(\mathbf{x}_m\mathbf{x}_m^T - \mathbf{b}_\Sigma)\right],$$

and

$$\frac{\mathrm{d}}{\mathrm{d}\zeta_{x_k}}(\mu\mu^T) = 2\mu\frac{\mathrm{d}}{\mathrm{d}\zeta_{x_k}}\mathbb{E}\left[\mathbf{x}_m^T\right].$$

In practice, one performs a sampling based estimate $\hat{\zeta}_{x_k}$, and there may be concern that the estimators are conditional on the sample $\hat{\zeta}_{x_k}$, but we are interested in unconditional estimates. I will explain that the conditional estimate is equivalent. For the variance,

note that μ_m is an estimate of the unconditional mean, so the whole estimate directly corresponds to an estimate of the unconditional variance. For the mean, apply the rule of iterated expectations: $\mathbb{E}_{\mathbf{x}_k \sim p\left(\mathbf{x}_k;\zeta_{x_k}\right)}\left[\mathbf{x}_m\right] = \mathbb{E}_{\hat{\zeta}_{x_k} \sim p\left(\hat{\zeta}_{x_k}\right)}\left[\mathbb{E}_{\mathbf{x}_k \sim p\left(\mathbf{x}_k;\hat{\zeta}_{x_k}\right)}\left[\mathbf{x}_m\right]\right]$ from which it is clear that the conditional gradient estimate is an unbiased estimator for the gradient of the unconditional mean.

Efficient algorithm for accumulating gradients In Figure 3.6, for each $\mathbf{x}_k$ node, we want to perform an LR jump to every $\mathbf{x}_m$ node after k and compute a gradient with the Gaussian approximation of the distribution at node m. I will accumulate across all nodes during a backwards pass in a backpropagation like manner. Note that for each k and each m, one can write the gradient as $\frac{\mathrm{d}\mathbb{E}[c_m]}{\mathrm{d}\Gamma_m}\frac{\mathrm{d}\Gamma_m}{\mathrm{d}\zeta_{x_k}}\left(\frac{\mathrm{d}\zeta_{x_k}}{\mathrm{d}\mathbf{u}_{k-1}}\frac{\mathrm{d}\mathbf{u}_{k-1}}{\mathrm{d}\theta}\right)$. The term $\frac{\mathrm{d}\mathbb{E}[c_m]}{\mathrm{d}\Gamma_m}\frac{\mathrm{d}\Gamma_m}{\mathrm{d}\zeta_{x_k}}$ is estimated as $\frac{\mathrm{d}\mathbb{E}[c_m]}{\mathrm{d}\Gamma_m}\mathbf{z}_m\frac{\mathrm{d}\log p(\mathbf{x_k};\zeta_{x_k})}{\mathrm{d}\zeta_{x_k}}$, where $\mathbf{z}_m$ corresponds to a vector summarizing the $\mathbf{x}_m - \mathbf{b}_\mu$, etc. terms above. Note that $\frac{\mathrm{d}\mathbb{E}[c_m]}{\mathrm{d}\Gamma_m}\mathbf{z}_m$ is just a scalar quantity g_m. Thus, one can use an algorithm which accumulates a sum of all g during a backwards pass, and sums over all m nodes at each k node. Later, in Chapter 4, I explain in detail how this fits together with my total propagation algorithm (Parmas et al., 2018) (Alg. 6). The final algorithm essentially just replaces the usual cost/reward $c_{m,i}$ for each sample i with a modified value $g_{m,i} = \frac{\mathrm{d}\mathbb{E}[c_m]}{\mathrm{d}\Gamma_m}\mathbf{z}_{m,i}$, and such an approach would also be applicable in model-free policy gradient algorithms using a stochastic policy and LR gradients.

Two interpretations of GS:

1. I am making a Gaussian approximation of the marginal distribution at a node.
2. I am performing a type of reward shaping based on the distribution of the particles. In particular I am essentially promoting the trajectory distributions to stay unimodal, such that all of the particles concentrate at one "island" of reward rather than splitting the distribution between multiple regions of reward—this may simplify optimization.

GS allows for fundamentally new algorithms: Sometimes, in a computational graph there are nodes that cannot be differentiated, e.g., if there is a discrete computation in a graph. Other times, a computational graph can be extremely long, and backpropagating through the whole graph can be computationally impractical. Previously, it was possible to use LR gradients to jump from a node in the computation directly to the terminal node to obtain a gradient estimator, but this is an all-or-nothing approach. It may be better to jump over only part of the computation and continue applying the chain rule from some distal node. Such gradient estimators can be realized using GS, and I am excited about potential future algorithms taking advantage of such an opportunity.

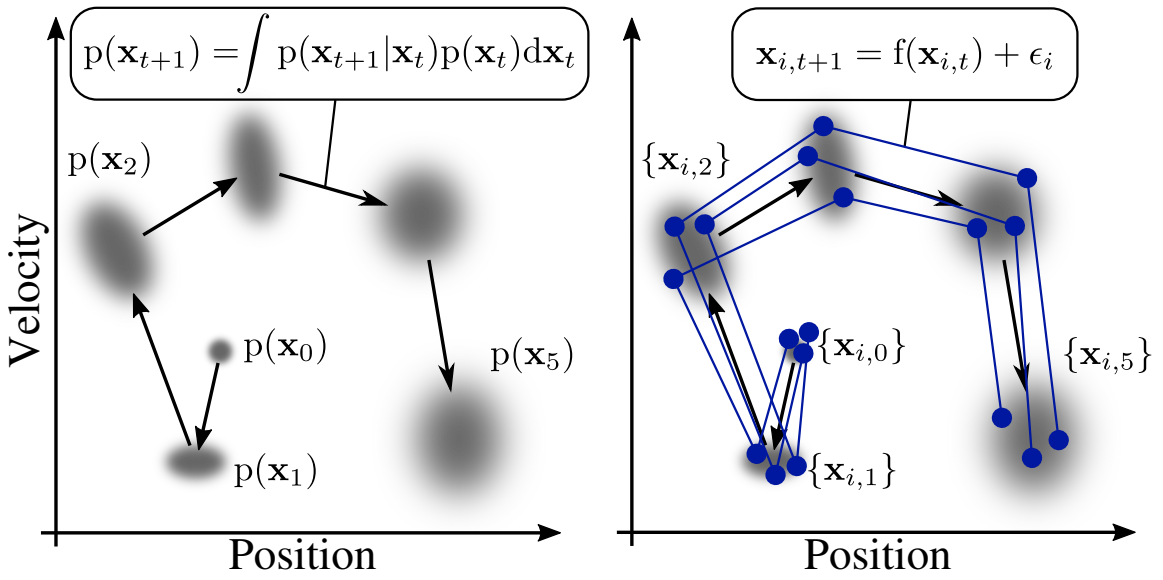
Experimental evaluation: The GS algorithm is evaluated in Section 4.5.4 and is shown to be practical and lead to improved performance in some model-based RL settings.

Chapter 4

Model-based reinforcement learning with particle predictions

A possible naïve approach to model-based reinforcement learning would be to use typical model-free algorithms on simulated trajectories with a learned model, e.g., Dyna (Sutton, 1990); however, such an approach neglects the opportunity to use information about the model to learn in a more efficient manner. For example, it may be possible to differentiate the predictions via backpropagation to efficiently compute the gradient of the expected cost/reward, which could then be used for optimization, as is done in PILCO. Unfortunately, as I explained in the Background section in Chapter 1, the moment matching predictions in PILCO are a severe limitation. To overcome this problem, I suggested that using particle sampling predictions (Fig. 4.1) would be a better approach, which could solve all of the issues. I call the resulting algorithm PIPPS: Probabilistic Inference for Particle-based Policy Search. When one uses sampling, it becomes necessary to differentiate through the stochasticities. The "obvious" idea would be to differentiate through the sampling operations using the reparameterization trick. However, at the end of the Background section in Chapter 1 I hinted at an issue with the reparameterization (RP) gradients in the model-based reinforcement learning setting. In this chapter, I will explain that repeating long chains of non-linear computations often lead to chaotic dynamics, and in this setting, RP is hopelessly poor. I call this phenomenon the *curse of chaos*. This is an inherent property of the environment, and is not caused by numerical bugs. In this chapter, I will show that surprisingly, despite such chaotic properties, it can be possible to estimate accurate gradients. It will turn out that the key to robust gradient estimation in my algorithms is the likelihood ratio gradient (LR) often used in model-free reinforcement learning.

In fact, other researches have also attempted to use backpropagation with model predictions for reinforcement learning. An overview of such algorithms is given by Schmidhuber (2015b), but many researchers found that backpropagation does not work well. For example, the stochastic value gradients paper (Heess et al., 2015) also compared to this approach, and mentioned that it did not work. Works, which found that backpropagation does work either tested on simple/non-standard tasks (Nguyen and Widrow, 1990; Depeweg et al., 2016) or used a stochastic approximation to moment matching (Gal et al., 2016). Recently there has been a surge of very impressive successful model-based reinforcement learning (MBRL) algorithms that sample without



(a) Analytic approximation **(b)** Particle approximation

Figure 4.1: Sampling trajectories rather than performing analytic approximations to the trajectory distribution is a promising solution to overcome limitations of PILCO.

moment matching approximations (Ha and Schmidhuber, 2018; Kurutach et al., 2018; Chua et al., 2018; Hafner et al., 2019), but none of these approaches differentiated through the model predictions. If proper derivatives could be computed through the model, this may greatly improve efficiency of MBRL algorithms. I will begin by explaining experiments describing the curse of chaos, then discuss two potential solutions: 1. using resampling methods, 2. new gradient estimation techniques including the total propagation algorithm. Resampling based methods try to stochastically replicate what moment matching was doing analytically, and I will explain that this approach still suffers from the same issues as moment matching. Moreover, simple ideas to try to extend this approach to multimodal distributions are not straightforward to get to work. Total propagation is an elegant algorithm for combining likelihood ratio and reparameterization gradients, obtaining the best of both estimators. Total propagation based approaches perform standard sampling, thus obtaining an unbiased estimate of the true possibly multimodal distribution, but still achieve the same or better performance than the original PILCO. The work shows that total propagation is a promising approach, which may allow scaling up these kinds of differentiable model-based RL algorithms. Moreover, it is a general purpose algorithm that can be applied in any setting where backpropagation can be used. Hence, it may also be useful in other domains, such as meta learning or training time-series models.

4.1 Preliminaries: model, prediction and gradients

I begin with a brief reminder of the model-based RL setting when particle-based predictions are used. Everything is the same to the standard PILCO, just the framework for performing predictions and evaluating the gradients is changed (see Alg. 3). Two

Algorithm 3 Particle based trajectory prediction and policy evaluation

Input: policy π with parameters θ, episode length T, initial arbitrary state distribution $p(\mathbf{x}_0)$, cost function $c(\mathbf{x})$, learned dynamics model $\hat{f}$, number of particles P.
Initialize: Sample P initial particles $\{\mathbf{x}_{i,0}\}_{i=1}^{P} \sim p(\mathbf{x}_0)$
for $t = 0$ **to** $T - 1$ **do**
  for each particle i **do**
    1. Compute controls: $\mathbf{u}_{i,t} = \pi(\mathbf{x}_{i,t}; \theta)$
    2. Predict next state distribution: $p(\mathbf{x}_{i,t+1}) = \hat{f}(\mathbf{x}_{i,t}, \mathbf{u}_{i,t})$
    3. Sample next particle $\mathbf{x}_{i,t+1} \sim p(\mathbf{x}_{i,t+1})$
  end for
  Average the cost: $\frac{1}{P} \sum_{i=1}^{P} c(\mathbf{x}_{i,t+1})$
end for
Gradient computation: $\frac{\mathrm{d}}{\mathrm{d}\theta} \left(\sum_{t=1}^{T} \mathbb{E}\left[c(\mathbf{x}_{t+1})\right] \right)$ is stochastically approximated from the particles. Various estimators exist, such as the reparameterization gradient or the likelihood ratio gradient estimator.

components are necessary for the predictions: a model $\hat{f}$ and a policy π.

Model: For the purpose of the explanation, it is sufficient to consider the model as a non-linear function that predicts the next state as $\mathbf{x}' = \hat{f}(\mathbf{x}, \mathbf{u}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_f(\mathbf{x})^2 + \sigma_n^2)$, and $\sigma_f(\mathbf{x})^2$ is a term representing the epistemic uncertainty (i.e., uncertainty due to a lack of data in a region), and σ_n^2 represents the aleatoric uncertainty (i.e., inherent noise in the system). See the Gaussian process explanation in Appendix A, and specifics about model learning in Section 1.3.1 for a more detailed explanation. Unlike particles, moment matching predictions can only be applied with certain types of models and policies. In particular it must be possible to analytically compute the expectation and variance of the output, when the input is a Gaussian distribution. This limits the choice of models to sums of Gaussian basis functions, polynomials, sinusoids, etc., but for example deep neural networks are not directly applicable. Particle based predictions on the other hand are completely general, and can easily be applied with any kind of model.

Policy: I usually use a radial basis function network policy (a sum of Gaussians): $\tilde{\pi}(\mathbf{x}; \theta) = \sum_{i=1}^{M} w_i \exp\left(-(\mathbf{x} - \mathbf{m}_i)^T \Lambda^{-1} (\mathbf{x} - \mathbf{m}_i)\right)$, where w_i are the weights for each basis function, $\mathbf{m}_i$ are the centers of the basis functions, and Λ^{-1} are the length scale parameters, which are shared between the different basis functions. Together, Λ^{-1}, $\mathbf{m}_i$ and w_i comprise the policy parameters θ. The outputs from the policies were constrained by a saturation function: $\mathrm{sat}(\tilde{\mathbf{u}}) = 9 \sin(\tilde{\mathbf{u}})/8 + \sin(3\tilde{\mathbf{u}})/8$, where $\tilde{\mathbf{u}} = \tilde{\pi}(\mathbf{x})$. The final output is then $\pi(\mathbf{x}) = U \mathrm{sat}(\tilde{\mathbf{u}})$, where U is the maximum control, e.g., the maximum motor torque. This policy class was used in the original PILCO, and it leads to a reasonably rich class of non-linear policy functions.

Cost function: Typically, the cost function has the form

$$c(\mathbf{x}) = 1 - \exp\left(-(\mathbf{x}-\mathbf{t})^T Q(\mathbf{x}-\mathbf{t})\right), \tag{4.1}$$

where $\mathbf{t}$ is the target, and Q is a weighting function. Such a cost function is roughly quadratic close to the target, but saturates far away from the target. The intuition is that close to the target, deviations are important, but if the agent is reasonably far from the target it does not make a difference how far away it is—a failure is a failure. The cost emphasizes improving the part of the trajectory where things started going wrong, rather than caring about what happened afterwards. In the original PILCO, using such a cost was also partially motivated by experimental evidence that humans use such cost functions (Körding and Wolpert, 2004). More precise details about Q are given in the specific experimental sections.

Gradient estimators: It is possible to use both reparameterization or likelihood ratio gradient estimators in this setting. Using the reparameterization gradient is straightforward. I also previously explained in Section 3.2.3 how to obtain likelihood ratio gradient estimators while using a stochastic model:

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}\left[G\right] = \mathbb{E}\left[\sum_{t=1}^{H} \frac{\mathrm{d}\log p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{u}_{t-1})}{\mathrm{d}\mathbf{u}_{t-1}} \frac{\mathrm{d}\mathbf{u}_{t-1}}{\mathrm{d}\theta} \hat{V}_t(\mathbf{x}_t)\right],$$

where $\hat{V}_t(\mathbf{x}_t)$ is an estimator of the remaining return, e.g., it could be a sample based estimate $G_t = \sum_{h=t}^{T} r_t$. Here I want to introduce another technique for likelihood ratio gradient variance reduction specific to such particle-based prediction situations: the *batch importance weighted likelihood ratio gradient estimator* (BIW-LR). Multiple particles are sampled simultaneously, and thus the state distribution is represented as a mixture distribution $q(\mathbf{x}_{t+1}) = \sum_{i=1}^{P} p(\mathbf{x}_{t+1}|\mathbf{x}_{i,t};\theta)/P$, where P is the number of particles. Analogously to the importance sampling method to derive LR gradients in Section 2.1.1, one can derive a lower variance estimator with importance sampling within the batch for each time step:

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}\left[G_t\right] = \mathbb{E}\left[\sum_{t=0}^{T-1}\left(\frac{\mathrm{d}p(\mathbf{x}_{t+1}|\mathbf{x}_t;\theta)/\mathrm{d}\theta}{q(\mathbf{x}_{t+1})}(G_{t+1}(\mathbf{x}_{t+1}) - b_{t+1})\right)\right]. \tag{4.2}$$

In terms of a specific batch with P particles, the term for each separate time-step becomes $\sum_{i=1}^{P}\sum_{j=1}^{P}\left(\frac{\mathrm{d}p(\mathbf{x}_{j,t+1}|\mathbf{x}_{i,t};\theta)/\mathrm{d}\theta}{\sum_{k=1}^{P} p(\mathbf{x}_{j,t+1}|\mathbf{x}_{k,t})}(G_{t+1}(\mathbf{x}_{j,t+1}) - b_{i,t+1})\right)/P$. I choose to estimate a leave-one-out mean baseline of the returns (Sec. 2.4.2) by normalized importance sampling with the equation:

$$\begin{aligned} b_{i,t+1} &= \left(\sum_{j\neq i}^{P} c_{j,t+1} G_{t+1}(\mathbf{x}_{j,t+1})\right) / \sum_{j\neq i}^{P} c_{j,t+1}\ , \text{ where} \\ c_{j,t+1} &= p(\mathbf{x}_{j,t+1}|\mathbf{x}_{i,t}) / \sum_{k=1}^{P} p(\mathbf{x}_{j,t+1}|\mathbf{x}_{k,t})\ . \end{aligned} \tag{4.3}$$

Without normalizing, a large variance of the baseline estimation leads to poor LR gradients, as is expected from the theory in Section 2.4.3. Note that I compute P baselines for each time-step, whereas there are P^2 components in the gradient estimator. To obtain a true unbiased gradient, one should compute P^2 leave-one-out baselines—one for each particle for each mixture component of the distribution. My thesis contains evaluations only with the baseline presented here—I found that it already removes most of the bias. Finally, note that one may be interested whether such an estimator could be used with the Q-function in model-free reinforcement learning; however, in the model-free case, Q also depends on the state location of the sample, which is different between different particles, so it is not as principled to perform importance sampling within the batch in such a model-free setting. In the next section, I will explain why simply using reparameterization gradients everywhere is not sufficient.

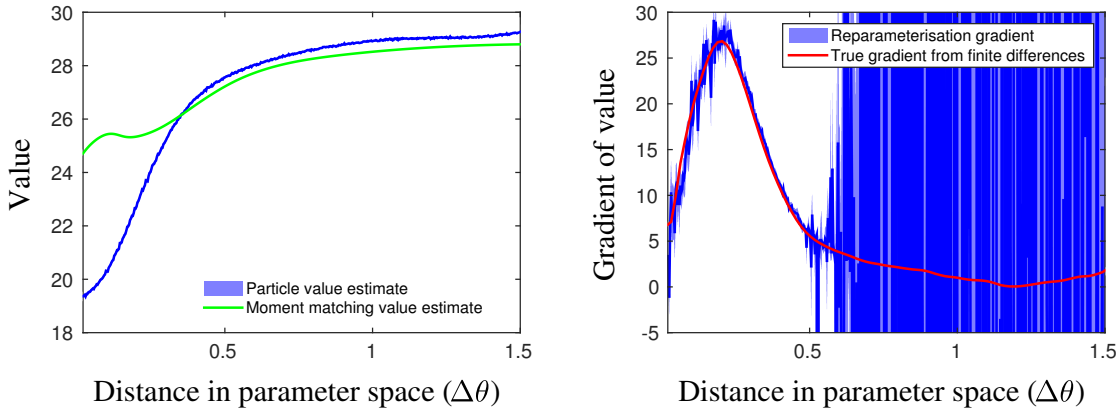
4.2 Explaining the Curse of Chaos

This section contains perhaps the most striking result in my thesis. I will explain that chaotic properties in the dynamics can cause the reparameterization gradient to become ill-behaved. In such situations, I can achieve 10^6 times and more improvement in gradient accuracy by using my other gradient estimators. These experiments were motivated by my initial tests of trying to get particles to work with probabilistic line searches (Mahsereci and Hennig, 2015), but in the end I found that gradient descent performed better.

4.2.1 Value estimator landscape view of chaotic dynamics

Gradient plotting experimental setup: I consider the cart-pole task. The policy is a radial basis function network with 20 basis functions and 104 policy parameters in total. At some point during the training of the policy, I stopped the optimization, picked a direction in the policy parameter space, and plotted what the objective function looks like in this direction. I plot the objective $\sum_{t=1}^{H}\sum_{i=1}^{P} c(\mathbf{x}_{i,t})$ as well as the reparameterization gradient of this objective, w.r.t. the norm of the perturbation in the parameter space $\Delta\theta$. I average across $P = 1000$ particles. Note that all of the parameters are perturbed, not just a single parameter. The perturbation $\Delta\theta$ itself is unimportant; the claim from the experiment is that there exist regions in the policy parameter space that act one way or another. The direction of the perturbation was chosen based on the gradient from the moment matching estimate of the trajectory and objective. Moreover, I had fixed the random number seed, turning the problem deterministic, so that the reparameterization gradient becomes the exact gradient of the objective function. The results are plotted in Figure 4.2.

Gradient plotting discussion: Figure 4.2b contains a peculiar result where the RP gradient behaves well for some regions of the parameters θ, but when θ is perturbed, a phase-transition-like change causes the variance to explode. The variance at $\Delta\theta = 1.5$ is $\sim 4 \times 10^5$ times larger than at $\Delta\theta = 0$, meaning that $\sim 4 \times 10^8$ particles would



(a) $\mathbb{E}[G]$ estimated with 1000 particles **(b)** $\mathrm{RP}_{\mathrm{FS}}$ (Exact gradient of value in 4.2a)

Figure 4.2: To illustrate the behavior of the gradient estimators in the cart-pole task, I fix the random number seed, vary the policy parameters θ in a fixed direction, and plot a 95% confidence interval for both the objective (4.2a) and the magnitude of the RP gradient estimator in this direction (4.2b). Figure 4.2b shows that reparameterization gradients suffer from the curse of chaos, which can cause the gradient variance to explode.

be necessary for the RP gradient to become accurate in that region. For practical purposes, optimizing with the RP gradient would lead to a simple random walk.

Moreover, since the seed was fixed, the RP gradient in Figure 4.2b is the *exact* gradient of the value in Figure 4.2a. Therefore, there is an infinitesimal deterministic "noise" at the right of Figure 4.2a. What is the reason for such a bizarre phenomenon? I perform another experiment to elucidate the cause.

Value estimator plotting: Recall from Section 3.2.3 that the gradient of the expectation can be written as $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}[G] = \mathbb{E}\left[\sum_{t=1}^{H} \frac{\mathrm{d}\hat{V}_t(\mathbf{x}_t)}{\mathrm{d}\mathbf{x}_t}\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}\mathbf{u}_{t-1}}\frac{\mathrm{d}\mathbf{u}_{t-1}}{\mathrm{d}\theta}\right]$, where $\hat{V}_t(\mathbf{x}_t)$ is an estimator for the remaining cumulative cost from $\mathbf{x}_t$. My aim will be to visualize $\hat{V}_t$ at $t = 1$, so as to explain the issue with the gradients. I plot $\hat{V}_t$ at the distribution $p(\mathbf{x}_1)$, by placing a grid at $p(\mathbf{x}_1)$, and computing the remaining return from each grid point. I do this for both the settings when $\Delta\theta = 0$, corresponding to the well-behaved case, and when $\Delta\theta = 1.5$, corresponding to the ill-behaved case. The boxes (Figs. 4.4a, 4.4b) are centered at the mean prediction from the center of the initial state distribution (if unclear, consider Figure 4.1 with $p(\mathbf{x}_0)$ as a point mass, then $p(\mathbf{x}_1)$ depicts the location of the box). This process is visualized in Figure 4.3. The axes on the boxes (Figs. 4.4a, 4.4b) are slightly different, because when θ is changed, the predicted location $p(\mathbf{x}_1;\theta)$ changes. The side lengths correspond to 4 standard deviations of the Gaussian distributions $p(\mathbf{x}_1;\theta)$. The velocities were kept fixed at the mean values. The random number seed was kept fixed. Note that because the random number seed is fixed, the value estimator $\hat{V}$ is the same as the remaining return G. This definition differs from the typical value function, which averages the return over an infinite amount of particles. The figures were created by predicting the trajectory at each point for 4 particles with

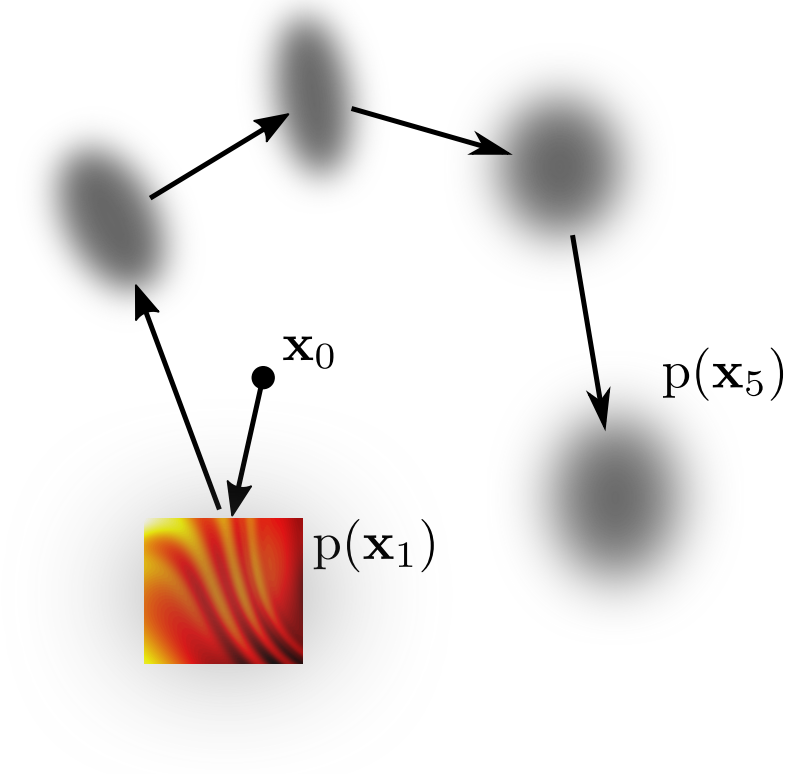
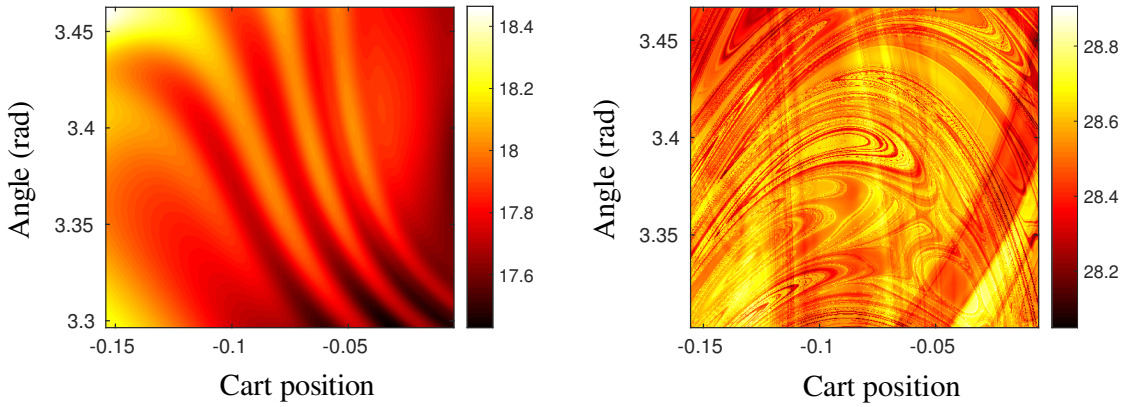


Figure 4.3: I plot the "value function" at a box centered at $p(\mathbf{x}_1)$.



(a) Value at $\mathrm{p}(\mathbf{x}_1)$ and $\Delta\theta = 0$ **(b)** Value at $\mathrm{p}(\mathbf{x}_1)$ and $\Delta\theta = 1.5$

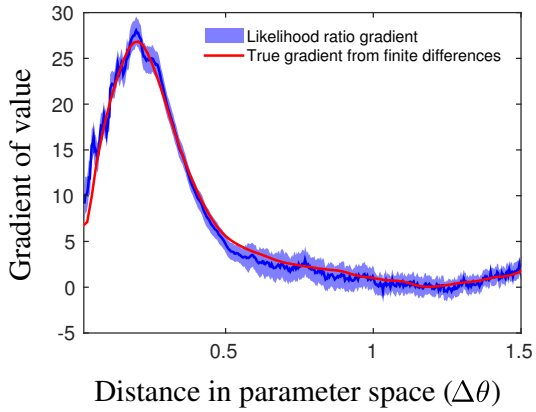
Figure 4.4: Value estimator landscapes corresponding to $\Delta\theta = 0$ and $\Delta\theta = 1.5$ in Figure 4.2.

different fixed seeds, then averaging the costs of the trajectories. I chose to predict 4 particles after trying 1 particle, for which the value appeared to include a step-like part, but was otherwise less interesting than the current figure. As the average value of the 4 particles is erratic, at least one of the 4 particles must have a highly erratic value estimator in the shown region.

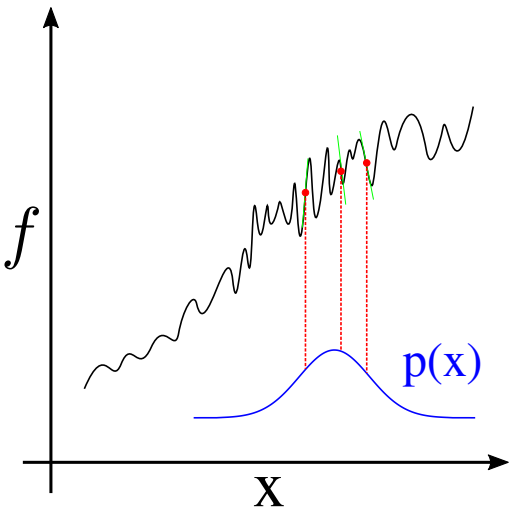
The results in Figures 4.4a and 4.4b explain the reason for the explosion of the variance when using RP gradients (Fig. 4.2b). Figure 4.4a corresponds to the leftmost parameter setting ($\Delta\theta = 0$); Figure 4.4b corresponds to the rightmost parameter setting ($\Delta\theta = 1.5$). As I had fixed the random number seed, RP computes the exact derivative $\frac{\mathrm{d}\hat{V}_t(\mathbf{x}_t)}{\mathrm{d}\mathbf{x}_t}$. It samples points inside the box, computes the gradient $\frac{\partial \hat{V}_t}{\partial \theta} = \frac{\partial \hat{V}_t}{\partial \mathbf{x}_t}\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}\theta}$ and averages the samples together.[1] In Figure 4.4b finding the gradient of the expectation by differentiating $\hat{V}_t$, and averaging is completely hopeless. Such a fractal input-output pattern is common in chaotic systems (Alligood et al., 1996), so we can expect that using RP gradients for such a system will not work. One may thus think that there is a fundamental issue, which would render solving such a problem impossible; however, the value averaged across 1000 particles is not the true objective—that would require averaging an infinite number of particles. When averaging an infinite amount of particles, is there still "noise" at the right-hand side of Figure 4.2b, or does the function become smooth? If the true objective is smooth, may there be some other gradient estimator, which does not suffer from the same problem as RP? Yes, there is.

Likelihood ratio gradient estimators are robust to chaos: One can see in Figure 4.5 that LR is able to estimate the gradient without the ill behavior observed for RP. RP estimates the gradient by differentiating the landscape. In contrast, LR (Fig. 4.12b) only uses the value $\hat{V}$, not its derivative, and does not suffer from this problem. To provide more evidence that the true objective is indeed smooth, I esti-

[1] Note that the same evaluation of the value gradient has to be performed at subsequent time-steps, and in practice the sum is evaluated simultaneously using backpropagation, but we ignore this for the purpose of the explanation.



(a) Likelihood ratio gradient.

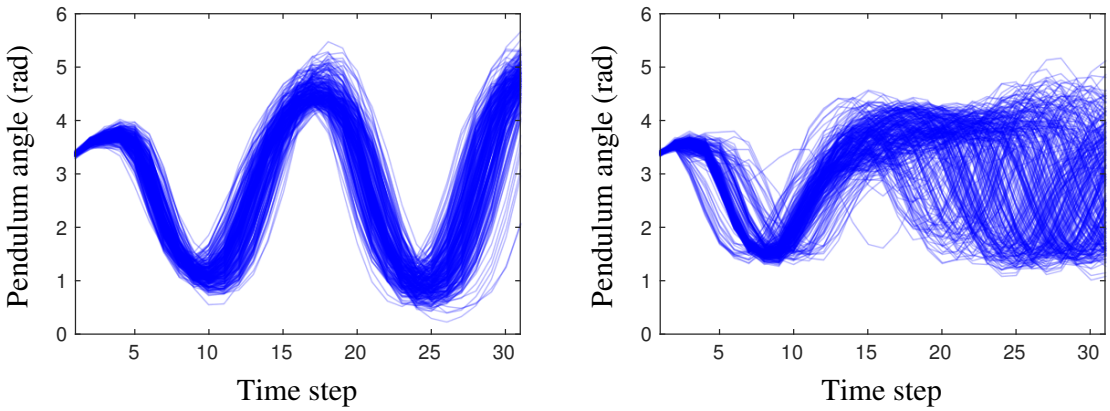


(b) The Gaussian acts as low-pass filter.

Figure 4.5: Likelihood ratio gradients do not use the gradient of the objective function to estimate the gradient of the expectation, and are thus robust to inifinitesimal "noise" on the objective.

mated the magnitude of the gradient from finite differences of the value in Figure 4.2a using a sufficiently large perturbation in θ, such that the "noise" is ignored. The fact that two separate approaches agree—one which varies the policy parameters θ, and another which keeps θ fixed, but estimates the gradient from the trajectories—provides convincing evidence that the true objective is smooth.

Why was LR able to estimate a gradient, even though the objective function appears to be extremely ill-behaved? The reason is illustrated in Figure 4.5b. Even though, the landscape in Figure 4.4b is erratic, averaging over a Gaussian distribution acts as a low-pass filter that removes the high-frequency components. Thus, the stochasticity is the reason why a sensible gradient could be estimated at all. If the dynamics were deterministic, such fractal input-output patterns would still occur, and there would be nothing that one could do about it. The reason that RP did not work, is that it tries to estimate the gradient of the smooth expected landscape by differentiating an ensemble of erratic non-smooth landscapes and averaging these together. LR on the other hand, does not use the derivative of these non-smooth value estimator landscapes, and is robust to chaos. It may seem a bit disappointing that RP does not work well in some situations, because typically it performs better than LR in many other tasks. In Section 4.4 I will show how the best of both LR and RP can be combined in the total propagation algorithm. But before that, I will examine the issue with chaos a bit more, and also discuss an alternative solution of resampling the particles at each time step.



(a) $\Delta\theta = 0$ **(b)** $\Delta\theta = 1.5$

(c) $\Delta\theta = 0$ **(d)** $\Delta\theta = 1.5$

Figure 4.6: Bifurcations and a chaos-like mixing of trajectories leads to poor gradients computed using backpropagation at $\Delta\theta = 1.5$ in Figure 4.2b.

4.2.2 A trajectory distribution view of chaotic dynamics

In the previous section, I explained the curse of chaos in terms of the value landscape in Figure 4.4b. In Figure 4.6, I show what the predicted trajectory distributions of the angle and angular velocity of the cart-pole look like for $\Delta\theta = 0$ (the well-behaved case) and $\Delta\theta = 1.5$ (the chaotic case). Due to the chaotic dynamics, there is a mixing of the trajectories, and the derivative of any individual trajectory does not provide information about the derivative of the whole distribution. Notice that if one compares individual trajectories between the two cases, the trajectories do not look that different, but the distributions have clearly different behavior. Such a mixing of trajectories is a hallmark of chaotic systems (Alligood et al., 1996) and provides another way to understand the phenomenon. Perturbing the parameters a little does not change the "ball" of trajectories much; however, the individual trajectories change a lot by being "remixed" within the distribution. Thus differentiating an individual trajectory does not provide meaningful information about how the whole "ball" will change.

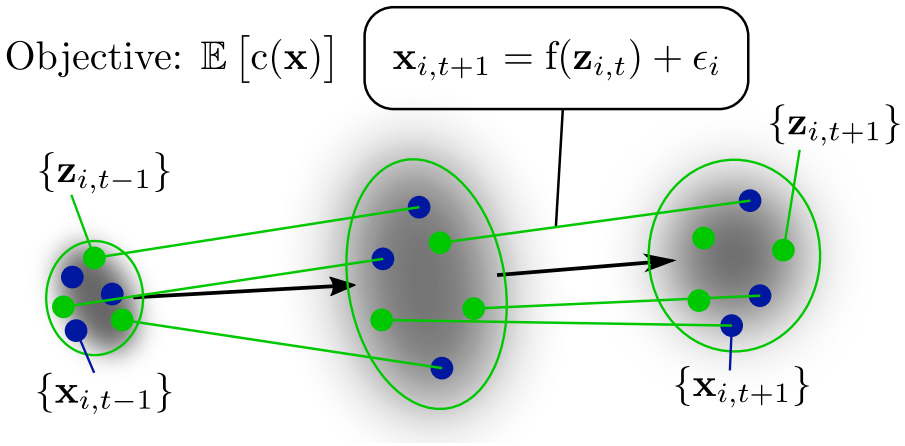
4.3 Resampling-based trajectory prediction

4.3.1 Resampling from a Gaussian

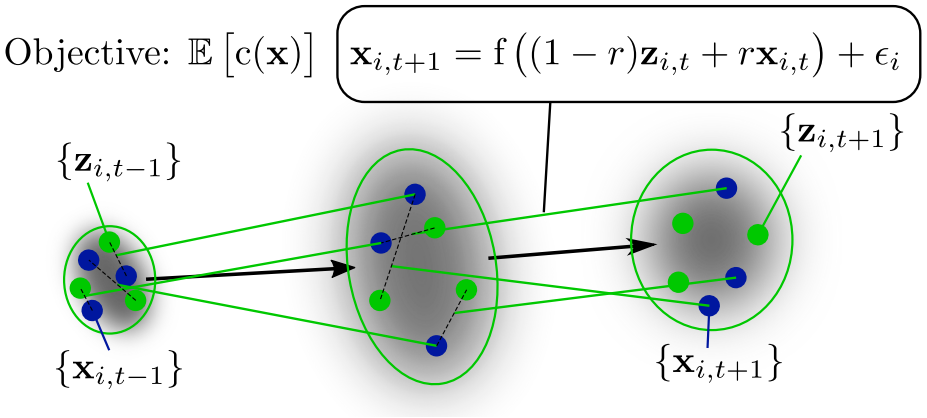
Originally, McHutchon (2014) had unsuccessfully attempted particle-based methods in PILCO, and suggested trying to replicate PILCO's Gaussian moment-matching effect by fitting a Gaussian on the particles and resampling to enforce unimodal trajectories. This was later tested in a Deep PILCO article (Gal et al., 2016), which found that resampling indeed improved the learning performance. I found that rather than the unimodality a more important effect of resampling is that the gradients are stabilized (compare Figure 4.2b to Figure 4.8a). This effect is most likely due to an averaging out of the value landscapes, i.e., it smooths out the high-frequency components in Figure 4.4b.

The resampling method works by fitting a Gaussian on the particles at each time step, i.e., $\hat{\mu} = \sum_{i=1}^{P} \mathbf{x}_i/P$ and $\hat{\Sigma} = \sum_{i=1}^{P}(\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T/(P-1)$. The particles are resampled from the fitted distribution $\mathbf{z}_i \sim \hat{\mu} + L\epsilon_i \mid \epsilon_i \sim \mathcal{N}(0, I)$, where L is the Cholesky factor of $\hat{\Sigma}$. The gradient is computed with standard reparameterization, but see (Murray, 2016) for how to compute $\frac{\mathrm{d}L}{\mathrm{d}\hat{\Sigma}}$. An illustration of this resampling method is in Figure 4.7a.

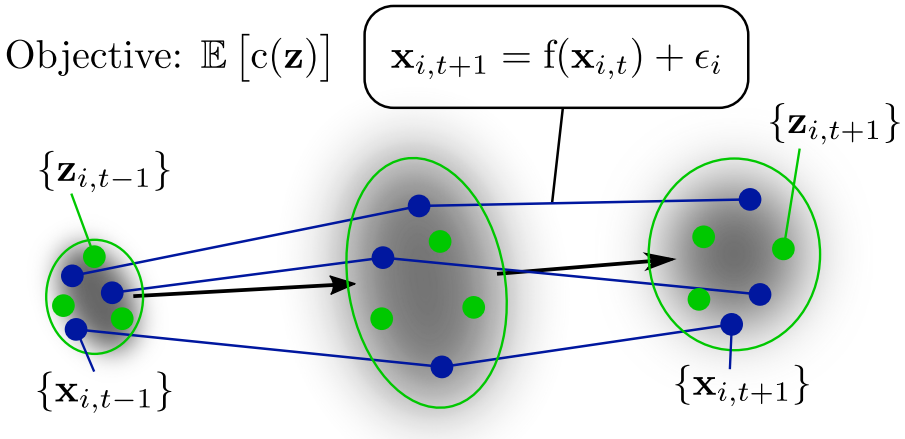
I further considered a method to smoothly interpolate between fully resampling, and keeping the particles on their original trajectory. The method works by sampling from the fitted Gaussian, but moving the original samples only a portion of the distance toward the resampled particles, i.e., for each each particle $\mathbf{z}'_i = (1-r)\mathbf{z}_i + r\mathbf{x}_i$, where $r \in [0, 1]$ is a ratio. An illustration of this resampling method is in Figure 4.7b. I have contrasted both resampling techniques to the Gaussian shaping gradient (Sec. 3.3.2), which is illustrated in Figure 4.7c. Unlike the resmapling methods, the Gaussian shaping gradient does not modify the trajectory distribution, but only changes the rewards by computing the reward over an expectation w.r.t. a fitted Gaussian distribution. The plots in Figure 4.8 show that the gradients in the ratio resampling method gradually transition from stable to unstable when one deviates from fully resampling.



(a) Full resampling

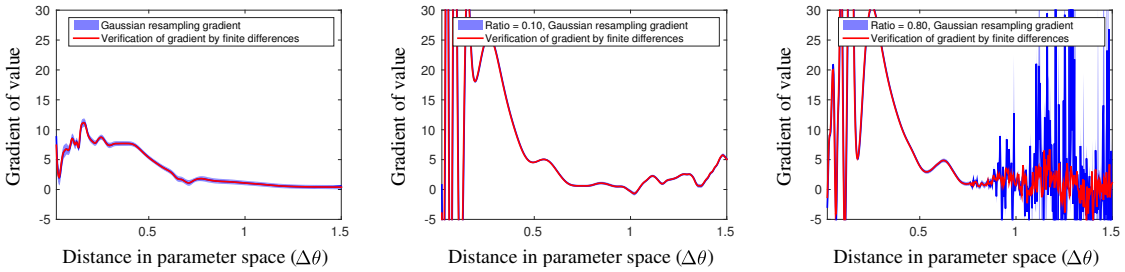


(b) Ratio resampling



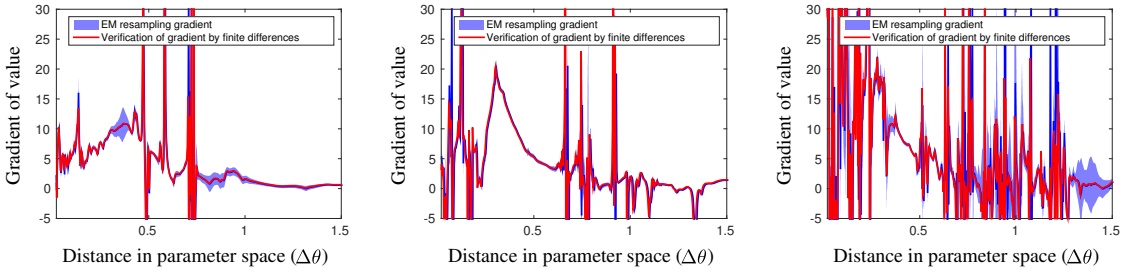
(c) Gaussian shaping

Figure 4.7: Illustration of Gaussian resampling with comparison to Gaussian shaping.



(a) Full resampling **(b)** Ratio = 0.1 **(c)** Ratio = 0.8

Figure 4.8: The gradients gradually transition from unstable to stable when the particles are moved closer towards being completely resampled from a Gaussian distribution.



(a) $K = 2, M = 2$ **(b)** $K = 2, M = 5$ **(c)** $K = 5, M = 2$

Figure 4.9: Increasing the # of mixture components K or EM iterations M leads to worse gradients.

4.3.2 Resampling from a mixture of Gaussians

Section 4.3.1 explained that resampling can smooth out the gradients. An interesting question is then whether resampling not from a unimodal Gaussian, but from a mixture distribution could allow stabilizing the gradients, while still allowing for multimodal trajectory distributions. I propose to fit mixture distributions using the EM algorithm, then backpropagate through the algorithm to obtain the gradient. Algorithm 4 explains the method. The results are in Figure 4.9. One can see that that the gradients were not stabilized. The result is not entirely conclusive, as the gradient behaves poorly in the full range of $\Delta\theta$, not just in the unstable region in Figure 4.2b; however, the gradients become worse, as the method deviates further from a unimodal Gaussian, and it does not appear straightforward to achieve good performance with a multimodal resampling method.

Algorithm 4 Particle-based trajectory predictions while resampling from a mixture of Gaussians fitted by the Expectation-Maximization Algorithm (EM)

Input: policy π with parameters θ, episode length T, initial Gaussian state distribution $p(\mathbf{x}_0)$, cost function $c(\mathbf{x})$, learned dynamics model $\hat{f}$, number of particles P, number of mixture components K, number of EM iterations M.

Initialize: Set initial mixture component weights equally $W_k = {}^1/_K$, set initial mixture distributions to the initial distribution $g_k = p(\mathbf{x}_0)$ for each k, sample ${}^P/_K$ initial particles separately from each mixture component $\{\mathbf{z}_{i,k,0}\}_{i=1}^{P} \sim g_k(\mathbf{z}_0)$.

for $t = 0$ **to** $T - 1$ **do**

1. **Predict next timestep:**

for each particle i **do**

Compute controls: $\mathbf{u}_{i,t} = \pi(\mathbf{z}_{i,t}; \theta)$

Predict next state distribution: $\mathcal{N}(\mathbf{x}_{i,t+1}; \mathbf{m}_{i,t+1}, \mathbf{S}_{i,t+1}) = \hat{f}(\mathbf{z}_{i,t}, \mathbf{u}_{i,t})$

Set particle weight: $w_{i,t+1} = W_{k,t}$ ▷ Based on which g_k, $\mathbf{z}_{i,t}$ was sampled from

end for

2. **Compute new initialization:**

for each mixture component k **do**

$\mu_k = \frac{1}{P/K} \sum_{i=1}^{P/K} \mathbf{m}_{i,k,t+1}$ ▷ The k index for $\mathbf{m}_{t+1}$ means the particle $\mathbf{z}_t$ came g_k

$\Sigma_k = \frac{1}{P/K-1} \sum_{i=1}^{P/K} (\mathbf{m}_{i,k,t+1} - \mu_k)(\mathbf{m}_{i,k,t+1} - \mu_k)^T + \frac{1}{P/K} \sum_{i=1}^{P/K} \mathbf{S}_{i,k,t+1}$

end for

3. **Run EM with weighted samples to fit the Gaussian components:**

$\{\mu_k, \Sigma_k, W_k\}_{k=1}^{K} = \mathrm{EM}\left(M, (\mathbf{m}_{t+1}, \mathbf{S}_{t+1}, \mathbf{w}_{t+1}), \{\mu_k, \Sigma_k, W_k\}_{k=1}^{K}\right)$ ▷ The update equations in EM are weighted versions of the equations in step 2. $\mathbf{S}$ is ignored when computing the responsibilities for the samples, but is used when updating the covariance of the mixture component.

4. **Resample particles and compute cost:**

for each mixture component k **do**

Sample ${}^P/_K$ new particles: $\mathbf{z}_{i,k,t+1} \sim g_k(\mathbf{z}_{i,k,t+1}; \mu_k, \Sigma_k)$

end for

Average the cost: $c_{t+1} = \frac{1}{P} \sum_{i=1}^{P} c(\mathbf{z}_{i,t+1})$

end for

Gradient computation: $\frac{\mathrm{d}}{\mathrm{d}\theta} \left(\sum_{t=1}^{T} \mathbb{E}\left[c(\mathbf{z}_{t+1})\right] \right)$ is stochastically approximated from the particles. Each computation can be differentiated, and the full gradient can be obtained by backpropagation.

4.3.3 Why resampling based methods are undesirable

Even if resampling from a multimodal distribution were to work, I believe it is not ideal, because it destroys the temporal dependence in the particles, which is undesirable for the reasons listed below.

- It is difficult to model sampling dependencies, i.e., if the uncertainty is caused by a lack of knowledge about the dynamics model, then the sampling should be correlated, because sampling a prediction restricts the probability space of the underlying dynamics function. For example, if the dynamics function is a GP, and one predicts a pair of points $[x_1, x_2]$, then these two samples have to covary, because a GP places a distribution on the underlying function. Therefore, the history of sampled points in a trajectory matters in determining what the next prediction should be. But resampling methods remove this temporal dependence, and incorporating a distribution over dynamics models properly becomes difficult.

- If the controller depends on past history, e.g., a recurrent neural network, resampling makes it more difficult to model such a dependency.

- Parallel computation becomes more difficult, as it is necessary to exchange information between the particles to make predictions.

- The trajectory distribution is modified, and does not correspond to the true trajectory, even if the model is perfect. This can lead to learning controllers with lower performance, or even worse, the task may even become impossible if the approximation is too conservative, e.g., see Figure 1.4.

4.4 Total propagation algorithm

In Section 4.2 we saw that in chaotic situations, the reparameterization gradient is completely hopeless. On the other hand, the likelihood ratio gradient was robust to such issues. However, the likelihood ratio gradient does not scale as well with the dimensionality of the problem (Nesterov and Spokoiny, 2017). Indeed, backpropagating gradients similarly as RP does has been central in the deep learning revolution (LeCun et al., 2015; Schmidhuber, 2015a), and reinforcement learning based approaches to train neural networks for supervised machine learning does not work as well. One of the main motivations for model-based reinforcement by differentiating the predictions was that this may be able to scale to much higher dimensions, as it relies on backpropagation. It is thus disappointing that a reinforcement learning gradient estimator may be the best approach available. In this section, I introduce my *total propagation algorithm*, which is an algorithm for combining both the likelihood ratio and reparameterization based approaches into a single estimator obtaining the robustness of the LR method, and aiming to achieve the efficiency of the RP method.

RP/LR weighted average: The bulk of the computation in the model-based reinforcement learning algorithm with particles is spent on the $\mathrm{d}p(\mathbf{x}_{t+1}|\mathbf{x_t};\theta)/\mathrm{d}\theta$ terms.

These terms are needed for both LR and RP gradients, so there is no penalty to combining both estimators. A well known statistics result states that for independent estimators, an optimal weighted average estimate is achieved if the weights are proportional to the inverse variance, i.e., $\mu = \mu_{LR}k_{LR} + \mu_{RP}k_{RP}$, where $k_{LR} = \hat{\sigma}_{LR}^{-2}/(\hat{\sigma}_{LR}^{-2} + \hat{\sigma}_{RP}^{-2})$ and $k_{RP} = 1 - k_{LR}$. This technique is known as inverse-variance weighting (Fleiss, 1993).

A naïve combination scheme would compute the gradient separately for the whole trajectory for both estimators, then combine them; however, this approach neglects the opportunity to use reparameterization gradients through shorter sections of the trajectory to obtain better gradient estimates. My *total propagation algorithm* (TP) goes beyond the naïve method. TP uses a single backward pass to compute a union over all possible RP depths, automatically giving greater weight to estimators with lower variance.

A description is provided in Algorithm 5. At each backward step, it evaluates the gradient w.r.t. the policy parameters using both the LR and RP methods. It evaluates a ratio based on the variances in *policy parameter space*—this variance is proportional to the variance of the policy gradient estimator. The gradients are combined, and a "best" estimate in *distribution parameter space* is passed to the previous time step. In the algorithm, the $\mathbb{V}$ operator takes the sample variance of gradient estimates from different particles; however, other variance estimation schemes could also be considered: one could estimate variances from moving averages of the magnitude of the gradient, use a different statistical estimator for the variance, use only a subset of policy parameters, etc. The algorithm can also be used to combine other kinds of gradient estimators, for example, I have explained how to combine total propagation and Gaussian shaping gradients in Algorithm 6. The algorithm is not limited to RL problems, but is applicable to general stochastic computation graphs (Schulman et al., 2015), and could be used for training probabilistic models, stochastic neural networks, etc.

To further illustrate the algorithm, I have created diagrams for both backpropagation (Fig. 4.10) and total propagation (Fig. 4.11) on a simple neural network with 3 hidden layers, to contrast the two algorithms. As can be seen, they are quite similar, but total propagation performs multiple estimates of the gradient using different gradient estimator, and sends the combined estimator backwards.

Algorithm 5 Total Propagation Algorithm
(used in PIPPS for evaluating the gradient)

This algorithm provides an efficient method to fuse LR and RP gradients by combining ideas from filtering and backpropagation. The algorithm is explained here with reference to my policy search framework.
Forward pass: Compute a set of particle trajectories.
Backward pass:

Initialize: $\frac{\mathrm{d}G_{T+1}}{\mathrm{d}\zeta_{T+1}} = \mathbf{0}$, $\frac{\mathrm{d}J}{\mathrm{d}\theta} = \mathbf{0}$, $G_{T+1} = 0$ where ζ are the distribution parameters, e.g. μ and σ.
for $t = T$ **to** 1 **do**
  for each particle i **do**
    $G_{i,t} = G_{i,t+1} + c_{i,t}$, where c_t is the cost at time t.
    $\frac{\mathrm{d}\zeta_{i,t+1}}{\mathrm{d}\mathbf{x}_{i,t}} = \frac{\partial\zeta_{i,t+1}}{\partial\mathbf{x}_{i,t}} + \frac{\mathrm{d}\zeta_{i,t+1}}{\mathrm{d}\mathbf{u}_{i,t}}\frac{\mathrm{d}\mathbf{u}_{i,t}}{\mathrm{d}\mathbf{x}_{i,t}}$
    $\frac{\mathrm{d}G^{RP}_{i,t}}{\mathrm{d}\zeta_{i,t}} = (\frac{\mathrm{d}G_{i,t+1}}{\mathrm{d}\zeta_{i,t+1}}\frac{\mathrm{d}\zeta_{i,t+1}}{\mathrm{d}\mathbf{x}_{i,t}} + \frac{\mathrm{d}c_{i,t}}{\mathrm{d}\mathbf{x}_{i,t}})\frac{\mathrm{d}\mathbf{x}_{i,t}}{\mathrm{d}\zeta_{i,t}}$
    $\frac{\mathrm{d}G^{LR}_{i,t}}{\mathrm{d}\zeta_{i,t}} = (G_{i,t} - b_{i,t})\frac{\mathrm{d}\log p(\mathbf{x}_{i,t})}{\mathrm{d}\zeta_{i,t}}$
    $\frac{\mathrm{d}G^{RP}_{i,t}}{\mathrm{d}\theta} = \frac{\mathrm{d}G^{RP}_{i,t}}{\mathrm{d}\zeta_{i,t}}\frac{\mathrm{d}\zeta_{i,t}}{\mathrm{d}\mathbf{u}_{i,t-1}}\frac{\mathrm{d}\mathbf{u}_{i,t-1}}{\mathrm{d}\theta}$
    $\frac{\mathrm{d}G^{LR}_{i,t}}{\mathrm{d}\theta} = \frac{\mathrm{d}G^{LR}_{i,t}}{\mathrm{d}\zeta_{i,t}}\frac{\mathrm{d}\zeta_{i,t}}{\mathrm{d}\mathbf{u}_{i,t-1}}\frac{\mathrm{d}\mathbf{u}_{i,t-1}}{\mathrm{d}\theta}$
  end for
  $\sigma^2_{RP} = \mathrm{trace}(\mathbb{V}\left[\frac{\mathrm{d}G^{RP}_{i,t}}{\mathrm{d}\theta}\right])$, $\sigma^2_{LR} = \mathrm{trace}(\mathbb{V}\left[\frac{\mathrm{d}G^{LR}_{i,t}}{\mathrm{d}\theta}\right])$
  $k_{LR} = 1/\left(1 + \frac{\sigma^2_{LR}}{\sigma^2_{RP}}\right)$
  $\frac{\mathrm{d}J}{\mathrm{d}\theta} = \frac{\mathrm{d}J}{\mathrm{d}\theta} + k_{LR}\frac{1}{P}\sum_i^P \frac{\mathrm{d}G^{LR}_{i,t}}{\mathrm{d}\theta} + (1 - k_{LR})\frac{1}{P}\sum_i^P \frac{\mathrm{d}G^{RP}_{i,t}}{\mathrm{d}\theta}$
  for each particle i **do**
    $\frac{\mathrm{d}G_{i,t}}{\mathrm{d}\zeta_{i,t}} = k_{LR}\frac{\mathrm{d}G^{LR}_{i,t}}{\mathrm{d}\zeta_{i,t}} + (1 - k_{LR})\frac{\mathrm{d}G^{RP}_{i,t}}{\mathrm{d}\zeta_{i,t}}$
  end for
end for

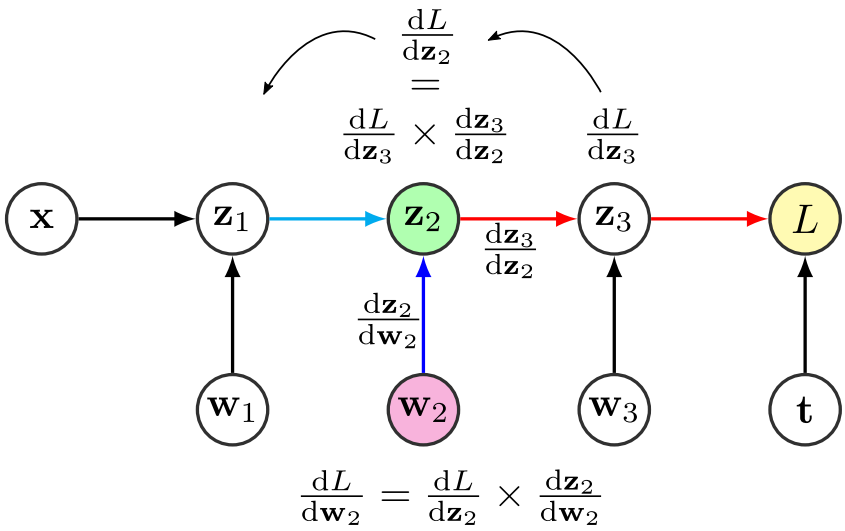


Figure 4.10: Illustration of the backpropagation algorithm, which is used in all neural network applications in machine learning, as well as many other applications. The total propagation algorithm (Fig. 4.11) modifies this procedure by obtaining multiple estimates of $\frac{\mathrm{d}L}{\mathrm{d}z_2}$ using different gradient estimation techniques (e.g. reparameterization and likelihood ratio methods), combining these estimates into a single gradient estimator, and passing the combined estimator backwards in the computational graph.

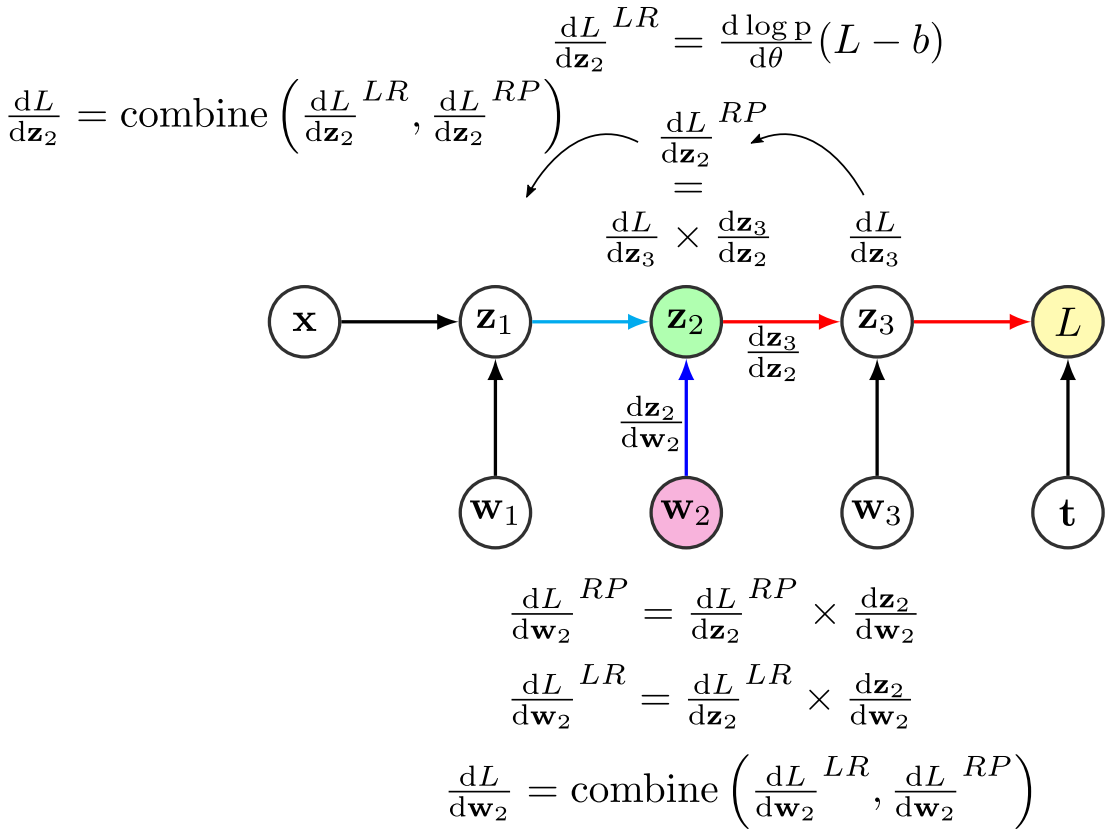
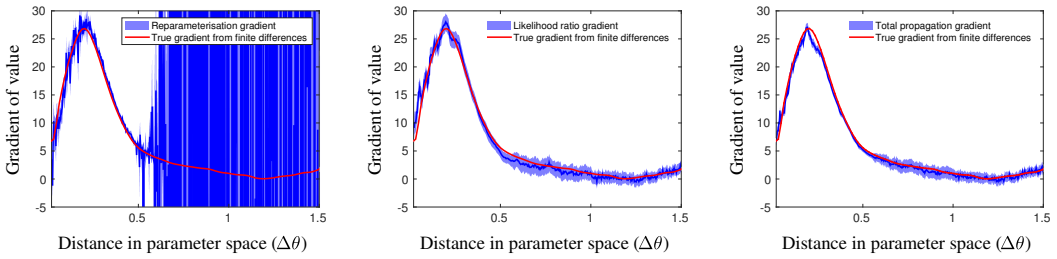


Figure 4.11: Illustration of the total propagation algorithm when the gradient estimation is performed by combining the likelihood ratio and reparameterization gradient estimators into a single gradient estimator.

Algorithm 6 Gaussian shaping gradient with total propagation

Gaussian shaping gradient for model-based policy search while combining both LR and RP variants using total propagation.
Forward pass: Sample a set of particle trajectories.
Backward pass:

Initialize: $\frac{\mathrm{d}G_{T+1}}{\mathrm{d}\zeta_{T+1}} = \mathbf{0}$, $\frac{\mathrm{d}J}{\mathrm{d}\theta} = \mathbf{0}$, $G_{T+1} = 0$ ▷ ζ are the distribution parameters, e.g. all of the μ and σ for each particle
for $t = T$ **to** 1 **do**
$\mu_t = \mathbb{E}[\mathbf{x}_t]$; $\Sigma_t = \mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right] - \mu_t\mu_t^T$ ▷ Estimate the marginal distribution as a Gaussian
Compute: $\frac{\mathrm{d}\mathbb{E}[c_t]}{\mathrm{d}\mu_t}$ and $\frac{\mathrm{d}\mathbb{E}[c_t]}{\mathrm{d}\Sigma_t}$, e.g. by sampling from this Gaussian, and using the RP gradient
for each particle i **do**
$\mathbf{m}_{i,t} = \mathbf{x}_{i,t} - \mu_t$; $\mathbf{v}_{i,t} = \mathrm{vec}\left(\mathbf{x}_{i,t}\mathbf{x}_{i,t}^T - \mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]\right)$; $\mathbf{w}_{i,t} = \mathrm{vec}\left(\mathbf{m}_{i,t}\mu_t^T\right)$ ▷ $\mathrm{vec}(\cdot)$ is a vectorization operator which stacks the elements in a matrix/tensor into a column vector
$g_{i,t} = \frac{\mathrm{d}\mathbb{E}[c_t]}{\mathrm{d}\mu_t}\mathbf{m}_{i,t} + \frac{\mathrm{d}\mathbb{E}[c_t]}{\mathrm{d}\Sigma_t}(\mathbf{v}_{i,t} - 2\mathbf{w}_{i,t})$ ▷ g is a scalar replacing the usual cost/reward
$G_{i,t} = G_{i,t+1} + g_{i,t}$ ▷ G is the return (the cost of the remaining trajectory)
$\frac{\mathrm{d}\mathbb{E}[c_t]}{\mathrm{d}\mathbf{x}_{i,t}} = \frac{\mathrm{d}\mathbb{E}[c_t]}{\mathrm{d}\mu_t}\frac{\mathrm{d}\mu_t}{\mathrm{d}\mathbf{x_{i,t}}} + \frac{\mathrm{d}\mathbb{E}[c_t]}{\mathrm{d}\Sigma_t}\frac{\mathrm{d}\Sigma_t}{\mathrm{d}\mathbf{x_{i,t}}}$ ▷ Direct derivative of expected cost for the RP gradient
$\frac{\mathrm{d}\zeta_{i,t+1}}{\mathrm{d}\mathbf{x}_{i,t}} = \frac{\partial\zeta_{i,t+1}}{\partial\mathbf{x}_{i,t}} + \frac{\mathrm{d}\zeta_{i,t+1}}{\mathrm{d}\mathbf{u}_{i,t}}\frac{\mathrm{d}\mathbf{u}_{i,t}}{\mathrm{d}\mathbf{x}_{i,t}}$
$\frac{\mathrm{d}G_{i,t}^{RP}}{\mathrm{d}\zeta_{i,t}} = (\frac{\mathrm{d}G_{i,t+1}}{\mathrm{d}\zeta_{i,t+1}}\frac{\mathrm{d}\zeta_{i,t+1}}{\mathrm{d}\mathbf{x}_{i,t}} + \frac{\mathrm{d}\mathbb{E}[c_t]}{\mathrm{d}\mathbf{x}_{i,t}})\frac{\mathrm{d}\mathbf{x}_{i,t}}{\mathrm{d}\zeta_{i,t}}$
$\frac{\mathrm{d}G_{i,t}^{LR}}{\mathrm{d}\zeta_{i,t}} = G_{i,t}\frac{\mathrm{d}\log\mathrm{p}(\mathbf{x}_{i,t})}{\mathrm{d}\zeta_{i,t}}$ ▷ One could also further subtract a baseline from G
$\frac{\mathrm{d}G_{i,t}^{RP}}{\mathrm{d}\theta} = \frac{\mathrm{d}G_{i,t}^{RP}}{\mathrm{d}\zeta_{i,t}}\frac{\mathrm{d}\zeta_{i,t}}{\mathrm{d}\mathbf{u}_{i,t-1}}\frac{\mathrm{d}\mathbf{u}_{i,t-1}}{\mathrm{d}\theta}$
$\frac{\mathrm{d}G_{i,t}^{LR}}{\mathrm{d}\theta} = \frac{\mathrm{d}G_{i,t}^{LR}}{\mathrm{d}\zeta_{i,t}}\frac{\mathrm{d}\zeta_{i,t}}{\mathrm{d}\mathbf{u}_{i,t-1}}\frac{\mathrm{d}\mathbf{u}_{i,t-1}}{\mathrm{d}\theta}$
end for
$\sigma_{RP}^2 = \mathrm{trace}(\mathbb{V}\left[\frac{\mathrm{d}G_{i,t}^{RP}}{\mathrm{d}\theta}\right])$; $\sigma_{LR}^2 = \mathrm{trace}(\mathbb{V}\left[\frac{\mathrm{d}G_{i,t}^{LR}}{\mathrm{d}\theta}\right])$ ▷ The sample variance of the particles
$k_{LR} = 1/\left(1 + \frac{\sigma_{LR}^2}{\sigma_{RP}^2}\right)$ ▷ Weight to combine LR and RP estimators
$\frac{\mathrm{d}J}{\mathrm{d}\theta} = \frac{\mathrm{d}J}{\mathrm{d}\theta} + k_{LR}\frac{1}{P}\sum_i^P\frac{\mathrm{d}G_{i,t}^{LR}}{\mathrm{d}\theta} + (1 - k_{LR})\frac{1}{P}\sum_i^P\frac{\mathrm{d}G_{i,t}^{RP}}{\mathrm{d}\theta}$ ▷ Combine LR and RP in θ space
for each particle i **do**
$\frac{\mathrm{d}G_{i,t}}{\mathrm{d}\zeta_{i,t}} = k_{LR}\frac{\mathrm{d}G_{i,t}^{LR}}{\mathrm{d}\zeta_{i,t}} + (1 - k_{LR})\frac{\mathrm{d}G_{i,t}^{RP}}{\mathrm{d}\zeta_{i,t}}$ ▷ Combine LR and RP in state space
end for
end for



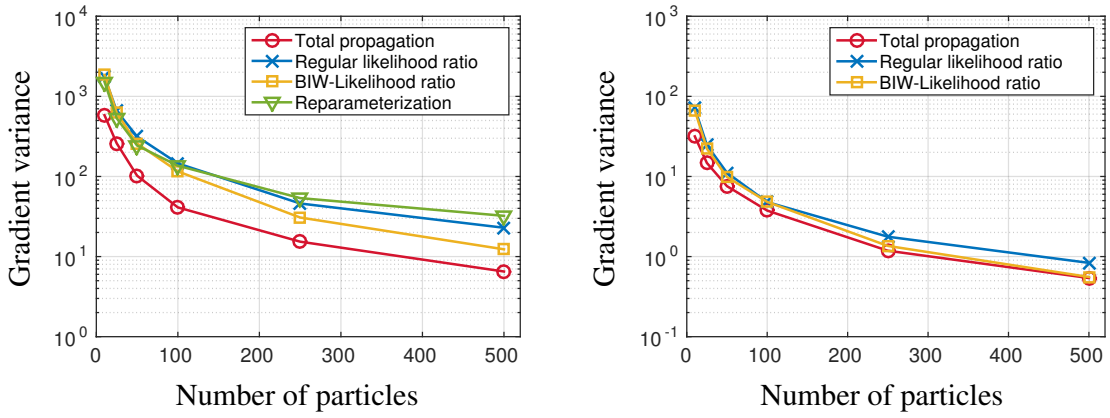
(a) Reparameterization **(b)** Likelihood ratio **(c)** Total propagation

Figure 4.12: Comparison of LR, RP and TP gradient estimators. RP is accurate on the left side, LR is robust on the right side, TP combines the best of both.

4.4.1 Gradient variance evaluation

I performed a simple comparison of the gradient estimators. First, in Figure 4.12 I plotted the gradient computed using total propagation (TP) in the perturbation range $\Delta\theta \in [0, 1.5]$ as I had previously done for RP in Figure 4.2. I plotted LR and RP separately to compare to TP. As can be seen, TP looks qualitatively the best—it is accurate in the region where RP was accurate, but is also robust to the issue with chaos. I investigated further how much the gradient accuracy is improved: In Figure 4.13, I plot how the variance of the gradient estimators at $\Delta\theta = 0$ and $\Delta\theta = 1.5$ depends on the number of particles P. The variance was computed by repeatedly sampling the estimator for a large number of times and calculating the variance from the set of evaluations. I compare RP, total propagation (TP) as well as LR gradients both with and without batch importance weighting (BIW) to show that my importance sampling scheme reduces the variance. I used the importance sampled baseline; in practice the regular LR gradient would use a simpler baseline, and have even higher variance. The RP gradient is omitted from Figure 4.13b, because the variance was between 10^8-10^{15}. The TP gradient combined the BIW-LR and RP gradients.

The results confirm that BIW significantly reduces the variance. Moreover, my total propagation algorithm was the best. Importantly, in Figure 4.13b, even though the variance of the RP gradient for the full trajectory is over 10^6 larger than the other estimators, TP utilizes shorter path-length RP gradients to obtain 10-50% reduction in variance for 250 particles and fewer. It is particularly remarkable that TP achieved a higher accuracy, than the sums of the accuracies of the individual estimators it was combining. In general, when combining two independent estimates of the same random variable, the best that one could hope to achieve is an accuracy that sums the individual accuracies; however, TP was able to utilize the graph structure of the computations, and hence outperformed the best possible naïve combination of the gradient estimators. In the next section, I show that such improved gradient estimators, also lead to improved learning performance, and allowed matching up to PILCO. Thus, the field may now be ready to attempt swapping out the models, and trying to scale such differentiable model-based RL algorithms to larger scale problems.



(a) $\Delta\theta = 0$ **(b)** $\Delta\theta = 1.5$

Figure 4.13: Computed variances corresponding to plots in Figure 4.12.

4.5 Learning experiments

I performed learning experiments to show that the new improved gradient estimators translate to better learning performance compared to the standard reparameterization gradient estimator, and that my method can match up to PILCO, while lifting the restrictions of PILCO, which have prevented attempting to scale up such an algorithm. I compare PILCO in episodic learning tasks to the following particle-based methods: reparameterization gradients (RP), RP with a fixed seed (RP_{FS}), Gaussian resampling (GR), GR with a fixed seed (GR_{FS}), model-based batch importance weighted likelihood ratio (LR), total propagation combining the BIW-LR and RP estimators (TP), Gaussian shaping gradients using only the likelihood ratio method (GLR), Gaussian shaping gradients combining the BIW-LR and RP variants using total propagation (GTP), and finally, the density estimation likelihood ratio gradient estimator (DEL). Moreover, I evaluate two variations of the particle predictions: 1. TP while ignoring model uncertainty, and adding only the noise at each time step ($\text{TP} - \sigma_f$). 2. TP and GTP with increased prediction noise ($\text{TP} + \sigma_n$): 100 and 25 times more noise for TP and GTP respectively.

I performed learning tasks from a recent PILCO paper (Deisenroth et al., 2015): cart-pole swing-up and balancing, and unicycle balancing (Fig. 4.14). Moreover, for the DEL estimator, I tried a simpler cart-pole balancing only task, without the swing-up. The DEL estimator is a fairly crude approach, and I am simply showing that even such a method can achieve non-trivial success rate. The simulation dynamics were set to be the same as in the original PILCO papers, and other aspects were also similar to the original PILCO, but modifications were made to the experimental setup to accommodate ease of implementation and explore different aspects of the algorithms.

4.5.1 Optimizers:

RMSprop-like stochastic gradient descent: The main algorithm I used was inspired by RMSprop (Tieleman and Hinton, 2012). RMSprop normalizes its stochastic

gradient descent (SGD) steps by utilizing a running average of the square of the gradients. In my case, since the batch sizes were large, I directly estimate the expectation of the square from the batch by $z = \mathbb{E}[g^2] = \mathbb{E}[g]^2 + \mathbb{V}[g]$, where g is the gradient. I use the variance of the mean, i.e., $\mathbb{V}[g]$ is the variance divided by the number of particles P. The gradient step becomes $g/\sqrt{z}$. I use momentum with the parameter γ. The full update equations become:

$$m \leftarrow \gamma m + g/\sqrt{\mathbb{E}[g]^2 + \mathbb{V}[g]}$$
$$\theta \leftarrow \theta - \alpha m$$

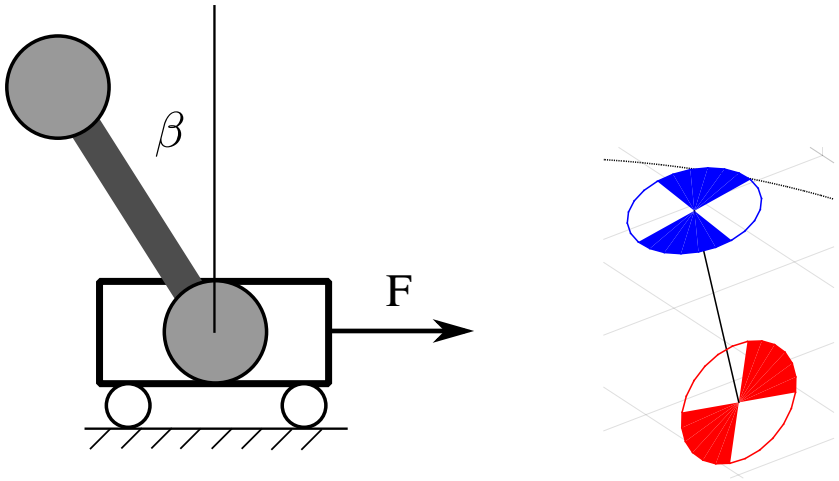
Deterministic optimizer: The random number seed can be fixed to turn a stochastic problem deterministic, also known as the PEGASUS trick (Ng and Jordan, 2000). With a fixed seed, the RP gradient is an exact gradient of the objective, and quasi-Newton optimizers, such as BFGS (Nocedal and Wright, 2006) can be used.

4.5.2 Task Descriptions

Cart-pole swing-up and balancing: This is a standard control theory benchmark problem. The task consists of pushing a cart back and forth, to swing an attached pendulum to an upright position, then keep it balanced. The state space was represented as $\mathbf{x} = [s, \beta, \dot{s}, \dot{\beta}]$, where s is the cart-position and β the pole angle. The base noise levels were $\sigma_s = 0.01$ m, $\sigma_\beta = 1$ deg, $\sigma_{\dot{s}} = 0.1$ m/s, $\sigma_{\dot{\beta}} = 10$ deg/s. The noise was modified in different experiments by a multiplier k: $\sigma_o^2 = k\sigma_{base}^2$. The original PILCO paper considered direct access to the true state. I set $k = 10^{-2}$ to obtain a similar setting, but also tested $k \in \{1, 4, 9, 16\}$. The policy $\tilde{\pi}$ was a radial basis function network (a sum of Gaussians) with 50 basis functions. I considered two cost functions (Fig. 4.15). Both costs were of the saturated cost type explained in Section 4.1. One was the same as in the original PILCO, with $\mathbf{x}$ including the sine and cosine, and depended on the distance between the tip of the pendulum to the position of the tip when the pendulum is balanced (*Tip Cost*). The other cost used the raw angle and had $Q = \mathrm{diag}([1, 1, 0, 0])$ (*Angle Cost*). This cost differs conceptually from the *Tip Cost*, because there is only one correct direction in which to swing up the pendulum. To test dependence on hyperparameter tuning: in the RP, *Angle Cost*, $k = 1$ case, I tested $\alpha \in \{10^{-3}, 10^{-4}\}$, which lie above and below the standard learning rate, and both of these yielded worse performance.

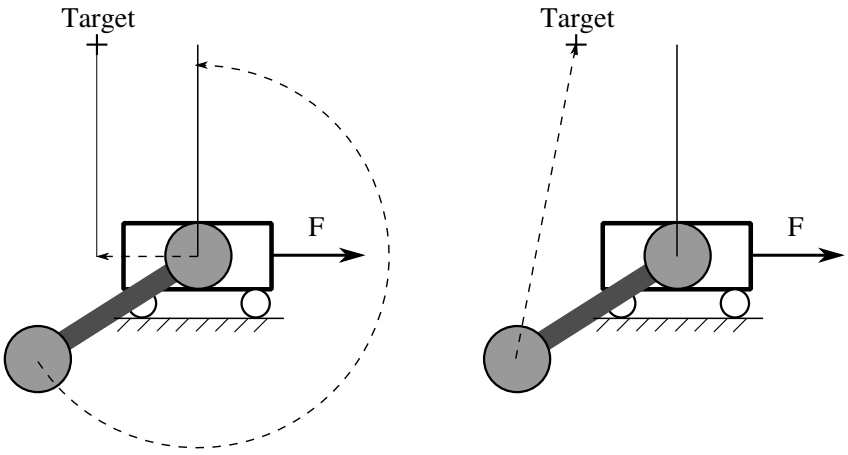
Cart-pole balancing with DEL estimator This task is much simpler—the pole starts upright and must be balanced. The experiment was devised to show that DEL is feasible and may be useful if further developed. The *Angle Cost* and the base noise level were used.

Unicycle balancing: See the work of Deisenroth et al. (2015) for a description. The task consists of balancing a unicycle robot, with state dimension $D = 12$, and control dimension $F = 2$. The robot can be controlled by applying torques to the wheel on the floor to control backward and forward movement, and to the flywheel at the top



(a) Cart-pole. $D = 4, F = 1$ **(b)** Unicycle. $D = 12, F = 2$

Figure 4.14: Illustrations of the systems used in my simulations experiments.



(a) Angle cost **(b)** Tip cost

Figure 4.15: Illustrations of the costs used in the cart-pole task.

of the robot to turn the robot and balance it. The noise was set to a low value. The controller $\tilde{\pi}$ is linear. I performed this experiment to show that my algorithm can also match PILCO in higher dimensional problems. The unicycle task is at about the upper limit in terms of dimensionality among tasks that PILCO can handle.

4.5.3 Experimental setup

The optimizer was run for 600 policy evaluations between each trial. The stochastic gradient descent learning rate, and momentum parameters were $\alpha = 5 \times 10^{-4}$ and $\gamma = 0.9$. The episode lengths were 3s for the cart-pole, and 2s for the unicycle. Note that for the unicycle task, 2s was not sufficient for the policy to generalize to long trials, but it still allowed comparing to PILCO. The control frequencies were 10Hz. The costs were of the type $1 - \exp(-(\mathbf{x} - \mathbf{t})^T Q(\mathbf{x} - \mathbf{t}))$, where $\mathbf{t}$ is the target. The outputs from the policies $\pi(\mathbf{x})$ were constrained by a saturation function: $\text{sat}(\mathbf{u}) = 9\sin(\mathbf{u})/8 + \sin(3\mathbf{u})/8$, where $\mathbf{u} = \tilde{\pi}(\mathbf{x})$. One experiment consisted of (1; 5) random trials followed by (15; 30) learned trials for the cart and unicycle tasks respectively. Each experiment was repeated 100 times and averaged. Each trial was evaluated by running the policy 30 times, and averaging, though note that this was performed only for evaluation purposes—the algorithms only had access to 1 trial. Success was determined by whether the return of the final trial passed below a threshold. The threshold was calibrated at a total cost of below 15. Moreover, I checked all of the final trajectory distributions by plotting them, and the threshold based approach matched my visual classification. Trials were considered successful if the pendulum was swung up and kept balanced.

4.5.4 Learning experiment results

The success rates of cart-pole swing-up are in Tables 4.2 and 4.1 while Figure 4.3 compares the standard particle-based methods to the Gaussian shaping variants. The success rate of unicycle balancing is in Table 4.4. Figures 4.16 and 4.17 show the learning efficiency of the algorithms. Additionally, I tested TP and PILCO in the Angle Cost k=1 scenario, when Q was multiplied by 0.01 to test the importance of the moment matching approximation for "exploration", and the results were 97% success rate for TP and 46% success rate for PILCO, implying that moment matching may not be that crucial for exploration.

Table 4.1: Angle cost. Success rate of learning cart-pole swing-up

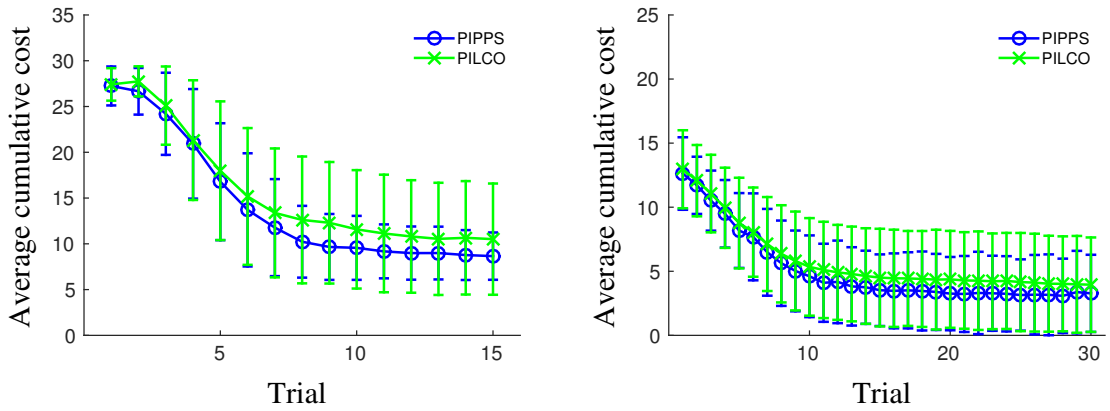
Noise Mult.	PILCO	RP	RP_{FS}	GR	GR_{FS}	LR	TP	TP$-\sigma_f$	TP$+\sigma_n$
$k = 10^{-2}$	**0.88**	0.69	0.24	0.63	0.74	0.57	**0.82**		**0.96**
$k = 1$	0.79	0.74	0.23	0.89	0.71	**0.96**	**0.99**	**0.93**	
$k = 4$	0.70	0.58	0.08	0.62	0.41	**0.94**	**0.95**	0.87	
$k = 9$	0.37	0.44	0.04	0.34	0.25	**0.86**	**0.83**	0.78	
$k = 16$	0.01	0.11	0.00	0.08	0.02	**0.45**	**0.40**	**0.42**	

Table 4.2: Tip cost. Success rate of learning cart-pole swing-up

Noise Mult.	PILCO	RP	RP_{FS}	GR	GR_{FS}	LR	TP
$k = 10^{-2}$	**0.92**	0.44	0.20	0.47	0.78	0.36	0.54
$k = 1$	**0.73**	0.15	0.08	**0.68**	0.50	0.28	0.48

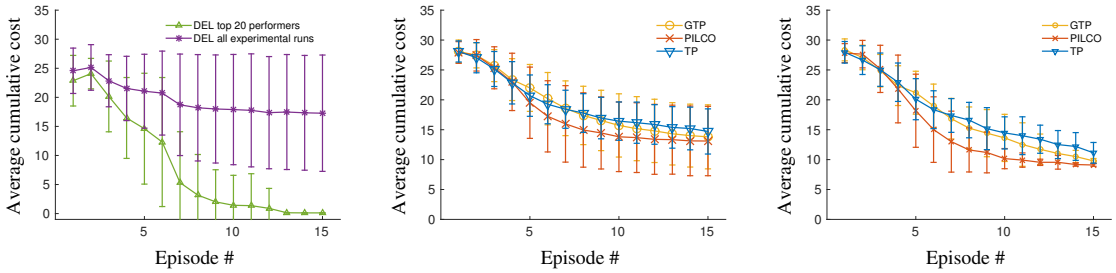
Table 4.3: Success rate of learning cart-pole swing-up: comparing to Gaussian shaping

Cost func.	σ_o^2 multiplier	PILCO	RP	GR	LR	TP	GTP	GLR	GTP$+\sigma_n$
Angle Cost	$k = 10^{-2}$	**0.88**	0.69	0.63	0.57	**0.82**	0.65	0.42	**0.88**
Angle Cost	$k = 1$	0.79	0.74	0.89	**0.96**	**0.99**	**0.9**	**0.93**	
Tip Cost	$k = 10^{-2}$	**0.92**	0.44	0.47	0.36	0.54	0.6	0.45	0.8
Tip Cost	$k = 1$	**0.73**	0.15	**0.68**	0.28	0.48	**0.69**	0.35	



(a) Cart-pole **(b)** Unicycle

Figure 4.16: PIPPS using TP matches PILCO in data-efficiency.



(a) Cart-pole balancing only **(b)** Swing-up and balancing All experimental runs **(c)** Swing-up and balancing Top 40 experimental runs

Figure 4.17: Data-efficiency and performance of learning algorithms on cart-pole tasks. Figures 4.17b and 4.17c correspond to the $k = 1$, *Tip Cost* case.

Table 4.4: Success rate of learning unicycle balancing

PILCO	RP	RP_{FS}	GR	GR_{FS}	LR	TP
0.91	0.80	0.39	**0.96**	0.63	0.47	**0.94**

4.6 Discussion

In the experiments where I plotted the objective landscape, I identified an issue with chaotic gradients, and showed that my total propagation algorithm was able to solve that problem. In the learning experiments, I wanted to test whether solving that issue translates to better performance in the final task. The main conclusion is that indeed with the new gradient estimators, I am able to match up in all tasks that the original PILCO was considered. Now it appears that the foundation has been laid to scale up such differentiable model-based algorithms to more difficult tasks by swapping out the models. Note however, that I have not yet attempted extremely high dimensional tasks, and this may prove problematic, as the LR component in my algorithm may need some further developments to scale up. However, all policy gradient algorithms rely on such gradient estimators, and as my algorithm incorporates RP gradients, it is expected to at the very least perform better than other currently available policy optimization algorithms.

4.6.1 Learning Experiments

PILCO performs well in scenarios with no noise, but with noise added the results deteriorate. This deterioration is most likely caused by the issue of moment matching becoming conservative, which I presented in the background section in Figure 1.4. Such an accumulation of errors in the MM approximations was previously also observed by Vinogradska et al. (2016), who used quadrature for predictions. Particles do not suffer from this issue, and using TP gradients consistently outperforms PILCO with high noise.

On the other hand, at low noise levels, the performance of TP as well as LR reduces. If all of the particles are sampled from a small region, it becomes difficult to estimate the gradient from changes in the return—in the limit of a delta distribution an LR gradient could not even be evaluated. The TP gradient is less susceptible to this problem, because it incorporates information from RP. Finally, if the uncertainty in predictions is very low (as in $k = 10^{-2}$), one can consider model noise as a parameter that affects learning, and increase it to acquire more accurate gradients: see TP $+ \sigma_n$, where the model noise variance was multiplied by 100, or GTP $+ \sigma_n$, where the noise was multiplied by 25.

Notably, approaches which use MM, such as PILCO and GR, outperform the others when using the *Tip Cost*. Looking only at the costs in Figures 4.17b and 4.17c does not adequately display the difference. In contrast, the success rates show that TP did not perform as well. The losses of the peak performers at the final episode were TP: 11.14 ± 1.73, GTP: 9.78 ± 0.40, PILCO: 9.10 ± 0.22, which also show that TP was significantly worse. While the peak performers were still improving, the remaining experiments had converged. PILCO still appears slightly more data-efficient; however, the difference has little practical significance as the required amount of data is low. Also note that in Figure 4.17b TP has smaller variance. The larger variance of GTP and PILCO is caused by outliers with a large loss. These outliers converged to a local minimum, which takes advantage of the tail of the Gaussian approximation of the state distribution—this contrasts with prior suggestions that PILCO performs exploration using the tail of the Gaussian (Deisenroth and Rasmussen, 2011). The reason why approaches that smoothed with a Gaussian outperformed the pure particle-based approaches may be the multi-modality of the objective—with the *Tip Cost*, the pendulum may be swung up from either direction to solve the task; with the *Angle Cost* there is only one correct direction. Performing MM forces the algorithm along a unimodal path, whereas the particle approach could attempt a bimodal swing-up where some particles go from one side, and the rest from the other side. Thus, MM may be performing a kind of "distributional reward shaping", simplifying the optimization problem. Such an explanation was previously provided by Gal et al. (2016). My Gaussian shaping gradient experiments further strengthen this claim. The Gaussian shaping gradient (GTP) allows adding this kind of bias into the gradient estimator, without modifying the trajectory distribution. GTP matches up to PILCO also in the *Tip Cost* scenarios. Thus, I have managed to match PILCO in all scenarios so far, and the algorithms are ripe for attempting to scale up to more difficult tasks, though note that this will also require swapping out the models used in PILCO.

Finally, I point to the surprising TP $- \sigma_f$ experiment. Even though the predictions ignore model uncertainty, the method achieved competitive success rate in Table 4.1, with the performance only slightly dropping. This is a striking difference to the result reported by Deisenroth et al. (2015), who claimed that when PILCO ignores the model uncertainty σ_f, the success rate is 0%. It is difficult to explain why learning still worked, but I hypothesize that the success may be related to the 0 prior mean of the GP. In regions where there is no data, the mean of the GP dynamics model goes to 0, meaning that the input control signal has no effect on the particle. Therefore, for the policy optimization to be successful, the particles would have to be controlled to stay in regions where there exists data. Note that a similar result was found by

Chatzilygeroudis et al. (2017) who used an evolutionary algorithm and achieved 85-90% success rate at the cart-pole task even when ignoring model uncertainty. Finally, I note that many other works have also showed that incorporating model uncertainty is important (Kurutach et al., 2018; Chua et al., 2018), although none reported a 0% success rate when ignoring model uncertainty.

4.6.2 The Curse of Chaos in Deep Learning and elsewhere

Most machine learning problems involve optimizing the expectation of an objective function $J(\mathbf{x};\theta)$ over some data generating distribution $p_{Data}(\mathbf{x})$, where this distribution can only be accessed through sample data points $\{\mathbf{x}_i\}$. The predictive framework is analogous to a deep model: $p(\mathbf{x}_0)$ is the data generating distribution, $p(\mathbf{x}_t;\theta)$ are obtained by pushing $p_{Data}(\mathbf{x})$ through the model layers. The most common method of optimization is SGD with pathwise derivatives computed by backpropagation. My results suggest that in some situations—particularly with very deep or recurrent models—this approach could degenerate into a random walk due to an exploding gradient variance.

Exploding gradients have been observed in deep learning research for a long time (Doya, 1993; Bengio et al., 1994). Typically this phenomenon is regarded as a numerical issue, which leads to large steps and unstable learning. Common countermeasures include gradient clipping, ReLU activation functions (Nair and Hinton, 2010) and smart initializations. My explanation to the problem is different: it is not just that the gradient becomes large, the gradient variance explodes, meaning that any sample from $\mathbf{x}_i \sim p_{Data}$ provides essentially no information about how to change the model parameters θ to increase the expectation of the objective over the whole distribution $\mathbb{E}_{p_{Data}}[J(\mathbf{x})]$. While choosing a good initialization is an approach to tackle the problem, it appears difficult to guarantee that the system does not become chaotic during learning. For example, in econometrics there are even cases where the optimal policy may lead to chaotic dynamics (Deneckere and Pelikan, 1986). Gradient clipping can stop large parameter steps, but it will not fundamentally solve the problem if the gradients become random. Considering that chaos does not occur in linear systems (Alligood et al., 1996), my analysis suggests a reason for why piece-wise linear activations, which may be less susceptible to chaos, such as ReLUs perform well in deep learning.

While I have yet to computationally confirm my hypothesis regarding deep learning, several works have investigated chaos in neural networks (Kolen and Pollack, 1991; Sompolinsky et al., 1988), to name a few. Notably, Poole et al. (2016) suggested that such properties lead to "exponential expressivity", but I believe that this phenomenon may instead be a curse.

As further evidence that what I have found is correct: note that Ingraham et al. (2019) concurrently investigated issues with chaos in differentiable protein folding software, and found that the same issue with gradients occurs in their problem. They proposed to reduce the issue with chaos by adding a regularizer into their optimization task; however, such an approach can restrict the solution space, and may damage the performance. As another work, soon after I had published my results, Metz et al. (2019) built on my findings, and investigated issues with chaos in meta-learning in training optimizers that could optimize faster than hand-designed optimization algorithms, and they also observed issues with chaos, and used my suggestion of inverse

variance weighting to achieve much better performance.

I started with a specific problem of trying to make model-based reinforcement learning work while differentiating through the model-predictions. However, the problems that I encountered with chaos are actually more general and prevalent in many areas of machine learning where there are repeated non-linear computations. The solution that I found, total propagation, is general and I hope it could also be applied in many other scenarios.

Conclusions & Suggestions

My thesis went all the way from explaining novel views of elementary gradient estimators to how such estimators can be combined on a graph of computations, and finally applied the new techniques in model-based reinforcement learning algorithms.

I explained that the sampling distribution used to estimate likelihood ratio gradients should be kept separate from the distribution used to define the objective, and showed that drastic reductions in gradient variance can be achieved by using an optimal importance sampling distribution. Moreover, I discussed baseline techniques for such gradient estimators, and argued that the variance of the baseline estimation itself should also be taken into account in designing gradient variance reduction techniques.

I provided a novel framework for decomposing the total derivative in a graph of computations into partial derivatives along computational paths. My framework generalized standard "policy gradient theorems" into non-Markovian reinforcement learning environments, and provided an intuitive visual method to derive new gradient estimators. I used this framework to derive new estimators: the density estimation likelihood ratio gradient estimator as well as the Gaussian shaping gradient estimators. In particular the Gaussian shaping gradient estimator is exciting, because it allows for the gradient computation, to "jump" over multiple nodes in the computational graph to some distal node, then continue applying the chain rule from that node onward. Such a technique may prove useful in large graphs of complex interactions.

Finally, I investigated gradient estimation techniques in model-based reinforcement learning. Surprisingly, the go to solution of applying reparameterization gradients together with backpropagation proved to be hopelessly poor in situations where the dynamics become chaotic. I argued that such situations occur in many fields of machine learning and cast doubt on whether standard backpropagation-based approaches will be sufficient in the long term of machine learning research. Not only did I elucidate such phenomena, but I also provided a direction to overcome such challenges. Among my key contributions was the creation of the total propagation algorithm, which is an elegant algorithm for combining gradient estimators during the backwards gradient computation. Such an algorithm may be able to utilize the good scaling of backpropagation-based approaches, while achieving the robustness of likelihood ratio gradient estimators, and may allow overcoming challenges due to chaos. It appears that I may have been able to lay a foundation for scaling up differentiable model-based reinforcement learning, and the next step should be to swap out the rudimentary models used in PILCO to more advanced models, and try out more challenging tasks.

Unfortunately, current software frameworks, such as TensorFlow (Abadi et al., 2015), PyTorch (Paszke et al., 2017) and Chainer (Tokui et al., 2015) are not com-

patible with my new gradient estimation techniques. I envision that ever more flexible and customizable gradient estimation frameworks will be necessary. Gradient descent has enabled many exciting technologies in machine learning, and I believe that new gradient estimation techniques will enable applications previously unimaginable.

Appendix A

Gaussian process models

One way to represent a function is as an infinite vector containing one entry $f(\mathbf{x})$ for the function value at each argument location $\mathbf{x}$. Without additional assumptions, this representation is not useful. In a Gaussian process, the additional assumption is that any finite subset of points in this vector are jointly Gaussian distributed:

$$\begin{bmatrix} f(\mathbf{x}_1) \\ \vdots \\ f(\mathbf{x}_n) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \vdots \\ \mu_n \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \dots & \Sigma_{1n} \\ \vdots & \ddots & \vdots \\ \Sigma_{n1} & \dots & \Sigma_{nn} \end{bmatrix} \right) \tag{A.1}$$

In a machine learning context, we do not know this infinite dimensional distribution, we will only know it at a finite number number of points. However, Gaussian distributions satisfy the marginalization property: if one integrates over one block of dimensions in a Gaussian distribution, the remaining distribution will simply be the other block. In other words, we only need to know the covariances between the argument values that we are interested in.

Furthermore, if one knows the function value at some points, one can compute a conditional distribution at any other argument with the following equations:

$$\begin{bmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right) \tag{A.2}$$

In this distribution both F-s can be any length vectors. Given that Equation (A.2) holds, if we know $\mathbf{F}_1$, it is possible to condition the values of $\mathbf{F}_2$ on these values using the following equations:

$$\begin{aligned} \mu &= \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{F}_1 - \mu_1) \\ \Sigma &= \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12} \\ \mathbf{F}_2 &\sim \mathcal{N}(\mu, \Sigma) \end{aligned} \tag{A.3}$$

To perform predictions with the the values of $\mathbf{F}_1$, the only remaining issue is how to calculate the covariances and the means in Equation (A.2).

The mean gets calculated using the mean function $m(\mathbf{x})$. This is often set to either 0, or for example in PILCO, it is set to $m(\mathbf{x}) = \mathbf{x}$. Note that strictly speaking, each Gaussian process will have one output, so in practice the mean function for one output dimension will only copy the value of that particular dimension.

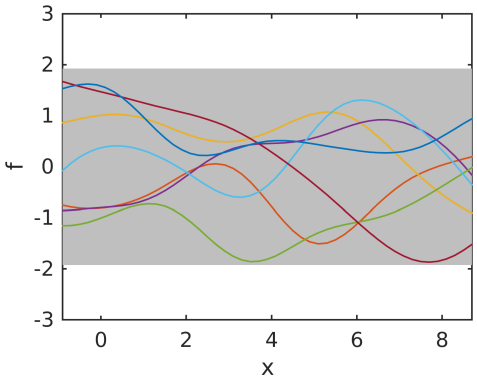


Figure A.1: Gaussian process prior on functions, together with a few sample functions drawn from this prior.

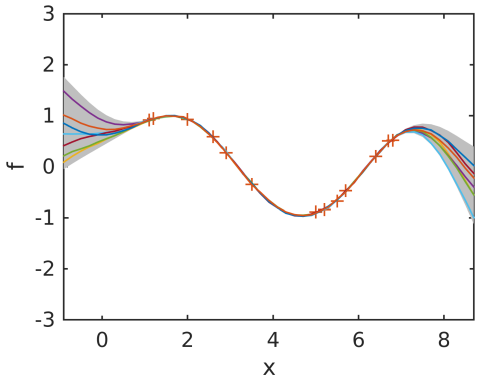


Figure A.2: Posterior function distribution after some data has been observed. Only the functions, which can explain the data remain.

The more important component of a Gaussian process is the covariance function $k(\mathbf{x}_i, \mathbf{x}_j)$. The covariance function is used to compute each entry in the covariance matrix. There are various kinds of convariance matrices that can capture different structures in the data, such as periodicity, symmetricity about a point, etc. PILCO uses possibly the most common covariance function, the squared exponential covariance function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp(-(\mathbf{x}_i - \mathbf{x}_j)^T \Lambda^{-1} (\mathbf{x}_i - \mathbf{x}_j)) \tag{A.4}$$

This covariance function says that if two data points are close to each other, their function value will be similar (the covariance will be higher). σ_f^2 is a hyperparameter which gives an indication of the size of the range over which the function values vary. Λ is a diagonal matrix of length-scale hyperparameters, which say over what distance two points covary. These hyperparameters will get optimized by maximizing the marginal likelihood of the data, and this will be explained below.

There is another hyperparameter, which determines the intrinsic variability of the observations, the noise hyperparameter. The noise is assumed to be Gaussian. The likelihood of a vector of observations $\mathbf{Y}$ for a given set of function values $\mathbf{F}$ becomes $\mathcal{N}(\mathbf{Y}; \mathbf{F}, I\sigma_n^2)\mathcal{N}(\mathbf{F}; M(X), K(X, X))$, where X are all of the argument values of each data point, $M(X)$ are the outputs of the mean function for each data point, and $K(X, X)$ is the covariance matrix evaluated with the covariance function. A product of two Gaussians is a Gaussian, and this can be easily integrated. To find the the marginal likelihood, the hidden variables F get integrated out. The final result is given by the equation below.

$$L \quad = \quad \mathcal{N}(\mathbf{Y}; M(X), I\sigma_n^2 + K(X, X)) \tag{A.5}$$

Usually, one takes the log of this equation, computes the derivatives with respect to the hyperparameters, and optimizes the hyperparameters. This kind of maximization of the marginal likelihood implements an automatic Occam's razor: if the noise is small, there will be few functions that fit the data well, if the noise is very large, so that many

functions could explain the data, the likelihood of this particular data set given any function will be fairly low (Rasmussen and Ghahramani, 2001). The algorithm finds a middle ground where many functions are able to explain the data. It thus penalises functions that are too complicated, while still trying to get a good fit.

To perform predictions when the noise is included, one should use the covariance matrix including the noise and combine this with Equation (A.3).

A Gaussian process models a whole distribution of functions. The covariance function is a prior on the type of functions. When data is observed, only the functions that can explain the data will remain. This is illustrated in Figures A.1 and A.2.

Appendix B

Basic vector calculus and fluid mechanics

Here I illustrate the background information in 3 dimensions, but it generalizes straightforwardly to higher dimensions.

Notation:
$\mathbf{F} = [F_x(x,y,z), F_y(x,y,z), F_z(x,y,z)]$ is a vector field.
$\phi(x,y,z)$ is a scalar field (a scalar function)
Div operator: $\nabla \cdot \mathbf{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}$.
Grad operator: $\nabla\phi = [\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}, \frac{\partial \phi}{\partial z}]$.

The vector field $\mathbf{F}$ could be for example thought of as a local flow velocity for some fluid. If $\mathbf{F}$ is the density flow rate, then the div operator essentially measures how much the density is decreasing at a point. If the outflow is larger than the inflow, the density would decrease and vice versa. The divergence theorem, illustrated in Figure B.1 illustrates how this change in density can be measured in two separate ways: one could integrate the divergence across the volume, or one could integrate the in and and outflow across the surface. The divergence theorem states:

$$\int_V \nabla \cdot \mathbf{F} \mathrm{d}V = \int_S \mathbf{F} \cdot \mathrm{d}\mathbf{S} \tag{B.1}$$

To prove the claim, consider the infinitesimal box in Figure B.1. The divergence can be calculated as $\delta x \delta y(\frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y})$. On the other hand, to take the integral across the surface, note that the surface normals point outwards, and the integral becomes $\delta_y(-F_x + F_x - \frac{\partial F_x}{\partial x}\delta x) + \delta x(-F_y + F_y + \frac{\partial F_y}{\partial y}\delta y) = \delta x \delta y(\frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y})$, which is the same as the divergence. To generalize this to arbitrarily large volumes, notice that if one stacks the boxes next to each other, then the surface integral across the area where the boxes meet cancels out, and only the integral across the outer surface remains. For an incompressible flow, the density does not change, and the divergence must be zero.

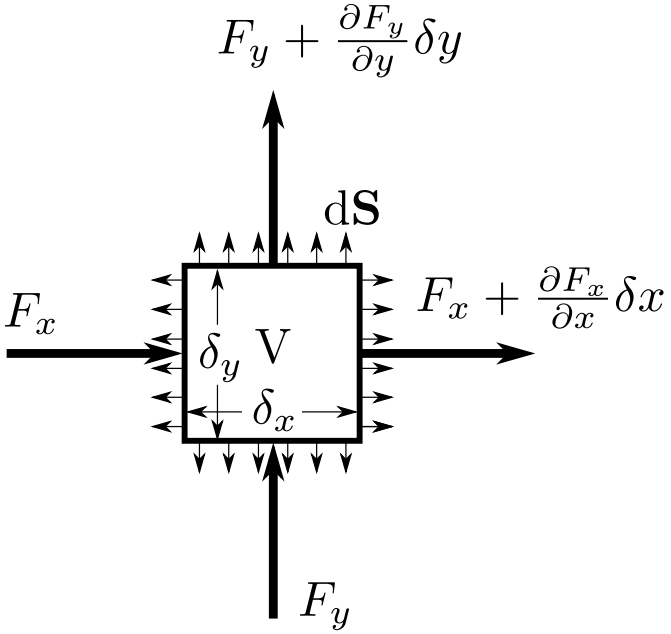


Figure B.1: Illustration of the divergence theorem.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. 4.6.2

Alligood, K. T., Sauer, T. D., and Yorke, J. A. (1996). *Chaos.* Springer. 4.2.1, 4.2.2, 4.6.2

Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J. Z. (2018). Differentiable MPC for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, pages 8299–8310. 1.3.4

Bauer, M., van der Wilk, M., and Rasmussen, C. E. (2016). Understanding probabilistic sparse Gaussian process approximations. In *Advances in neural information processing systems*, pages 1533–1541. 1.3.4

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166. 4.6.2

Bischoff, B., Nguyen-Tuong, D., Koller, T., Markert, H., and Knoll, A. (2013). Learning throttle valve control using policy search. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 49–64. Springer. 1.3.3, 1.3.5

Bischoff, B., Nguyen-Tuong, D., van Hoof, H., McHutchon, A., Rasmussen, C. E., Knoll, A., Peters, J., and Deisenroth, M. P. (2014). Policy search for learning robot control using sparse data. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3882–3887. IEEE. 1.3.3

Chatzilygeroudis, K., Rama, R., Kaushik, R., Goepp, D., Vassiliades, V., and Mouret, J.-B. (2017). Black-box data-efficient policy search for robotics. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 51–58. IEEE. 4.6.1

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765. 4, 4.6.1

Ciosek, K. and Whiteson, S. (2017). Expected policy gradients. *arXiv preprint arXiv:1706.05374*. 3.2.4

Corless, R. M., Gonnet, G. H., Hare, D. E., Jeffrey, D. J., and Knuth, D. E. (1996). On the Lambert W function. *Advances in Computational mathematics*, 5(1):329–359. 2.2.2

Deisenroth, M., Fox, D., and Rasmussen, C. (2014). Gaussian processes for data-efficient learning in robotics and control. *Transactions on Pattern Analysis and Machine Intelligence*, 36(5):1–1. 1.3.3

Deisenroth, M. P. (2010). *Efficient reinforcement learning using Gaussian processes*, volume 9. KIT Scientific Publishing. 1.3.3

Deisenroth, M. P., Calandra, R., Seyfarth, A., and Peters, J. (2012). Toward fast policy search for learning legged locomotion. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1787–1792. IEEE. 1.3.3

Deisenroth, M. P. and Fox, D. (2011). Multiple-target reinforcement learning with a single policy. In *ICML 2011 Workshop on Planning and Acting with Uncertain Models*. Citeseer. 1.3.3

Deisenroth, M. P., Fox, D., and Rasmussen, C. E. (2015). Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423. 4.5, 4.5.2, 4.6.1

Deisenroth, M. P. and Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, pages 465–472. (document), 1.3, 1.3, 1.3.3, 4.6.1

Deisenroth, M. P., Rasmussen, C. E., and Fox, D. (2011). Learning to control a low-cost manipulator using data-efficient reinforcement learning. 1.3.3

Deneckere, R. and Pelikan, S. (1986). Competitive chaos. *Journal of economic theory*, 40(1):13–25. 4.6.2

Depeweg, S., Hernández-Lobato, J. M., Doshi-Velez, F., and Udluft, S. (2016). Learning and policy search in stochastic dynamical systems with Bayesian neural networks. *arXiv preprint arXiv:1605.07127*. 4

Doya, K. (1993). Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on neural networks*, 1:75–80. 4.6.2

Englert, P., Paraschos, A., Deisenroth, M. P., and Peters, J. (2013). Probabilistic model-based imitation learning. *Adaptive Behavior*, 21(5):388–403. 1.3.3

Fairbank, M. and Alonso, E. (2012). Value-gradient learning. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE. 3.2.3

Fleiss, J. (1993). Review papers: The statistical basis of meta-analysis. *Statistical methods in medical research*, 2(2):121–145. 4.4

Fu, M. C. and Hu, J.-Q. (1995). Sensitivity analysis for Monte Carlo simulation of option pricing. *Probability in the Engineering and Informational Sciences*, 9(3):417–446. 1.1

Gal, Y., McAllister, R., and Rasmussen, C. (2016). Improving PILCO with bayesian neural network dynamics models. In *Workshop on Data-efficient Machine Learning, ICML*. 1.3.3, 4, 4.3.1, 4.6.1

Glynn, P. W. (1990). Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84. 1.1.1

Greensmith, E., Bartlett, P. L., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530. 1.1.1, 2, 2.3.2

Ha, D. (2017). Evolving stable strategies. *blog.otoro.net*. 2.3.2

Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462. 4

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019). Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. 4

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. (2015). Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952. 3.2.3, 4

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347. 1.1

Ingraham, J., Riesselman, A., Sander, C., and Marks, D. (2019). Learning protein structure with a differentiable simulator. In *International Conference on Learning Representations*. 4.6.2

Jankowiak, M. and Obermeyer, F. (2018). Pathwise derivatives beyond the reparameterization trick. In *International Conference on Machine Learning*, pages 2240–2249. 2.1.2, 2.1.2

Jie, T. and Abbeel, P. (2010). On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems*, pages 1000–1008. 2.1.1, 2.4.1

Jones, E., Oliphant, T., Peterson, P., et al. (2001–). SciPy: Open source scientific tools for Python. [Online; accessed May 2019]. 2.2.2, 6

Jordan, M. I. and Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive science*, 16(3):307–354. (document)

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2.3.2

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*. 2

Kolen, J. F. and Pollack, J. B. (1991). Back propagation is sensitive to initial conditions. In *Advances in neural information processing systems*, pages 860–867. 4.6.2

Körding, K. P. and Wolpert, D. M. (2004). The loss function of sensorimotor learning. *Proceedings of the National Academy of Sciences*, 101(26):9839–9842. 4.1

Kupcsik, A., Deisenroth, M. P., Peters, J., Loh, A. P., Vadakkepat, P., and Neumann, G. (2014). Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*. 1.3.4

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. (2018). Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*. 4, 4.6.1

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436. (document), 4.4

Mahsereci, M. and Hennig, P. (2015). Probabilistic line searches for stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 181–189. 4.2

Mania, H., Guy, A., and Recht, B. (2018). Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1800–1809. 2.1.1, 2.3.2

McAllister, R. (2017). *Bayesian Learning for Data-Efficient Control*. PhD thesis, Department of Engineering, University of Cambridge. 1.3.4

McAllister, R. and Rasmussen, C. E. (2016). Data-efficient reinforcement learning in continuous-state POMDPs. *arXiv preprint arXiv:1602.02523*. 1.3.1, 1.3.3

McAllister, R., van der Wilk, M., and Rasmussen, C. (2016). Data-efficient policy search using PILCO and directed-exploration. In *Workshop on Data-efficient Machine Learning, ICML*. 1.3.3, 1.3.4

McHutchon, A. (2014). *Modelling nonlinear dynamical systems with Gaussian Processes*. PhD thesis, University of Cambridge. 1.3.4, 1.3.6, 4.3.1

Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., and Sohl-Dickstein, J. (2019). Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565. 4.6.2

Mnih, A. and Rezende, D. (2016). Variational inference for Monte Carlo objectives. In *International Conference on Machine Learning*, pages 2188–2196. 1.1.1, 2.4.2

Murray, I. (2016). Differentiation of the Cholesky decomposition. *arXiv preprint arXiv:1602.07527*. 4.3.1

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814. 4.6.2

Naumann, U. (2008). Optimal Jacobian accumulation is NP-complete. *Mathematical Programming*, 112(2):427–441. 3.1.2

Nesterov, Y. and Spokoiny, V. (2017). Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566. 4.4

Ng, A. Y. and Jordan, M. (2000). Pegasus: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc. 4.5.1

Nguyen, D. H. and Widrow, B. (1990). Neural networks for self-learning control systems. *IEEE Control systems magazine*, 10(3):18–23. 4

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media. 1.3, 4.5.1

Owen, A. B. (2013). *Monte Carlo theory, methods and examples*. 2.2.2

Parmas, P., Rasmussen, C. E., Peters, J., and Doya, K. (2018). PIPPS: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*. 3, 3.3, 3.3.2

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*. 4.6.2

Pearl, J. (2014). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier. 3

Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., and Ganguli, S. (2016). Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pages 3360–3368. 4.6.2

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*. 1.3.4

Rasmussen, C. E. and Ghahramani, Z. (2001). Occam's razor. *Advances in neural information processing systems*, pages 294–300. A

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press. 1

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*. 1.1.2

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1. (document)

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*. 2, 2.1.1, 2.3.2, 3.2.2

Schmidhuber, J. (2015a). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117. (document), 4.4

Schmidhuber, J. (2015b). On learning to think: Algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models. *arXiv preprint arXiv:1511.09249*. 4

Schulman, J., Heess, N., Weber, T., and Abbeel, P. (2015). Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pages 3528–3536. 3, 3.1.3, 4.4

Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., and Schmidhuber, J. (2010). Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559. 3.2.2

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning*. 1.1.1, 3, 3.2.1, 3.2.1

Sompolinsky, H., Crisanti, A., and Sommers, H.-J. (1988). Chaos in random neural networks. *Physical review letters*, 61(3):259. 4.6.2

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*, pages 216–224. Elsevier. 4

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge. (document), 3.2.1

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063. (document), 3.2.1, 3.2.1

Tangkaratt, V., Mori, S., Zhao, T., Morimoto, J., and Sugiyama, M. (2014). Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation. *Neural networks*, 57:128–140. 2.4.2

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31. 4.5.1

Tokui, S., Oono, K., Hido, S., and Clayton, J. (2015). Chainer: a next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*. 4.6.2

Vinogradska, J., Bischoff, B., Schmidt, H., and Romer, A. (2016). Stability of controllers for Gaussian process forward models. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 545–554. 1.3.3, 1.3.4, 4.6.1

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge. 1.1

Weaver, L. and Tao, N. (2001). The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth Conference on Uncertainty in artificial intelligence*, pages 538–545. Morgan Kaufmann Publishers Inc. 2, 2.4, 2.4.1

Weber, T., Heess, N., Buesing, L., and Silver, D. (2019). Credit assignment techniques in stochastic computation graphs. *arXiv preprint arXiv:1901.01761*. 3

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*. (document)

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256. 1.1.1