Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs

by

John Schulman

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Chair
Professor Stuart Russell
Professor Michael Jordan
Assistant Professor Joan Bruna

Fall 2016

Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs

Abstract

Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs

by

John Schulman

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Chair

This thesis is mostly focused on reinforcement learning, which is viewed as an optimization problem: maximize the expected total reward with respect to the parameters of the policy. The first part of the thesis is concerned with making policy gradient methods more sample-efficient and reliable, especially when used with expressive nonlinear function approximators such as neural networks. Chapter 3 considers how to ensure that policy updates lead to monotonic improvement, and how to optimally update a policy given a batch of sampled trajectories. After providing a theoretical analysis, we propose a practical method called *trust region policy optimization* (TRPO), which performs well on two challenging tasks: simulated robotic locomotion, and playing Atari games using screen images as input. Chapter 4 looks at improving sample complexity of policy gradient methods in a way that is complementary to TRPO: reducing the variance of policy gradient estimates using a state-value function. Using this method, we obtain state-of-the-art results for learning locomotion controllers for simulated 3D robots.

Reinforcement learning can be viewed as a special case of optimizing an expectation, and similar optimization problems arise in other areas of machine learning; for example, in variational inference, and when using architectures that include mechanisms for memory and attention. Chapter 5 provides a unifying view of these problems, with a general calculus for obtaining gradient estimators of objectives that involve a mixture of sampled random variables and differentiable operations. This unifying view motivates applying algorithms from reinforcement learning to other prediction and probabilistic modeling problems.

ACKNOWLEDGMENTS

All of the work described in this thesis was done in collaboration with my advisor, Pieter Abbeel, who has continually pointed me in the right direction and provided inspiration to do the best work I could.

I'd also like to thank Sergey Levine and Philipp Moritz, who were my closest collaborators on the main work in the thesis, and with whom I shared many great conversations. The work on stochastic computation graphs grew out of discussions with my coauthors Pieter Abbeel, Nick Heess, and Theophane Weber. Thanks to Mike Jordan, Stuart Russell, and Joan Bruna for serving on my quals and thesis committee, and for many insightful conversations over the past few years. I also collaborated with a number of colleagues at Berkeley on several projects that are not included in this thesis document, including Jonathan Ho, Alex Lee, Sachin Patil, Zoe McCarthy, Greg Kahn, Michael Laskey, Ibrahim Awwal, Henry Bradlow, Jia Pan, Cameron Lee, Ankush Gupta, Sibi Venkatesan, Mallory Tayson-Frederick, and Yan Duan. I am thankful to DeepMind for giving me the opportunity to do an internship there in the Spring of 2015, so I would like to thank my supervisor, David Silver, as well as Yuval Tassa, Greg Wayne, Tom Erez, and Tim Lillicrap.

Thanks to UC Berkeley for being flexible and allowing me to switch from the neuroscience program to the computer science program without much difficulty. Thanks to the wonderful staff at Philz Coffee in Berkeley, where most of the research and writing was performed, along with Brewed Awakening, Nefeli's, Strada, and Asha Tea House.

Finally, this thesis is dedicated to my parents, for all the years of love and support, and for doing so much to further my education.

CONTENTS

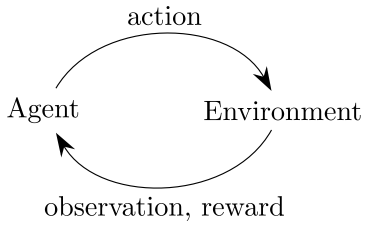
LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 REINFORCEMENT LEARNING



Reinforcement learning (RL) is the branch of machine learning that is concerned with making sequences of decisions. It considers an agent situated in an environment: each timestep, the agent takes an action, and it receives an observation and reward. An RL algorithm seeks to maximize the agent's total reward, given a previously unknown environment, through a trial-and-error learning process. Chapter 2 provides a more detailed description of the mathematical formulation of reinforcement learning.

The reinforcement learning problem sketched above, involving a reward-maximizing agent, is extremely general, and RL algorithms have been applied in a variety of different fields, from business inventory management [VR+97] to robot control [KBP13], to structured prediction [DILM09]

1.2 DEEP LEARNING

Modern machine learning is mostly concerned with learning functions from data. Deep learning is based on a simple recipe: choose a loss function, choose an expressive function approximator (a deep neural network), and optimize the parameters with gradient descent. The remarkable empirical finding is that it is possible to learn functions that perform complicated multi-step computations with this recipe, as has been shown by groundbreaking results in object recognition [KSH12] and speech recognition [Dah+12]. The recipe involves a reduction from a learning problem to an optimization problem: in supervised learning, we are reducing *obtain a function that makes good predictions on unseen* *data*, to *minimize prediction-error-plus-regularization on training data*.

The reduction from learning to optimization is less straightforward in reinforcement learning (RL) than it is in supervised learning. One difficulty is that we don't have full analytic access to the function we're trying to optimize, the agent's expected total reward—this objective also depends on the unknown dynamics model and reward function. Another difficulty is that the agent's input data strongly depends on its behavior, which makes it hard to develop algorithms with monotonic improvement. Complicating the problem, there are several different functions that one might approximate, as we will discuss in Section 1.4

1.3 DEEP REINFORCEMENT LEARNING

Deep reinforcement learning is the study of reinforcement using neural networks as function approximators. The idea of combining reinforcement learning and neural networks is not new—Tesauro's TD-Gammon [Tes95], developed in the early 1990s, used a neural network value function and played at the level of top human players, and neural networks have been used for long time in system identification and control [NP90]. Lin's 1993 thesis [Lin93] explored the combination of various reinforcement learning algorithms with neural networks, with application to robotics.

However, in the two decades following Tesauro's results, RL with nonlinear function approximation remained fairly obscure. At the time when this thesis work was beginning (2013), none of the existing RL textbooks (such as [SB98; Sze10]) devoted much attention to nonlinear function approximation. Most RL papers, in leading machine learning conferences such as NIPS and ICML were mostly focused on theoretical results and on toy problems where linear-in-features or tabular function approximators could be used.

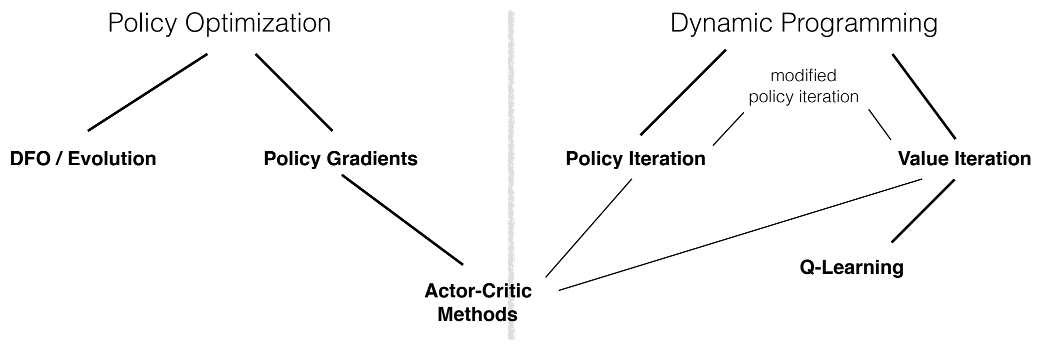
In the early 2010s, the field of deep learning begin to have groundbreaking empirical success, in speech recognition [Dah+12] and computer vision [KSH12]. The work described in this thesis began after the realization that similar breakthroughs were possible (and inevitable) in reinforcement learning, and would eventually dominate the special-purposes methods which were being used in domains like robotics. Whereas much work in reinforcement learning only applies in the case of linear or tabular functions, such methods will not be applicable in settings where we need to learn functions that perform multi-step computation. On the other hand, deep neural networks can successfully approximate these functions, and their empirical success in supervised learning shows that it is tractable to optimize them.

An explosion of interest in deep reinforcement learning occurred following the results from Mnih et al. [Mni+13], who demonstrated learning to play a collection of Atari games, using screen images as input, using a variant of Q-learning. These results improved on previous results obtained by Hausknecht et al. [Hau+12] using an evolutionary algorithm, despite using a more challenging input representation. Since then, there have been many interesting results occurring concurrently with the work described in this thesis. To sample a couple of the more influential ones, Silver et al. [Sil+16] learned to play Go better than the best human experts, using a combination of supervised learning and several reinforcement learning steps to train deep neural networks, along with a tree search algorithm. Levine et al. [Lev+16] showed the learning of manipulation behaviors on a robot from vision input, with a small number of inputs. Mnih et al. [Mni+16] demonstrated strong results with a classic policy gradient method on a variety of tasks. Silver et al. [Sil+14], Lillicrap et al. [Lil+15], and Heess et al. [Hee+15] explored a different kind of policy gradient method, which can be used in settings with a continuous action space. Furthermore, there have been a variety of improvements on the original Deep Q-learning algorithm [Mni+13], including methods for exploration [Osb+16] and stability improvements [VHGS15].

1.4 WHAT TO LEARN, WHAT TO APPROXIMATE

In reinforcement learning there are many different choices of what to approximate—policies, value functions, dynamics models, or some combination thereof. This contrasts with supervised learning, where one usually learns the mapping from inputs to outputs. In reinforcement learning, we have two orthogonal choices: what kind of objective to optimize (involving a policy, value function, or dynamics model), and what kind of function approximators to use.

The figure below shows a taxonomy of model-free RL algorithms (algorithms that do are not based on a dynamics model). At the top level, we have two different approaches for deriving RL algorithms: policy optimization and dynamic programming.



Policy optimization methods are centered around the *policy*, the function that maps the agent's state to its next action. These methods view reinforcement learning as a numerical optimization problem where we optimize the expected reward with respect to the policy's parameters. There are two ways to optimize a policy. First, there are derivative free optimization (DFO) algorithms, including evolutionary algorithms. These algorithms work by perturbing the policy parameters in many different ways, measuring the performance, and then moving in the direction of good performance. They are simple to implement and work very well for policies with a small number of parameters, but they scale poorly with the number of parameters. Some DFO algorithms used for policy optimization include cross-entropy method [SL06], covariance matrix adaptation [WP09], and natural evolution strategies [Wie+08] (these three use Gaussian distributions); and HyperNEAT, which also evolves the network topology [Hau+12]. Second, there are policy gradient methods [Wil92; Sut+99; JJS94; Kak02]. These algorithms can estimate the policy improvement direction by using various quantities that were measured by the agent; unlike DFO algorithms, they don't need to perturb the parameters to measure the improvement direction. Policy gradient methods are a bit more complicated to implement, and they have some difficulty optimizing behaviors that unfold over a very long timescale, but they are capable of optimizing much larger policies than DFO algorithms.

The second approach for deriving RL algorithms is through approximate dynamic programming (ADP). These methods focus on learning *value functions*, which predict how much reward the agent is going to receive. The true value functions obey certain consistency equations, and ADP algorithms work by trying to satisfy these equations. There are two well-known algorithms for exactly solving RL problems that have a finite number of states and actions: policy iteration and value iteration. (Both of these algorithms are special cases of a general algorithm called modified policy iteration.) These algorithms can be combined with function approximation in a variety of different ways; currently, the leading descendents of value iteration work by approximating Q-functions (e.g., [Mni+15]).

Finally, there are actor-critic methods that combine elements from both policy optimization and dynamic programming. These methods optimize a policy, but they use value functions to speed up this optimization, and often use ideas from approximate dynamic programming to fit the value functions. The method described in Chapter 4, along with deterministic policy gradient methods [Lil+15; Hee+15], are examples of actor-critic methods.

1.5 OPTIMIZING STOCHASTIC POLICIES

This thesis focuses on a particular branch in the family tree of RL algorithms from the previous section—methods that optimize a stochastic policy, using gradient based methods. Why *stochastic* policies, (defining $\pi(a \mid s) =$ probability of action given state) rather than deterministic policies ($a = \pi(s)$)? Stochastic policies have several advantages:

- Even with a discrete action space, it's possible to make an infinitesimal change to a stochastic policy. That enables *policy gradient methods*, which estimate the gradient of performance with respect to the policy parameters. Policy gradients do not make sense with a discrete action space.
- We can use the *score function* gradient estimator, which tries to make good actions more probable. This estimator, and its alternative, the *pathwise derivative* estimator, will be discussed in Chapter 5. The score function estimator is better at dealing with systems that contain discrete-valued or discontinuous components.
- The randomness inherent in the policy leads to exploration, which is crucial for most learning problems. In other RL methods that aren't based on stochastic policies, randomness usually needs to be added in some other way. On the other hand, stochastic policies explore poorly in many problems, and policy gradient methods often converge to suboptimal solutions.

The approach taken in this thesis—optimizing stochastic policies using gradient-based methods—makes reinforcement learning much more like other domains where deep learning is used. Namely, we repeatedly compute a noisy estimate of the gradient of performance, and plug that into a stochastic gradient descent algorithm. This situation contrasts with methods that use function approximation along with dynamic programming methods like value iteration and policy iteration—there, we can also formulate optimization problems; however, we are not directly optimizing the expected performance. While there has been success using neural networks in value iteration [Mni+13], this sort of algorithm is hard to analyze because it is not clear how errors in the dynamic programming updates will accumulate or affect the performance—thus, these methods have not shown good performance across as wide of a variety of tasks that policy gradient methods have; however, when they work, they tend to be more sample-efficient than policy gradient methods.

While the approach of this thesis simplifies the problem of reinforcement learning by reducing it to a more well-understood kind of optimization with stochastic gradients, there are still two sources of difficulty that arise, motivating the work of this thesis.

1. Most prior applications of deep learning involve an objective where we have access to the loss function and how it depends on the parameters of our function approximator. On the other hand, reinforcement learning involves a dynamics model that is unknown and possibly nondifferentiable. We can still obtain gradient estimates, but they have high variance, which leads to slow learning.
2. In the typical supervised learning setting, the input data doesn't depend on the current predictor; on the other hand, in reinforcement learning, the input data strongly depends on the current policy. The dependence of the state distribution on the policy makes it harder to devise stable reinforcement learning algorithms.

1.6 contributions of this thesis

This thesis develops policy optimization methods that are more stable and sample efficient than their predecessors and that work effectively when using neural networks as function approximators.

First, we study the following question: *after collecting a batch of data using the current policy, how should we update the policy?* In a theoretical analysis, we show that there is certain loss function that provides a local approximation of the policy performance, and the accuracy of this approximation is bounded in terms of the KL divergence between the old policy (used to collect the data) and the new policy (the policy after the update). This theory justifies a policy updating scheme that is guaranteed to monotonically improve the policy (ignoring sampling error). This contrasts with previous analyses of policy gradient methods (such as [JJS94]), which did not specify what finite-sized stepsizes would guarantee policy improvement. By making some practically-motivated approximations to this scheme, we develop an algorithm called trust region policy optimization (TRPO). This algorithm is shown to yield strong empirical results in two domains: simulated robotic locomotion, and Atari games using images as input. TRPO is closely related to natural gradient methods, [Kak02; BS03; PS08]; however, there are some changes introduced, which make the algorithm more scalable and robust. Furthermore, the derivation of TRPO motivates a new class of policy gradient methods that controls the size of the policy update but doesn't necessarily use the natural gradient step direction. *This work was previously published as [Sch+15c].*

Policy gradient methods, including TRPO, often require a large number of samples to learn. They work by trying to determine which actions were *good*, and then increasing the probability of the good actions. Determining which actions were good is called the *credit assignment problem* (e.g., see [SB98])—when the agent receives a reward, we need to determine which preceding actions deserve credit for it and should be reinforced. The next line of work described in this thesis analyzes this credit assignment problem, and how we can reduce the variance of policy gradient estimation through the use of value functions. By combining the proposed technique, which we call *generalized advantage estimation*, with TRPO, we are able to obtain state-of-the-art results on simulated 3D robotic tasks. 3D locomotion has been considered to be a challenging problem for all methods for a long time; yet our method is able to automatically obtain stable walking controllers for a 3D humanoid and quadruped, as well as a policy that enables a 3D humanoid to stand up off the ground—all using the same algorithm and hyperparameters. *This work was previously published as [Sch+15b]*

When optimizing stochastic policies, the reinforcement learning problem turns into a problem of optimizing an expectation, defined on a stochastic process with many sampled random variables. Problems with similar structure occur in problems outside of reinforcement learning; for example, in variational inference, and in models that use "hard decisions" for memory and attention. The last contribution of this thesis is the formalism of *stochastic computation graphs*, which are aimed to unify reinforcement learning and these other problems that involve optimizing expectations. Stochastic computation graphs allow one to automatically derive gradient estimators and variance-reduction schemes for a variety of different objectives that have been used in reinforcement learning and probabilistic modeling, reproducing the special-purpose estimators that were previously derived for these objectives. The formalism of stochastic computation graphs could assist researchers in developing intricate models involving a combination of stochastic and deterministic operations, enabling, for example, attention, memory, and control actions—and also in creating software that automatically computes these gradients given a model definition, as with automatic differentiation software. *This work was previously published as [Sch+15a]*.

2

BACKGROUND

2.1 MARKOV DECISION PROCESSES

A Markov Decision Process (MDP) is a mathematical object that describes an agent interacting with a stochastic environment. It is defined by the following components:

- $\mathcal{S}$: **state space**, a set of states of the environment.
- $\mathcal{A}$: **action space**, a set of actions, which the agent selects from at each timestep.
- $P(r, s' \mid s, a)$: a transition probability distribution. For each state s and action a, P specifies the probability that the environment will emit reward r and transition to state s'.

In certain problem settings, we will also be concerned with an **initial state distribution** $\mu(s)$, which is the probability distribution that the initial state s_0 is sampled from.

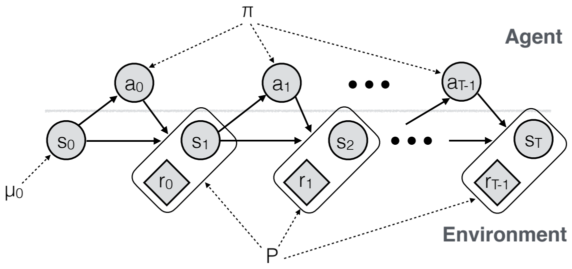
Various different definitions of MDP are used throughout the literature. Sometimes, the reward is defined as a deterministic function $R(s)$, $R(s, a)$, or $R(s, a, s')$. These formulations are equivalent in expressive power. That is, given a deterministic-reward formulation, we can simulate a stochastic reward by lumping the reward into the state.

The end goal is to find a **policy** π, which maps states to actions. We will mostly consider stochastic policies, which are conditional distributions $\pi(a \mid s)$, though elsewhere in the literature, one frequently sees deterministic policies $a = \pi(s)$.

2.2 THE EPISODIC REINFORCEMENT LEARNING PROBLEM

This thesis will be focused on the episodic setting of reinforcement learning, where the agent's experience is broken up into a series of **episodes**—sequences with a finite number of states, actions and rewards. Episodic reinforcement learning in the fully-observed setting is defined by the following process. Each episode begins by sampling an initial state of the environment, s_0, from distribution $\mu(s_0)$. Each timestep $t = 0, 1, 2, \ldots$, the agent chooses an action a_t, sampled from distribution $\pi(a_t \,|\, s_t)$. π is called the *policy*—it's the probability distribution that the agent uses to sample its actions. Then the enviroment generates the next state and reward, according to some distribution $P(s_{t+1}, r_t \,|\, s_t, a_t)$. The episode ends when a *terminal state* s_T is reached. This process can be described by the following equations or diagram below.

$$\begin{aligned} s_0 &\sim \mu(s_0) \\ a_0 &\sim \pi(a_0 \,|\, s_0) \\ s_1, r_0 &\sim P(s_1, r_0 \,|\, s_0, a_0) \\ a_1 &\sim \pi(a_1 \,|\, s_1) \\ s_2, r_1 &\sim P(s_2, r_1 \,|\, s_1, a_1) \\ &\dots \\ a_{T-1} &\sim \pi(a_{T-1} \,|\, s_{T-1}) \\ s_T, r_{T-1} &\sim P(s_T \,|\, s_{T-1}, a_{T-1}) \end{aligned}$$



The goal is to find a policy π that optimizes the expected total reward per episode.

$$\begin{aligned} &\underset{\pi}{\text{maximize}}\; E_\tau[R \,|\, \pi] \\ &\text{where } R = r_0 + r_1 + \dots + r_{\text{length}(\tau)-1} \end{aligned}$$

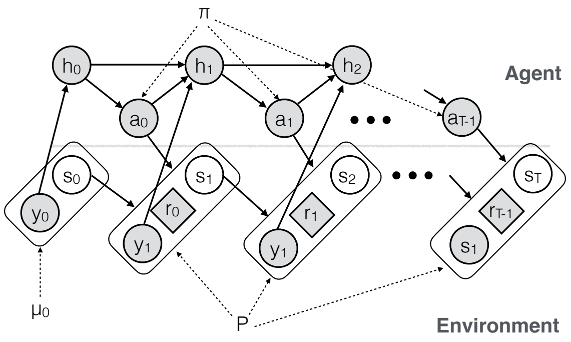
The expectation is taken over trajectories τ, defined as the sequence of states, actions, and rewards, $(s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T)$, ending in a terminal state. These trajectories are sampled using policy π to generate actions.

Note: expectation notation. $E_x[f(x) \,|\, z]$ is defined to mean $E_x[f(x) \,|\, z] = \int dx\, p(x|z) f(x)$. In words, the subscript is the random variable we are averaging over, and the conditioning expression (z) is a variable that affects the distribution over x. In a slight abuse of notation, we'll place functions like the policy π in this conditioning expression, i.e., $E_\tau[f(\tau) \,|\, \pi] = \int d\tau\, p_\pi(\tau) f(\tau)$, where $p_\pi(\tau)$ is the probability distribution of trajectories obtained by executing policy π.

2.3 PARTIALLY OBSERVED PROBLEMS

In the *partially-observed setting*, the agent only has access to an observation at each timestep, which may give noisy and incomplete information about the state. The agent should combine information from many previous timesteps, so the action a_t depends on the preceding history $h_t = (y_0, a_0, y_1, a_1, \ldots, y_{t-1}, a_{t-1}, y_t)$. The data-generating process is given by the following equations, and the figure below.

$$\begin{aligned}
s_0, y_0 &\sim \mu_0 \\
a_0 &\sim \pi(a_0 \mid h_0) \\
s_1, y_1, r_0 &\sim P(s_1, y_1, r_0 \mid s_0, a_0) \\
a_1 &\sim \pi(a_1 \mid h_1) \\
s_2, y_2, r_1 &\sim P(s_2, y_2, r_1 \mid s_1, a_1) \\
&\ldots \\
a_{T-1} &\sim \pi(a_{T-1} \mid h_{T-1}) \\
s_T, y_T, r_{T-1} &\sim P(s_T, y_T, r_{T-1} \mid s_{T-1}, a_{T-1})
\end{aligned}$$



This process is called a partially observed Markov decision process (POMDP). The partially-observed setting is equivalent to the fully-observed setting because we can call the observation history h_t the state of the system. That is, *a POMDP can be written as an MDP* (with infinite state space). When using function approximation, the partially observed setting is not much different conceptually from the fully-observed setting.

2.4 policies

We'll typically use *parameterized stochastic policies*, which we'll write as $\pi_\theta(a \mid s)$. Here, $\theta \in \mathbb{R}^d$ is a parameter vector that specifies the policy. For example, if the policy is a neural network, θ would correspond to the flattened weights and biases of the network. The parameterization of the policy will depend on the action space of the MDP, and whether it is a discrete set or a continuous space. The following are sensible choices (but not the only choices) for how to how to define deterministic and stochastic neural network policies. With a discrete action space, we'll use a neural network that outputs action probabilities, i.e., the final layer is a softmax layer. With a continuous action space, we'll use a neural network that outputs the mean of a Gaussian distribution, with a separate set of parameters specifying a diagonal covariance matrix. Since the optimal policy in an MDP or POMDP is deterministic, we don't lose much by using a simple action distribution (e.g., a diagonal covariance matrix, rather than a full covariance matrix or a more complicated multi-model distribution.)

2.5 DERIVIATIVE FREE OPTIMIZATION OF POLICIES

Recall from the previous chapter that episodic reinforcement learning can be viewed as the following optimization problem:

$$\underset{\pi}{\text{maximize}}\, E[R \mid \pi]$$

where R is the total reward of an episode. If we choose a parameterized model π_θ for the policies, then this becomes an optimization problem with respect to $\theta \in \mathbb{R}^d$.

$$\underset{\theta}{\text{maximize}}\, E[R \mid \pi_\theta]$$

In derivative-free optimization, we treat the whole process for turning a parameter θ into a reward R as a black box, which gives us noisy evaluations $\theta \to \blacksquare \to R$, but we know nothing about what's inside the box.

A thorough discussion of derivative-free optimization algorithms is beyond the scope of this thesis. However, we'll introduce one algorithm, which is applicable in the noisy black-box optimization setting, and is used in comparisons later. Cross entropy method (CEM) is a simple but effective evolutionary algorithm, which works with Gaussian distributions, repeatedly updating the mean and variance of a distribution over candidate parameters. A simple instantiation is as follows.

Algorithm 1 Cross Entropy Method

```
Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d$
for iteration = $1, 2, \ldots$ do
    Collect n samples of $\theta_i \sim N(\mu, \mathrm{diag}(\sigma))$
    Perform one episode with each $\theta_i$, obtaining reward $R_i$
    Select the top p% of samples (e.g. $p = 20$), which we'll call the elite set
    Fit a Gaussian distribution, with diagonal covariance, to the elite set, obtaining a new $\mu, \sigma$.
end for
Return the final $\mu$.
```

Algorithm 1 is prone to reducing the variance too quickly and converging to a bad local optimum. It can be improved by artificially adding extra variance, according to a schedule where this added noise decreases to zero. Details of this technique can be found in [SL06].

2.6 POLICY GRADIENTS

Policy gradient methods are a class of reinforcement learning algorithms that work by repeatedly estimating the gradient of the policy's performance with respect to its parameters. The simplest way to derive them is to use the *score function gradient estimator*, a general method for estimating gradients of expectations. Suppose that x is a random variable with probability density $p(x \mid \theta)$, f is a scalar-valued function (say, the reward), and we are interested in computing $\nabla_\theta E_x[f(x)]$. Then we have the following equality:

$$\nabla_\theta E_x[f(x)] = E_x[\nabla_\theta \log p(x \mid \theta) f(x)].$$

This equation can be derived by writing the expectation as an integral:

$$\begin{aligned}\nabla_\theta E_x[f(x)] &= \nabla_\theta \int dx\ p(x \mid \theta) f(x) = \int dx\ \nabla_\theta p(x \mid \theta) f(x) \\ &= \int dx\ p(x \mid \theta) \nabla_\theta \log p(x \mid \theta) f(x) = E_x[f(x) \nabla_\theta \log p(x \mid \theta)].\end{aligned}$$

To use this estimator, we can sample values $x \sim p(x \mid \theta)$, and compute the LHS of the equation above (averaged over N samples) to get an estimate of the gradient (which becomes increasingly accurate as $N \to \infty$. That is, we take $x_1, x_2, \dots, x_N \sim p(x \mid \theta)$, and then take our gradient estimate $\hat{g}$ to be

$$\hat{g} = \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta \log p(x_i \mid \theta) f(x_i)$$

To use this idea in reinforcement learning, we will need to use a *stochastic policy*. That means that at each state s, our policy gives us a probability distribution over actions, which will be denoted $\pi(a \mid s)$. Since the policy also has a parameter vector θ, we'll write $\pi_\theta(a \mid s)$ or $\pi(a \mid s, \theta)$.

In the following discussion, a *trajectory* τ will refer to a sequence of states and actions $\tau \equiv (s_0, a_0, s_1, a_1, \dots, s_T)$. Let $p(\tau \mid \theta)$ denote the probability of the entire trajectory τ under policy parameters θ, and let $R(\tau)$ denote the total reward of the trajectory.

The derivation of the score function gradient estimato tells us that

$$\nabla_\theta E_\tau[R(\tau)] = E_\tau[\nabla_\theta \log p(\tau \mid \theta) R(\tau)]$$

Next, we need to expand the quantity $\log p(\tau \mid \theta)$ to derive a practical formula. Using the chain rule of probabilities, we obtain

$$\begin{aligned}p(\tau \mid \theta) =& \mu(s_0) \pi(a_0 \mid s_0, \theta) P(s_1, r_0 \mid s_0, a_0) \pi(a_1 \mid s_1, \theta) \\ & P(s_2, r_1 \mid s_1, a_1) \dots \pi(a_{T-1} \mid s_{T-1}, \theta) P(s_T, r_{T-1} \mid s_{T-1}, a_{T-1}),\end{aligned}$$

where μ is the initial state distribution. When we take the logarithm, the product turns into a sum, and when we differentiate with respect to θ, the terms $P(s_t \mid s_{t-1}, a_{t-1})$ terms drop out as does $\mu(s_0)$. We obtain

$$\nabla_\theta E_\tau[R(\tau)] = E_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) R(\tau) \right]$$

It is somewhat remarkable that we are able to compute the policy gradient without knowing anything about the system dynamics, which are encoded in transition probabilities P. The intuitive interpretation is that we collect a trajectory, and then *increase its log-probability proportionally to its goodness*. That is, if the reward $R(\tau)$ is very high, we ought to move in the the direction in parameter space that increases $\log p(\tau \mid \theta)$.

Note: trajectory lengths and time-dependence. Here, we are considering trajectories with fixed length T, whereas the definition of MDPs and POMDPs above assumed variable or infinite length, and *stationary* (time-independent) dynamics. The derivations in policy gradient methods are much easier to analyze with fixed length trajectories—otherwise we end up with infinite sums. The fixed-length case can be made to mostly subsume the variable-length case, by making T very large, and instead of trajectories ending, the system goes into a *sink* state with zero reward. As a result of using finite-length trajectories, certain quantities become time-dependent, because the problem is no longer stationary. However, we can include time in the state so that we don't need to separately account for the dependence on time. Thus, we will omit the time-dependence of various quantities below, such as the state-value function V^π.

We can derive versions of this formula that eliminate terms to reduce variance. This calculation is provided in much more generality in Chapter 5 on *stochastic computation graphs*, but we'll include it here because the concrete setting of this chapter will be easier to understand.

First, we can apply the above argument to compute the gradient for a single reward term:

$$\nabla_\theta E_\tau[r_t] = E_\tau \left[\sum_{t'=0}^{t} \nabla_\theta \log \pi(a_{t'} \mid s_{t'}, \theta) r_t \right]$$

Note that the sum goes up to t, because the expectation over r_t can be written in terms of actions $a_{t'}$ with $t' \leqslant t$. Summing over time (taking $\sum_{t=0}^{T-1}$ of the above equation), we get

$$\begin{aligned}\nabla_\theta E_\tau[R(\tau)] &= E_\tau\left[\sum_{t=0}^{T-1} r_t \sum_{t'=0}^{t} \nabla_\theta \log \pi(a_{t'} \mid s_{t'}, \theta)\right] \\ &= E_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \sum_{t'=t}^{T-1} r_{t'}\right]. \qquad (1)\end{aligned}$$

The second formula (Equation (1)) results from the first formula by reordering the summation. We will mostly work with the second formula, as it is more convenient for numerical implementation.

We can further reduce the variance of the policy gradient estimator by using a *baseline*: that is, we subtract a function $b(s_t)$ from the empirical returns, giving us the following formula for the policy gradient:

$$\nabla_\theta \mathbb{E}_\tau\left[R(\tau)\right] = \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta)\left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t)\right)\right] \qquad (2)$$

This equality holds for arbitrary baseline functions b. To derive it, we'll show that the added terms $b(s_t)$ have no effect on the expectation, i.e., that $\mathbb{E}_\tau\left[\nabla_\theta \log \pi(a_{t'} \mid s_{t'}, \theta) b(s_t)\right] = 0$. To show this, split up the expectation over whole trajectories $\mathbb{E}_\tau\left[\dots\right]$ into an expectation over all variables before a_t, and all variables after and including it.

$$\begin{aligned}&\mathbb{E}_\tau\left[\nabla_\theta \log \pi(a_t \mid s_t, \theta) b(s_t)\right] \\ &= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[\mathbb{E}_{s_{(t+1):T}, a_{t:(T-1)}}\left[\nabla_\theta \log \pi(a_t \mid s_t, \theta) b(s_t)\right]\right] \qquad \text{(break up expectation)} \\ &= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[b(s_t)\mathbb{E}_{s_{(t+1):T}, a_{t:(T-1)}}\left[\nabla_\theta \log \pi(a_t \mid s_t, \theta)\right]\right] \qquad \text{(pull baseline term out)} \\ &= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[b(s_t)\mathbb{E}_{a_t}\left[\nabla_\theta \log \pi(a_t \mid s_t, \theta)\right]\right] \qquad \text{(remove irrelevant vars.)} \\ &= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[b(s_t) \cdot 0\right]\end{aligned}$$

The last equation follows because $\mathbb{E}_{a_t}\left[\nabla_\theta \log \pi(a_{t'} \mid s_{t'}, \theta)\right] = \nabla_\theta \mathbb{E}_{a_t}\left[1\right] = 0$ by the definition of the score function gradient estimator.

A near-optimal choice of baseline is the state-value function,

$$V^\pi(s) = \mathbb{E}\left[r_t + r_{t+1} + \dots + r_{T-1} \mid s_t = s,\ a_{t:(T-1)} \sim \pi\right]$$

See [GBB04] for a discussion of the choice of baseline that optimally reduces variance of the policy gradient estimator. So in practice, we will generally choose the baseline to approximate the value function, $b(s) \approx V^\pi(s)$.

We can intuitively justify the choice $b(s) \approx V^{\pi}(s)$ as follows. Suppose we collect a trajectory and compute a noisy gradient estimate

$$\hat{g} = \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi(a_t \mid s_t, \theta) \sum_{t'=t}^{T-1} r_{t'}$$

which we will use to update our policy $\theta \to \theta + \epsilon \hat{g}$. This update increases the log-probability of a_t proportionally to the sum of rewards $r_t + r_{t+1} + \cdots + r_{T-1}$, following that action. In otherwords, if the sum of rewards is high, then the action was probably good, so we increase its probability. To get a better estimate of whether the action was good, we should check to see if the returns were *better than expected*. Before taking the action, the expected returns were $V^{\pi}(s_t)$. Thus, the difference $\sum_{t'=t}^{T-1} r_{t'} - b(s_t)$ is an approximate estimate of the goodness of action a_t—Chapter 4 discusses in a more precise way how it is an estimate of the *advantage function*. Including the baseline in our policy gradient estimator, we get

$$\hat{g} = \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi(a_t \mid s_t, \theta) \left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right),$$

which increases the probability of the actions that we infer to be good—meaning that the estimated advantage $\hat{A}_t = \sum_{t'=t}^{T-1} r_{t'} - b(s_t)$ is positive.

If the trajectories are very long (i.e., T is high), then the preceding formula will have excessive variance. Thus, practitioners generally use a discount factor, which reduces variance at the cost of some bias. The following expression gives a biased estimator of the policy gradient.

$$\hat{g} = \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \left(\sum_{t'=t}^{T-1} r_{t'} \gamma^{t'-t} - b(s_t) \right)$$

To reduce variance in this biased estimator, we should choose $b(s_t)$ to optimally estimate the *discounted sum of rewards*,

$$b(s) \approx V^{\pi,\gamma}(s) = \mathbb{E}\left[\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} \;\middle|\; s_t = s; a_{t:(T-1) \sim \pi} \right]$$

Intuitively, the discount makes us pretend that the action a_t has no effect on the reward $r_{t'}$ for t' sufficiently far in the future, i.e., we are downweighting delayed effects by a factor of $\gamma^{t'-t}$. By adding up a series with coefficients $1, \gamma, \gamma^2, \dots$, we are effectively including $1/(1-\gamma)$ timesteps in the sum.

The policy gradient formulas given above can be used in a practical algorithm for optimizing policies.

Algorithm 2 "Vanilla" policy gradient algorithm

```
Initialize policy parameter $\theta$, baseline $b$
for iteration=$1, 2, \dots$ do
    Collect a set of trajectories by executing the current policy
    At each timestep in each trajectory, compute
      the return $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
      the advantage estimate $\hat{A}_t = R_t - b(s_t)$.
    Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
      summed over all trajectories and timesteps.
    Update the policy, using a policy gradient estimate $\hat{g}$,
      which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$
end for
```

In the algorithm above, the policy update can be performed with stochastic gradient ascent, $\theta \to \theta + \epsilon\hat{g}$, or one can use a more sophisticated method such as Adam [KB14].

To numerically compute the policy gradient estimate using automatic differentiation software, we swap the sum with the expectation in the policy gradient estimator:

$$\begin{aligned}\hat{g} &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T-1} r_{t'} \gamma^{t'-t} - b(s_t) \right) \\ &= \nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) \hat{A}_t\end{aligned}$$

Hence, one can construct the scalar quantity $\sum_t \log \pi_\theta(a_t|s_t)\hat{A}_t$ and differentiate it to obtain the policy gradient.

The vanilla policy gradient method described above has been well-known for a long time; some early papers include [Wil92; Sut+99; JJS94]. It was considered to be a poor choice on most problems because of its high sample complexity. A couple of other practical difficulties are that (1) it is hard to choose a stepsize that works for the entire course of the optimization, especially because the statistics of the states and rewards changes;

(2) often the policy prematurely converges to a nearly-deterministic policy with a suboptimal behavior. Simple methods to prevent this issue, such as adding an entropy bonus, usually fail.

The next two chapters in this thesis improve on the vanilla policy gradient method in two orthogonal ways, enabling us to obtain strong empirical results. Chapter 3 shows that instead of stepping in the gradient direction, we should move in the natural gradient direction, and that there is an effective way to choose stepsizes for reliable monotonic improvement. Chapter 4 provides much more detailed analysis of discounts, and Chapter 5 also revisits some of the variance reduction ideas we have just described, but in a more general setting. Concurrently with this thesis work, Mnih et al. [Mni+16] have shown that it is in fact possible to obtain state-of-the-art performance on various large-scale control tasks with the vanilla policy gradient method, however, the number of samples used for learning is extremely large.

3

TRUST REGION POLICY OPTIMIZATION

3.1 OVERVIEW

This chapter studies how to develop policy optimization methods that lead to monotonically improving performance and make efficient use of data. As we argued in the Introduction, in order to optimize function approximators, we need to reduce the reinforcement learning problem to a series of optimization problems. This reduction is nontrivial in reinforcement learning because the state distribution depends on the policy. This chapter shows that to update the policy, we should improve a certain surrogate objective as much as possible, while changing the policy as little as possible, where this change is measured as a KL divergence between action distributions. We show that by bounding the size of the policy update, we can bound the change in state distributions, guaranteeing policy improvement despite non-trivial step sizes.

Following this theoretical analysis, we make a series of approximations to the theoretically-justified algorithm, yielding a practical algorithm that we call trust region policy optimization (TRPO). We describe two variants of this algorithm: first, the *single-path* method, which can be applied in the model-free setting; second, the *vine* method, which requires the system to be restored to particular states, which is typically only possible in simulation. These algorithms are scalable and can optimize nonlinear policies with tens of thousands of parameters, which have previously posed a major challenge for model-free policy search [DNP13]. In our experiments, we show that the same TRPO methods can learn complex policies for swimming, hopping, and walking, as well as playing Atari games directly from raw images.

3.2 PRELIMINARIES

Consider an infinite-horizon discounted Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability distribution, $r : \mathcal{S} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}$ is the distribution of the initial state s_0, and $\gamma \in (0, 1)$ is the discount factor. Note that this setup differs from the Chapter 2 due to the discount, which is necessary for the theoretical analysis.

Let π denote a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, and let $\eta(\pi)$ denote its expected discounted reward:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right], \text{ where}$$
$$s_0 \sim \rho_0(s_0),\ a_t \sim \pi(a_t \mid s_t),\ s_{t+1} \sim P(s_{t+1} \mid s_t, a_t).$$

We will use the following standard definitions of the state-action value function Q^π, the value function V^π, and the advantage function A^π:

$$\begin{aligned}
Q^\pi(s_t, a_t) &= \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right], \\
V^\pi(s_t) &= \mathbb{E}_{a_t, s_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right], \\
A_\pi(s, a) &= Q_\pi(s, a) - V_\pi(s), \text{ where} \\
&\quad a_t \sim \pi(a_t \mid s_t), s_{t+1} \sim P(s_{t+1} \mid s_t, a_t) \text{ for } t \geqslant 0.
\end{aligned}$$

The following useful identity expresses the expected return of another policy $\tilde{\pi}$ in terms of the advantage over π, accumulated over timesteps (see Kakade and Langford [KL02] or Appendix 3.10 for proof):

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \cdots \sim \tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)\right] \tag{3}$$

where the notation $\mathbb{E}_{s_0, a_0, \cdots \sim \tilde{\pi}}[\ldots]$ indicates that actions are sampled $a_t \sim \tilde{\pi}(\cdot \mid s_t)$. Let ρ_π be the (unnormalized) discounted visitation frequencies

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \ldots,$$

where $s_0 \sim \rho_0$ and the actions are chosen according to π. We can rewrite Equation (3) with a sum over states instead of timesteps:

$$\begin{aligned}\eta(\tilde{\pi}) &= \eta(\pi) + \sum_{t=0}^{\infty} \sum_{s} P(s_t = s \mid \tilde{\pi}) \sum_{a} \tilde{\pi}(a \mid s) \gamma^t A_\pi(s,a) \\ &= \eta(\pi) + \sum_{s} \sum_{t=0}^{\infty} \gamma^t P(s_t = s \mid \tilde{\pi}) \sum_{a} \tilde{\pi}(a \mid s) A_\pi(s,a) \\ &= \eta(\pi) + \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \tilde{\pi}(a \mid s) A^{\pi}(s,a). \end{aligned} \tag{4}$$

This equation implies that any policy update $\pi \to \tilde{\pi}$ that has a nonnegative expected advantage at *every* state s, i.e., $\sum_a \tilde{\pi}(a \mid s) A_\pi(s,a) \geqslant 0$, is guaranteed to increase the policy performance η, or leave it constant in the case that the expected advantage is zero everywhere. This implies the classic result that the update performed by exact policy iteration, which uses the deterministic policy $\tilde{\pi}(s) = \arg\max_a A^{\pi}(s,a)$, improves the policy if there is at least one state-action pair with a positive advantage value and nonzero state visitation probability, otherwise the algorithm has converged to the optimal policy. However, in the approximate setting, it will typically be unavoidable, due to estimation and approximation error, that there will be some states s for which the expected advantage is negative, that is, $\sum_a \tilde{\pi}(a \mid s) A_\pi(s,a) < 0$. The complex dependency of $\rho_{\tilde{\pi}}(s)$ on $\tilde{\pi}$ makes Equation (4) difficult to optimize directly. Instead, we introduce the following local approximation to η:

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_{s} \rho_\pi(s) \sum_{a} \tilde{\pi}(a \mid s) A^{\pi}(s,a). \tag{5}$$

Note that L_π uses the visitation frequency ρ_π rather than $\rho_{\tilde{\pi}}$, ignoring changes in state visitation density due to changes in the policy. However, if we have a parameterized policy π_θ, where $\pi_\theta(a \mid s)$ is a differentiable function of the parameter vector θ, then L_π matches η to first order (see Kakade and Langford [KL02]). That is, for any parameter value θ_0,

$$\begin{aligned} L_{\pi_{\theta_0}}(\pi_{\theta_0}) &= \eta(\pi_{\theta_0}), \\ \nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)\big|_{\theta=\theta_0} &= \nabla_\theta \eta(\pi_\theta)\big|_{\theta=\theta_0}. \end{aligned} \tag{6}$$

Equation (6) implies that a sufficiently small step $\pi_{\theta_0} \to \tilde{\pi}$ that improves $L_{\pi_{\theta_{\mathrm{old}}}}$ will also improve η, but does not give us any guidance on how big of a step to take.

To address this issue, Kakade and Langford [KL02] proposed a policy updating scheme called conservative policy iteration, for which they could provide explicit lower bounds on the improvement of η. To define the conservative policy iteration update, let π_{old} denote the current policy, and let $\pi' = \arg\min_{\pi'} L_{\pi_{\text{old}}}(\pi')$. The new policy π_{new} was defined to be the following mixture:

$$\pi_{new}(a \mid s) = (1-\alpha)\pi_{\text{old}}(a \mid s) + \alpha\pi'(a \mid s). \tag{7}$$

Kakade and Langford proved the following result for this update:

$$\begin{gathered}\eta(\pi_{new}) \geqslant L_{\pi_{\text{old}}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma(1-\alpha))(1-\gamma)}\alpha^2, \\ \text{where } \epsilon = \max_s |\mathbb{E}_{a\sim\pi'(a \mid s)}\left[A^{\pi}(s,a)\right]|\end{gathered} \tag{8}$$

Since $\alpha, \gamma \in [0,1]$, Equation (8) implies the following simpler bound, which we refer to in the next section:

$$\eta(\pi_{new}) \geqslant L_{\pi_{\text{old}}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2. \tag{9}$$

The simpler bound is only slightly weaker when $\alpha \ll 1$, which is typically the case in the conservative policy iteration method of Kakade and Langford [KL02]. Note, however, that so far this bound only applies to mixture policies generated by Equation (7). This policy class is unwieldy and restrictive in practice, and it is desirable for a practical policy update scheme to be applicable to all general stochastic policy classes.

3.3 MONOTONIC IMPROVEMENT GUARANTEE FOR GENERAL STOCHASTIC POLICIES

Equation (8), which applies to conservative policy iteration, implies that a policy update that improves the right-hand side is guaranteed to improve the true performance η. Our principal theoretical result is that the policy improvement bound in Equation (8) can be extended to general stochastic policies, rather than just mixture polices, by replacing α with a distance measure between π and $\tilde{\pi}$. Since mixture policies are rarely used in practice, this result is crucial for extending the improvement guarantee to practical problems. The particular distance measure we use is the total variation divergence, which is defined by $D_{TV}(p \parallel q) = \frac{1}{2}\sum_i |p_i - q_i|$ for discrete probability distributions p, q.[1] Define

[1] Our result is straightforward to extend to continuous states and actions by replacing the sums with integrals.

$D_{\mathrm{TV}}^{\max}(\pi, \tilde{\pi})$ as

$$D_{\mathrm{TV}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{\mathrm{TV}}(\pi(\cdot \mid s) \parallel \tilde{\pi}(\cdot \mid s)). \tag{10}$$

Proposition 1. *Let $\alpha = D_{\mathrm{TV}}^{\max}(\pi_{\mathrm{old}}, \pi_{\mathrm{new}})$. Then Equation (9) holds.*

We provide two proofs in the appendix. The first proof extends Kakade and Langford's result using the fact that the random variables from two distributions with total variation divergence less than α can be coupled, so that they are equal with probability $1-\alpha$. The second proof uses perturbation theory to prove a slightly stronger version of Equation (9), with a more favorable definition of ϵ that depends on $\tilde{\pi}$.

Next, we note the following relationship between the total variation divergence and the KL divergence (Pollard [Pol00], Ch. 3): $D_{\mathrm{TV}}(p \parallel q)^2 \leqslant D_{\mathrm{KL}}(p \parallel q)$. Let $D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{\mathrm{KL}}(\pi(\cdot \mid s) \parallel \tilde{\pi}(\cdot \mid s))$. The following bound then follows directly from Equation (9):

$$\begin{gathered}\eta(\tilde{\pi}) \geqslant L_\pi(\tilde{\pi}) - C D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}), \\ \text{where } C = \frac{2\epsilon\gamma}{(1-\gamma)^2}.\end{gathered} \tag{11}$$

Algorithm 3 describes an approximate policy iteration scheme based on the policy improvement bound in Equation (11). Note that for now, we assume exact evaluation of the advantage values A^π. Algorithm 3 uses a constant $\epsilon' \leqslant \epsilon$ that is simpler to describe in terms of measurable quantities.

It follows from Equation (11) that Algorithm 3 is guaranteed to generate a monotonically improving sequence of policies $\eta(\pi_0) \leqslant \eta(\pi_1) \leqslant \eta(\pi_2) \leqslant \dots$. To see this, let $M_i(\pi) = L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi)$. Then

$$\begin{aligned}&\eta(\pi_{i+1}) \geqslant M_i(\pi_{i+1}) \text{ by Equation (11)}\\ &\eta(\pi_i) = M_i(\pi_i), \text{ therefore,}\\ &\eta(\pi_{i+1}) - \eta(\pi_i) \geqslant M_i(\pi_{i+1}) - M(\pi_i).\end{aligned} \tag{12}$$

Thus, by maximizing M_i at each iteration, we guarantee that the true objective η is non-decreasing. This algorithm is a type of minorization-maximization (MM) algorithm [HL04], which is a class of methods that also includes expectation maximization. In the terminology of MM algorithms, M_i is the surrogate function that minorizes η with

Algorithm 3 Approximate policy iteration algorithm guaranteeing non-decreasing expected return η

```
Initialize $\pi_0$.
for $i = 0, 1, 2, \ldots$ until convergence do
    Compute all advantage values $A_{\pi_i}(s, a)$.
    Solve the constrained optimization problem
        $\pi_{i+1} = \arg\max_{\pi} [L_{\pi_i}(\pi) - (\frac{2\epsilon'\gamma}{(1-\gamma)^2}) D_{KL}^{\max}(\pi_i, \pi)]$
            where $\epsilon' = \max_s \max_a |A^{\pi}(s, a)|$
            and $L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a \mid s) A_{\pi_i}(s, a)$
end for
```

equality at π_i. This algorithm is also reminiscent of proximal gradient methods and mirror descent.

Trust region policy optimization, which we propose in the following section, is an approximation to Algorithm 3, which uses a constraint on the KL divergence rather than a penalty to robustly allow large updates.

3.4 optimization of parameterized policies

In the previous section, we considered the policy optimization problem independently of the parameterization of π and under the assumption that the policy can be evaluated at all states. We now describe how to derive a practical algorithm from these theoretical foundations, under finite sample counts and arbitrary parameterizations.

Since we consider parameterized policies $\pi_\theta(a \mid s)$ with parameter vector θ, we will overload our previous notation to use functions of θ rather than π, e.g. $\eta(\theta) := \eta(\pi_\theta)$, $L_\theta(\tilde{\theta}) := L_{\pi_\theta}(\pi_{\tilde{\theta}})$, and $D_{KL}(\theta \parallel \tilde{\theta}) := D_{KL}(\pi_\theta \parallel \pi_{\tilde{\theta}})$. We will use θ_{old} to denote the previous policy parameters that we want to improve upon.

The preceding section showed that $\eta(\theta) \geqslant L_{\theta_{old}}(\theta) - C D_{KL}^{\max}(\theta_{old}, \theta)$, with equality at $\theta = \theta_{old}$. Thus, by performing the following maximization, we are guaranteed to improve the true objective η:

$$\underset{\theta}{\text{maximize}} [L_{\theta_{old}}(\theta) - C D_{KL}^{\max}(\theta_{old}, \theta)].$$

In practice, if we used the penalty coefficient C recommended by the theory above, the step sizes would be very small. One way to take larger steps in a robust way is to use a constraint on the KL divergence between the new policy and the old policy, i.e., a trust region constraint:

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta) \\ &\text{subject to } D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \leqslant \delta. \end{aligned} \tag{13}$$

This problem imposes a constraint that the KL divergence is bounded at every point in the state space. While it is motivated by the theory, this problem is impractical to solve due to the large number of constraints. Instead, we can use a heuristic approximation which considers the average KL divergence:

$$\overline{D}_{\text{KL}}^{\rho}(\theta_1, \theta_2) := \mathbb{E}_{s\sim\rho}\left[D_{\text{KL}}(\pi_{\theta_1}(\cdot \mid s) \parallel \pi_{\theta_2}(\cdot \mid s))\right].$$

We therefore propose solving the following optimization problem to generate a policy update:

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta) \\ &\text{subject to } \overline{D}_{\text{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leqslant \delta. \end{aligned} \tag{14}$$

Similar policy updates have been proposed in prior work [BS03; PS08; PMA10], and we compare our approach to prior methods in Section 3.7 and in the experiments in Section 3.8. Our experiments also show that this type of constrained update has similar empirical performance to the maximum KL divergence constraint in Equation (13).

3.5 SAMPLE-BASED ESTIMATION OF THE OBJECTIVE AND CONSTRAINT

The previous section proposed a constrained optimization problem on the policy parameters (Equation (14)), which optimizes an estimate of the expected total reward η subject to a constraint on the change in the policy at each update. This section describes how the objective and constraint functions can be approximated using Monte Carlo simulation.

We seek to solve the following optimization problem, obtained by expanding $L_{\theta_{\text{old}}}$ in Equation (14):

$$\begin{aligned} &\underset{\theta}{\text{maximize}} \sum_s \rho_{\theta_{\text{old}}}(s) \sum_a \pi_\theta(a \mid s) A_{\theta_{\text{old}}}(s, a) \\ &\qquad\qquad \text{subject to } \overline{D}_{\text{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leqslant \delta. \end{aligned} \tag{15}$$

We first replace $\sum_s \rho_{\theta_{\text{old}}}(s)[\dots]$ in the objective by the expectation $\frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}[\dots]$. Next, we replace the advantage values $A_{\theta_{\text{old}}}$ by the Q-values $Q_{\theta_{\text{old}}}$ in Equation (15), which only changes the objective by a constant. Last, we replace the sum over the actions by an importance sampling estimator. Using q to denote the sampling distribution, the contribution of a single s_n to the loss function is

$$\sum_a \pi_\theta(a \mid s_n) A_{\theta_{\text{old}}}(s_n, a) = \mathbb{E}_{a\sim q}\left[\frac{\pi_\theta(a \mid s_n)}{q(a \mid s_n)} A_{\theta_{\text{old}}}(s_n, a)\right].$$

Our optimization problem in Equation (15) is exactly equivalent to the following one, written in terms of expectations:

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\, \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}, a\sim q}\left[\frac{\pi_\theta(a \mid s)}{q(a \mid s)} Q_{\theta_{\text{old}}}(s, a)\right] \\ &\text{subject to } \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot \mid s) \parallel \pi_\theta(\cdot \mid s))\right] \leqslant \delta. \end{aligned} \tag{16}$$

All that remains is to replace the expectations by sample averages and replace the Q value by an empirical estimate. The following sections describe two different schemes for performing this estimation.

The first sampling scheme, which we call *single path*, is the one that is typically used for policy gradient estimation [BB11], and is based on sampling individual trajectories. The second scheme, which we call *vine*, involves constructing a rollout set and then performing multiple actions from each state in the rollout set. This method has mostly been explored in the context of policy iteration methods [LP03; GGS13].

3.5.1 *Single Path*

In this estimation procedure, we collect a sequence of states by sampling $s_0 \sim \rho_0$ and then simulating the policy $\pi_{\theta_{\text{old}}}$ for some number of timesteps to generate a trajectory $s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, s_T$. Hence, $q(a \mid s) = \pi_{\theta_{\text{old}}}(a \mid s)$. $Q_{\theta_{\text{old}}}(s, a)$ is computed at each state-action pair (s_t, a_t) by taking the discounted sum of future rewards along the trajectory.

3.5.2 *Vine*

In this estimation procedure, we first sample $s_0 \sim \rho_0$ and simulate the policy π_{θ_i} to generate a number of trajectories. We then choose a subset of N states along these trajectories,

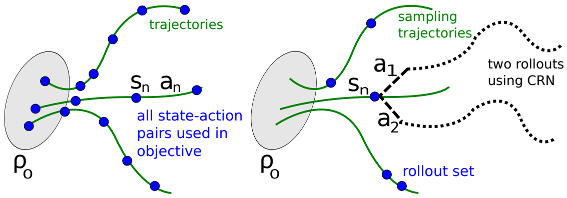


Figure 1: Left: illustration of single path procedure. Here, we generate a set of trajectories via simulation of the policy and incorporate all state-action pairs (s_n, a_n) into the objective. Right: illustration of vine procedure. We generate a set of "trunk" trajectories, and then generate "branch" rollouts from a subset of the reached states. For each of these states s_n, we perform multiple actions (a_1 and a_2 here) and perform a rollout after each action, using common random numbers (CRN) to reduce the variance.

denoted $s_1, s_2, \ldots, s_N$, which we call the "rollout set". For each state s_n in the rollout set, we sample K actions according to $a_{n,k} \sim q(\cdot \mid s_n)$. Any choice of $q(\cdot \mid s_n)$ with a support that includes the support of $\pi_{\theta_i}(\cdot \mid s_n)$ will produce a consistent estimator. In practice, we found that $q(\cdot \mid s_n) = \pi_{\theta_i}(\cdot \mid s_n)$ works well on continuous problems, such as robotic locomotion, while the uniform distribution works well on discrete tasks, such as the Atari games, where it can sometimes achieve better exploration.

For each action $a_{n,k}$ sampled at each state s_n, we estimate $\hat{Q}_{\theta_i}(s_n, a_{n,k})$ by performing a rollout (i.e., a short trajectory) starting with state s_n and action $a_{n,k}$. We can greatly reduce the variance of the Q-value differences between rollouts by using the same random number sequence for the noise in each of the K rollouts, i.e., *common random numbers*. See [Ber05] for additional discussion on Monte Carlo estimation of Q-values and [NJ00] for a discussion of common random numbers in reinforcement learning.

In small, finite action spaces, we can generate a rollout for every possible action from a given state. The contribution to $L_{\theta_{\text{old}}}$ from a single state s_n is as follows:

$$L_n(\theta) = \sum_{k=1}^{K} \pi_\theta(a_k \mid s_n) \hat{Q}(s_n, a_k),$$

where the action space is $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$. In large or continuous state spaces, we can construct an estimator of the surrogate objective using importance sampling. The self-normalized estimator (Owen [Owe13], Chapter 8) of $L_{\theta_{\text{old}}}$ obtained at a single state

s_n is

$$L_n(\theta) = \frac{\sum_{k=1}^{K} \frac{\pi_\theta(a_{n,k} \mid s_n)}{\pi_{\theta_{old}}(a_{n,k} \mid s_n)} \hat{Q}(s_n, a_{n,k})}{\sum_{k=1}^{K} \frac{\pi_\theta(a_{n,k} \mid s_n)}{\pi_{\theta_{old}}(a_{n,k} \mid s_n)}},$$

assuming that we performed K actions $a_{n,1}, a_{n,2}, \ldots, a_{n,K}$ from state s_n. This self-normalized estimator removes the need to use a baseline for the Q-values (note that the gradient is unchanged by adding a constant to the Q-values). Averaging over $s_n \sim \rho(\pi)$, we obtain an estimator for $L_{\theta_{old}}$, as well as its gradient.

The *vine* and *single path* methods are illustrated in Figure 1. We use the term *vine*, since the trajectories used for sampling can be likened to the stems of vines, which branch at various points (the rollout set) into several short offshoots (the rollout trajectories).

The benefit of the *vine* method over the *single path* method that is our local estimate of the objective has much lower variance given the same number of Q-value samples in the surrogate objective. That is, the *vine* method gives much better estimates of the advantage values. The downside of the *vine* method is that we must perform far more calls to the simulator for each of these advantage estimates. Furthermore, the *vine* method requires us to generate multiple trajectories from each state in the rollout set, which limits this algorithm to settings where the system can be reset to an arbitrary state. In contrast, the single path algorithm requires no state resets and can be directly implemented on a physical system [PS08].

3.6 practical algorithm

Here we present two practical policy optimization algorithm based on the ideas above, which use either the *single path* or *vine* sampling scheme from the preceding section. The algorithms repeatedly perform the following steps:

1. Use the *single path* or *vine* procedures to collect a set of state-action pairs along with Monte Carlo estimates of their Q-values.
2. By averaging over samples, construct the estimated objective and constraint in Equation (16).
3. Approximately solve this constrained optimization problem to update the policy's parameter vector θ. We use the conjugate gradient algorithm followed by a line search, which is altogether only slightly more expensive than computing the gradient itself. See Section 3.12 for details.

With regard to (3), we construct the Fisher information matrix (FIM) by analytically computing the Hessian of the KL divergence, rather than using the covariance matrix of the gradients. That is, we estimate A_{ij} as $\frac{1}{N}\sum_{n=1}^{N}\frac{\partial^2}{\partial\theta_i\partial\theta_j}D_{KL}(\pi_{\theta_{old}}(\cdot \mid s_n) \parallel \pi_\theta(\cdot \mid s_n))$, rather than $\frac{1}{N}\sum_{n=1}^{N}\frac{\partial}{\partial\theta_i}\log\pi_\theta(a_n \mid s_n)\frac{\partial}{\partial\theta_j}\log\pi_\theta(a_n \mid s_n)$. The analytic estimator integrates over the action at each state s_n, and does not depend on the action a_n that was sampled. As described in Section 3.12, this analytic estimator has computational benefits in the large-scale setting, since it removes the need to store a dense Hessian or all policy gradients from a batch of trajectories. The rate of improvement in the policy is similar to the empirical FIM, as shown in the experiments.

Let us briefly summarize the relationship between the theory from Section 3.3 and the practical algorithm we have described:

- The theory justifies optimizing a surrogate objective with a penalty on KL divergence. However, the large penalty coefficient $\frac{2\epsilon\gamma}{(2-\gamma)^2}$ leads to prohibitively small steps, so we would like to decrease this coefficient. Empirically, it is hard to robustly choose the penalty coefficient, so we use a hard constraint instead of a penalty, with parameter δ (the bound on KL divergence).
- The constraint on $D_{KL}^{max}(\theta_{old}, \theta)$ is hard for numerical optimization and estimation, so instead we constrain $\overline{D}_{KL}(\theta_{old}, \theta)$.
- Our theory ignores estimation error for the advantage function. Kakade and Langford [KL02] consider this error in their derivation, and the same arguments would hold in the setting of this chapter, but we omit them for simplicity.

3.7 CONNECTIONS WITH PRIOR WORK

As mentioned in Section 3.4, our derivation results in a policy update that is related to several prior methods, providing a unifying perspective on a number of policy update schemes. The natural policy gradient [Kak02] can be obtained as a special case of the update in Equation (14) by using a linear approximation to L and a quadratic approximation to the $\overline{D}_{KL}$ constraint, resulting in the following problem:

$$\begin{aligned}
&\underset{\theta}{\text{maximize}}[\nabla_\theta L_{\theta_{old}}(\theta)\big|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old})] \qquad (17)\\
&\text{subject to } \frac{1}{2}(\theta_{old} - \theta)^T A(\theta_{old})(\theta_{old} - \theta) \leqslant \delta,\\
&\text{where } A(\theta_{old})_{ij} =\\
&\frac{\partial}{\partial\theta_i}\frac{\partial}{\partial\theta_j}\mathbb{E}_{s\sim\rho_\pi}\left[D_{KL}(\pi(\cdot \mid s, \theta_{old}) \parallel \pi(\cdot \mid s, \theta))\right]\big|_{\theta=\theta_{old}}.
\end{aligned}$$

The update is $\theta_{new} = \theta_{old} + \frac{1}{\lambda} A(\theta_{old})^{-1} \nabla_\theta L(\theta)\big|_{\theta=\theta_{old}}$, where the stepsize $\frac{1}{\lambda}$ is typically treated as an algorithm parameter. This differs from our approach, which enforces the constraint at each update. Though this difference might seem subtle, our experiments demonstrate that it significantly improves the algorithm's performance on larger problems.

We can also obtain the standard policy gradient update by using an ℓ_2 constraint or penalty:

$$\begin{aligned} &\underset{\theta}{\text{maximize}}[\nabla_\theta L_{\theta_{old}}(\theta)\big|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old})] \\ &\text{subject to } \frac{1}{2}\|\theta - \theta_{old}\|^2 \leqslant \delta. \end{aligned} \tag{18}$$

The policy iteration update can also be obtained by solving the unconstrained problem $\text{maximize}_\pi L_{\pi_{old}}(\pi)$, using L as defined in Equation (5).

Several other methods employ an update similar to Equation (14). *Relative entropy policy search* (REPS) [PMA10] constrains the state-action marginals $p(s, a)$, while TRPO constrains the conditionals $p(a \mid s)$. Unlike REPS, our approach does not require a costly nonlinear optimization in the inner loop. Levine and Abbeel [LA14] also use a KL divergence constraint, but its purpose is to encourage the policy not to stray from regions where the estimated dynamics model is valid, while we do not attempt to estimate the system dynamics explicitly. Pirotta et al. [Pir+13] also build on and generalize Kakade and Langford's results, and they derive different algorithms from the ones here.

3.8 EXPERIMENTS

We designed our experiments to investigate the following questions:

1. What are the performance characteristics of the *single path* and *vine* sampling procedures?
2. TRPO is related to prior methods (e.g. natural policy gradient) but makes several changes, most notably by using a fixed KL divergence rather than a fixed penalty coefficient. How does this affect the performance of the algorithm?
3. Can TRPO be used to solve challenging large-scale problems? How does TRPO compare with other methods when applied to large-scale problems, with regard to final performance, computation time, and sample complexity?

To answer (1) and (2), we compare the performance of the *single path* and *vine* variants of TRPO, several ablated variants, and a number of prior policy optimization algorithms. With regard to (3), we show that both the *single path* and *vine* algorithm can obtain high-

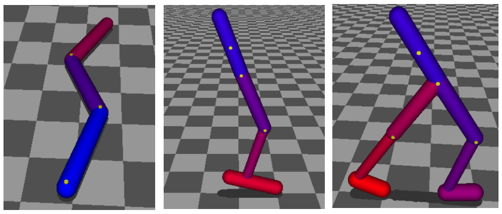


Figure 2: 2D robot models used for locomotion experiments. From left to right: swimmer, hopper, walker. The hopper and walker present a particular challenge, due to underactuation and contact discontinuities.

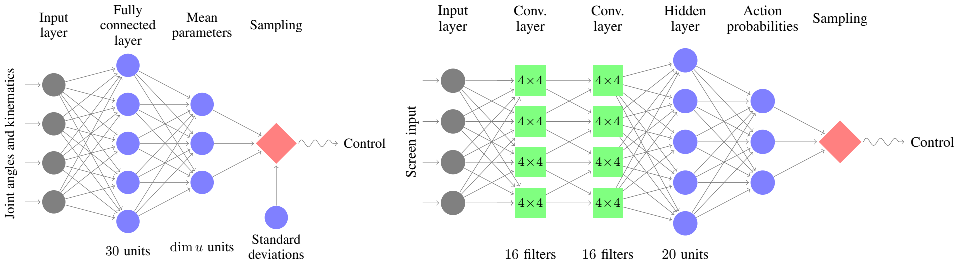


Figure 3: Neural networks used for the locomotion task (left) and for playing Atari games (right). In the locomotion task, the sampled control (red diamond) is a vector u, whereas in Atari, it is a triple of integers that forms a factored representation of the action: see Section 3.13.

quality locomotion controllers from scratch, which is considered to be a hard problem. We also show that these algorithms produce competitive results when learning policies for playing Atari games from images using convolutional neural networks with tens of thousands of parameters.

3.8.1 *Simulated Robotic Locomotion*

We conducted the robotic locomotion experiments using the MuJoCo simulator [TET12]. The three simulated robots are shown in Figure 2. The states of the robots are their generalized positions and velocities, and the controls are joint torques. Underactuation, high dimensionality, and non-smooth dynamics due to contacts make these tasks very challenging. The following models are included in our evaluation:

1. *Swimmer*. 10-dimensional state space, linear reward for forward progress and a quadratic penalty on joint effort to produce the reward $r(x, u) = v_x - 10^{-5}\|u\|^2$. The swimmer can propel itself forward by making an undulating motion.

2. *Hopper*. 12-dimensional state space, same reward as the swimmer, with a bonus of +1 for being in a non-terminal state. We ended the episodes when the hopper fell over, which was defined by thresholds on the torso height and angle.
3. *Walker*. 18-dimensional state space. For the walker, we added a penalty for strong impacts of the feet against the ground to encourage a smooth walk rather than a hopping gait.

We used $\delta = 0.01$ for all experiments. See Table 2 in the Appendix for more details on the experimental setup and parameters used. We used neural networks to represent the policy, with the architecture shown in Figure 3, and further details provided in Section 3.13. To establish a standard baseline, we also included the classic cart-pole balancing problem, based on the formulation from Barto, Sutton, and Anderson [BSA83], using a linear policy with six parameters that is easy to optimize with derivative-free black-box optimization methods.

The following algorithms were considered in the comparison: *single path TRPO*; *vine TRPO*; *cross-entropy method* (CEM), a gradient-free method [SL06]; *covariance matrix adaption* (CMA), another gradient-free method [HO96]; *natural gradient*, the classic natural policy gradient algorithm [Kak02], which differs from *single path* by the use of a fixed penalty coefficient (Lagrange multiplier) instead of the KL divergence constraint; *empirical FIM*, identical to *single path*, except that the FIM is estimated using the covariance matrix of the gradients rather than the analytic estimate; *max KL*, which was only tractable on the cart-pole problem, and uses the maximum KL divergence in Equation (13), rather than the average divergence, allowing us to evaluate the quality of this approximation. The parameters used in the experiments are provided in Section 3.14. For the *natural gradient* method, we swept through the possible values of the stepsize in factors of three, and took the best value according to the final performance.

Learning curves showing the total reward averaged across five runs of each algorithm are shown in Figure 4. *Single path* and *vine* TRPO solved all of the problems, yielding the best solutions. *Natural gradient* performed well on the two easier problems, but was unable to generate hopping and walking gaits that made forward progress. These results provide empirical evidence that constraining the KL divergence is a more robust way to choose step sizes and make fast, consistent progress, compared to using a fixed penalty. CEM and CMA are derivative-free algorithms, hence their sample complexity scales unfavorably with the number of parameters, and they performed poorly on the larger problems. The *max KL* method learned somewhat more slowly than our final method, due to the more restrictive form of the constraint, but overall the result suggests that the average KL divergence constraint has a similar effect as the theorecally justified

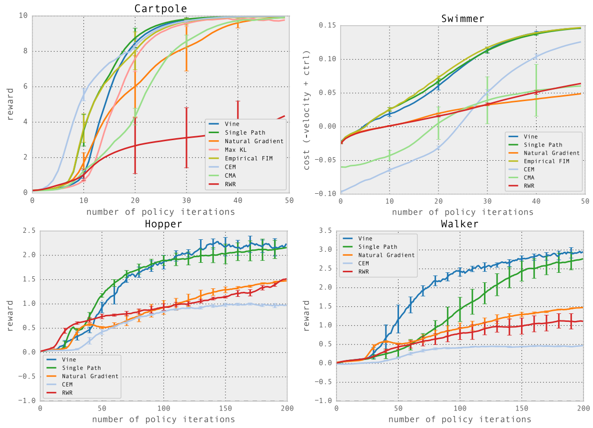


Figure 4: Learning curves for locomotion tasks, averaged across five runs of each algorithm with random initializations. Note that for the hopper and walker, a score of -1 is achievable without any forward velocity, indicating a policy that simply learned balanced standing, but not walking.

maximum KL divergence. Videos of the policies learned by TRPO may be viewed on the project website: http://sites.google.com/site/trpopaper.

Note that TRPO learned all of the gaits with general-purpose policies and simple reward functions, using minimal prior knowledge. This is in contrast with most prior methods for learning locomotion, which typically rely on hand-architected policy classes that explicitly encode notions of balance and stepping [TZS04; GPW06; WP09].

3.8.2 *Playing Games from Images*

To evaluate TRPO on a task with high-dimensional observations, we trained policies for playing Atari games, using raw images as input. The games require learning a variety of behaviors, such as dodging bullets and hitting balls with paddles. Aside from the high dimensionality, challenging elements of these games include delayed rewards (no immediate penalty is incurred when a life is lost in Breakout or Space Invaders); complex sequences of behavior (Q*bert requires a character to hop on 21 different platforms); and non-stationary image statistics (Enduro involves a changing and flickering background).

We tested our algorithms on the same seven games reported on in [Mni+13] and [Guo+14], which are made available through the Arcade Learning Environment [Bel+13] The images were preprocessed following the protocol in Mnih et al [Mni+13], and the policy was represented by the convolutional neural network shown in Figure 3, with two

	B. Rider	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S. Invaders*
Random	354	1.2	0	−20.4	157	110	179
Human [Mni+13]	7456	31.0	368	−3.0	18900	28010	3690
Deep Q Learning [Mni+13]	4092	168.0	470	20.0	1952	1705	581
UCC-I [Guo+14]	5702	380	741	21	20025	2995	692
TRPO - single path	1425.2	10.8	534.6	20.9	1973.5	1908.6	568.4
TRPO - vine	859.5	34.2	430.8	20.9	7732.5	788.4	450.2

Table 1: Performance comparison for vision-based RL algorithms on the Atari domain. Our algorithms (bottom rows) were run once on each task, with the same architecture and parameters. Performance varies substantially from run to run (with different random initializations of the policy), but we could not obtain error statistics due to time constraints.

convolutional layers with 16 channels and stride 2, followed by one fully-connected layer with 20 units, yielding 33,500 parameters.

The results of the *vine* and *single path* algorithms are summarized in Table 1, which also includes an expert human performance and two recent methods: deep Q-learning [Mni+13], and a combination of Monte-Carlo Tree Search with supervised training [Guo+14], called UCC-I. The 500 iterations of our algorithm took about 30 hours (with slight variation between games) on a 16-core computer. While our method only outperformed the prior methods on some of the games, it consistently achieved reasonable scores. Unlike the prior methods, our approach was not designed specifically for this task. The ability to apply the same policy search method to methods as diverse as robotic locomotion and image-based game playing demonstrates the generality of TRPO.

3.9 DISCUSSION

We proposed and analyzed trust region methods for optimizing stochastic control policies. We proved monotonic improvement for an algorithm that repeatedly optimizes a local approximation to the expected return of the policy with a KL divergence penalty, and we showed that an approximation to this method that incorporates a KL divergence constraint achieves good empirical results on a range of challenging policy learning tasks, outperforming prior methods. Our analysis also provides a perspective that unifies policy gradient and policy iteration methods, and shows them to be special limiting cases of an algorithm that optimizes a certain objective subject to a trust region constraint.

In the domain of robotic locomotion, we successfully learned controllers for swimming, walking and hopping in a physics simulator, using general purpose neural networks and minimally informative rewards. To our knowledge, no prior work has learned controllers from scratch for all of these tasks, using a generic policy search method and non-engineered, general-purpose policy representations. In the game-playing domain, we learned convolutional neural network policies that used raw images as inputs. This requires optimizing extremely high-dimensional policies, and only two prior methods report successful results on this task.

Since the method we proposed is scalable and has strong theoretical foundations, we hope that it will serve as a jumping-off point for future work on training large, rich function approximators for a range of challenging problems. At the intersection of the two experimental domains we explored, there is the possibility of learning robotic control policies that use vision and raw sensory data as input, providing a unified scheme for training robotic controllers that perform both perception and control. The use of more sophisticated policies, including recurrent policies with hidden state, could further make it possible to roll state estimation and control into the same policy in the partially-observed setting. By combining our method with model learning, it would also be possible to substantially reduce its sample complexity, making it applicable to real-world settings where samples are expensive.

3.10 PROOF OF POLICY IMPROVEMENT BOUND

This proof uses techniques from the proof of Theorem 4.1 in [KL02], adapting them to the more general setting considered in this chapter.

Lemma 1. *Given two policies $\pi, \tilde{\pi}$,*

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\tau \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right]$$

This expectation is taken over trajectories $\tau := (s_0, a_0, s_1, a_0, \dots)$, and the notation $\mathbb{E}_{\tau \sim \tilde{\pi}}[\dots]$ indicates that actions are sampled from $\tilde{\pi}$ to generate τ.

Proof. First note that $A_\pi(s,a) = \mathbb{E}_{s' \sim P(s' \mid s,a)}\left[r(s) + \gamma V^\pi(s') - V^\pi(s)\right]$. Therefore,

$$\begin{aligned}
&\mathbb{E}_{\tau \mid \tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)\right] \\
&= \mathbb{E}_{\tau \mid \tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t (r(s_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t))\right] \\
&= \mathbb{E}_{\tau \mid \tilde{\pi}}\left[-V^\pi(s_0) + \sum_{t=0}^{\infty} \gamma^t r(s_t)\right] \\
&= -\mathbb{E}_{s_0}\left[V^\pi(s_0)\right] + \mathbb{E}_{\tau \mid \tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right] \qquad (19) \\
&= -\eta(\pi) + \eta(\tilde{\pi})
\end{aligned}$$

Rearranging, the result follows. □

Define $\bar{A}^{\pi,\tilde{\pi}}(s)$ to be the expected advantage of $\tilde{\pi}$ over π at state s:

$$\bar{A}^{\pi,\tilde{\pi}}(s) = \mathbb{E}_{a \sim \tilde{\pi}(\cdot \mid s)}\left[A^\pi(s,a)\right].$$

Now Lemma 1 can be written as follows:

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\tau \sim \tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t \bar{A}^{\pi,\tilde{\pi}}(s_t)\right]$$

Note that L_π can be written as

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t \bar{A}^{\pi,\tilde{\pi}}(s_t)\right]$$

The difference in these equations is whether the states are sampled using π or $\tilde{\pi}$. To bound the difference between $\eta(\tilde{\pi})$ and $L_\pi(\tilde{\pi})$, we will bound the difference arising from each timestep. To do this, we first need to introduce a measure of how much π and $\tilde{\pi}$ agree. Specifically, we'll *couple* the policies, so that they define a joint distribution over pairs of actions.

Definition 1. *$(\pi, \tilde{\pi})$ is an α-coupled policy pair if it defines a joint distribution $(a, \tilde{a}) \mid s$, such that $P(a \neq \tilde{a} \mid s) \leqslant \alpha$ for all s. π and $\tilde{\pi}$ will denote the marginal distributions of a and $\tilde{a}$, respectively.*

In words, this means that at each state, $(\pi, \tilde{\pi})$ gives us a pair of actions, and these actions differ with probability $\leqslant \alpha$.

Lemma 2. *Let $(\pi, \tilde{\pi})$ be an α-coupled policy pair. Then*

$$\begin{aligned}&\left|\mathbb{E}_{s_t \sim \tilde{\pi}}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] - \mathbb{E}_{s_t \sim \pi}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right]\right| \leqslant 2\epsilon(1-(1-\alpha)^t),\\ &\qquad \textit{where } \epsilon = \max_s |\bar{A}^{\pi,\tilde{\pi}}(s)|\end{aligned}$$

Proof. Consider generating a trajectory using $\tilde{\pi}$, i.e., at each timestep i we sample $(a_i, \tilde{a}_i) \mid s_t$, and we choose the action $\tilde{a}_i$ and ignore a_i. Let n_t denote the number of times that $a_i \neq \tilde{a}_i$ for $i < t$, i.e., the number of times that π and $\tilde{\pi}$ disagree before arriving at state s_t.

$$\mathbb{E}_{s_t \sim \tilde{\pi}}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] = P(n_t = 0)\mathbb{E}_{s_t \sim \tilde{\pi} \mid n_t = 0}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] + P(n_t > 0)\mathbb{E}_{s_t \sim \tilde{\pi} \mid n_t > 0}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right]$$

$P(n_t = 0) = (1-\alpha)^t$, and $\mathbb{E}_{s_t \sim \tilde{\pi} \mid n_t = 0}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] = \mathbb{E}_{s_t \sim \pi \mid n_t = 0}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right]$, because $n_t = 0$ indicates that π and $\tilde{\pi}$ agreed on all timesteps less than t. Therefore, we have

$$\mathbb{E}_{s_t \sim \tilde{\pi}}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] = (1-\alpha^t)\mathbb{E}_{s_t \sim \pi \mid n_t = 0}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] + (1-(1-\alpha^t))\mathbb{E}_{s_t \sim \tilde{\pi} \mid n_t > 0}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right]$$

Subtracting $\mathbb{E}_{s_t \sim \pi \mid n_t = 0}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right]$ from both sides,

$$\begin{aligned}&\mathbb{E}_{s_t \sim \tilde{\pi}}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] - \mathbb{E}_{s_t \sim \pi}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] = (1-(1-\alpha^t))(-\mathbb{E}_{s_t \sim \pi \mid n_t = 0}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] + \mathbb{E}_{s_t \sim \tilde{\pi} \mid n_t > 0}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right])\\ &\left|\mathbb{E}_{s_t \sim \tilde{\pi}}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] - \mathbb{E}_{s_t \sim \pi}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right]\right| \leqslant (1-(1-\alpha^t))(\epsilon + \epsilon)\end{aligned}$$

□

Now we can sum over time to bound the error of L_π.

Lemma 3. *Suppose $(\pi, \tilde{\pi})$ is an α-coupled policy pair. Then*

$$|\eta(\tilde{\pi}) - L_\pi(\tilde{\pi})| \leqslant \frac{2\epsilon\gamma\alpha}{(1-\gamma)(1-\gamma(1-\alpha))}$$

Proof.

$$\begin{aligned}
\eta(\tilde{\pi}) - L_{\pi}(\tilde{\pi}) &= \mathbb{E}_{\tau\sim\tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t \bar{A}^{\pi,\tilde{\pi}}(s_t)\right] - \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t \bar{A}^{\pi,\tilde{\pi}}(s_t)\right] \\
&= \sum_{t=0}^{\infty} \gamma^t (\mathbb{E}_{s_t\sim\tilde{\pi}}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] - \mathbb{E}_{s_t\sim\pi}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right]) \\
|\eta(\tilde{\pi}) - L_{\pi}(\tilde{\pi})| &\leqslant \sum_{t=0}^{\infty} \gamma^t \left|\mathbb{E}_{s_t\sim\tilde{\pi}}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right] - \mathbb{E}_{s_t\sim\pi}\left[\bar{A}^{\pi,\tilde{\pi}}(s_t)\right]\right| \\
&\leqslant \sum_{t=0}^{\infty} \gamma^t \cdot 2\epsilon \cdot (1 - (1-\alpha^t)) \\
&= \frac{2\epsilon\gamma\alpha}{(1-\gamma)(1-\gamma(1-\alpha))}
\end{aligned}$$

□

Last, we need to use the correspondence between total variation divergence and coupled random variables:

> Suppose p_X and p_Y are distributions with $D_{TV}(p_X \parallel p_Y) = \alpha$. Then there exists a joint distribution (X, Y) whose marginals are p_X, p_Y, for which $X = Y$ with probability $1-\alpha$.

See [LPW09], Proposition 4.7.

It follows that if we have two policies π and $\tilde{\pi}$ such that $\max_s D_{TV}(\pi(\cdot \mid s) \parallel \tilde{\pi}(\cdot \mid s))\alpha$, then we can define an α-coupled policy pair $(\pi, \tilde{\pi})$ with appropriate marginals. Proposition 1 follows.

3.11 PERTURBATION THEORY PROOF OF POLICY IMPROVEMENT BOUND

We also provide a different proof of Proposition 1 using perturbation theory. This method makes it possible to provide slightly stronger bounds.

Proposition 1a. *Let α denote the maximum total variation divergence between stochastic policies π and $\tilde{\pi}$, as defined in Equation (10), and let L be defined as in Equation (5). Then*

$$\eta(\tilde{\pi}) \geqslant L(\tilde{\pi}) - \alpha^2 \frac{2\gamma\epsilon}{(1-\gamma)^2}$$

where

$$\epsilon = \min_s \left\{ \frac{\sum_a (\tilde{\pi}(a \mid s) Q^{\pi}(s,a) - \pi(a \mid s) Q^{\pi}(s,a))}{\sum_a |\tilde{\pi}(a \mid s) - \pi(a \mid s)|} \right\} \tag{20}$$

Note that the ϵ defined in Equation (20) is less than or equal to the ϵ defined in Proposition 1. So Proposition 1a is slightly stronger.

Proof. Let $G = (1 + \gamma P_{\pi} + (\gamma P_{\pi})^2 + \dots) = (1 - \gamma P_{\pi})^{-1}$, and similarly Let $\tilde{G} = (1 + \gamma P_{\tilde{\pi}} + (\gamma P_{\tilde{\pi}})^2 + \dots) = (1 - \gamma P_{\tilde{\pi}})^{-1}$. We will use the convention that ρ (a density on state space) is a vector and r (a reward function on state space) is a dual vector (i.e., linear functional on vectors), thus $r\rho$ is a scalar meaning the expected reward under density ρ. Note that $\eta(\pi) = rG\rho_0$, and $\eta(\tilde{\pi}) = c\tilde{G}\rho_0$. Let $\Delta = P_{\tilde{\pi}} - P_{\pi}$. We want to bound $\eta(\tilde{\pi}) - \eta(\pi) = r(\tilde{G} - G)\rho_0$. We start with some standard perturbation theory manipulations.

$$\begin{aligned} G^{-1} - \tilde{G}^{-1} &= (1 - \gamma P_{\pi}) - (1 - \gamma P_{\tilde{\pi}}) \\ &= \gamma \Delta. \end{aligned}$$

Left multiply by G and right multiply by $\tilde{G}$.

$$\begin{aligned} \tilde{G} - G &= \gamma G \Delta \tilde{G} \\ \tilde{G} &= G + \gamma G \Delta \tilde{G} \end{aligned}$$

Substituting the right-hand side into $\tilde{G}$ gives

$$\tilde{G} = G + \gamma G \Delta G + \gamma^2 G \Delta G \Delta \tilde{G}$$

So we have

$$\eta(\tilde{\pi}) - \eta(\pi) = r(\tilde{G} - G)\rho = \gamma r G \Delta G \rho_0 + \gamma^2 r G \Delta G \Delta \tilde{G} \rho_0$$

Let us first consider the leading term $\gamma r G \Delta G \rho_0$. Note that $rG = v$, i.e., the infinite-horizon state-value function. Also note that $G\rho_0 = \rho_{\pi}$. Thus we can write $\gamma c G \Delta G \rho_0 = \gamma v \Delta \rho_{\pi}$. We will show that this expression equals the expected advantage $L_{\pi}(\tilde{\pi}) - L_{\pi}(\pi)$.

$$\begin{aligned} L_{\pi}(\tilde{\pi}) - L_{\pi}(\pi) &= \sum_s \rho_{\pi}(s) \sum_a (\tilde{\pi}(a \mid s) - \pi(a \mid s)) A^{\pi}(s,a) \\ &= \sum_s \rho_{\pi}(s) \sum_a (\pi_{\theta}(a \mid s) - \pi_{\tilde{\theta}}(a \mid s))[r(s) + \sum_{s'} p(s' \mid s,a) \gamma v(s') - v(s)] \\ &= \sum_s \rho_{\pi}(s) \sum_{s'} \sum_a (\pi(a \mid s) - \tilde{\pi}(a \mid s)) p(s' \mid s,a) \gamma v(s') \\ &= \sum_s \rho_{\pi}(s) \sum_{s'} (p_{\pi}(s' \mid s) - p_{\tilde{\pi}}(s' \mid s)) \gamma v(s') \\ &= \gamma v \Delta \rho_{\pi} \end{aligned}$$

Next let us bound the $O(\Delta^2)$ term $\gamma^2 rG\Delta G\Delta\tilde{G}\rho$. First we consider the product $\gamma rG\Delta = \gamma v\Delta$. Consider the component s of this dual vector.

$$\begin{aligned}
(\gamma v\Delta)_s &= \sum_a (\tilde{\pi}(s,a) - \pi(s,a))Q^{\pi}(s,a) \\
&= \sum_a |\tilde{\pi}(a \mid s) - \pi(a \mid s)| \frac{\sum_a (\tilde{\pi}(s,a) - \pi(s,a))Q^{\pi}(s,a)}{\sum_a |\tilde{\pi}(a \mid s) - \pi(a \mid s)|} \\
&\leqslant \alpha\epsilon
\end{aligned}$$

We bound the other portion $G\Delta\tilde{G}\rho$ using the ℓ_1 operator norm

$$\|A\|_1 = \sup_{\rho}\{\frac{\|A\rho\|_1}{\|\rho\|_1}\}$$

where we have that $\|G\|_1 = \|\tilde{G}\|_1 = 1/(1-\gamma)$ and $\|\Delta\|_1 = 2\alpha$. That gives

$$\begin{aligned}
\|G\Delta\tilde{G}\rho\|_1 &\leqslant \|G\|_1\|\Delta\|_1\|\tilde{G}\|_1\|\rho\|_1 \\
&= \frac{1}{1-\gamma} \cdot \alpha \cdot \frac{1}{1-\gamma} \cdot 1
\end{aligned}$$

So we have that

$$\begin{aligned}
\gamma^2 |rG\Delta G\Delta\tilde{G}\rho| &\leqslant \gamma\|\gamma rG\Delta\|_{\infty}\|G\Delta\tilde{G}\rho\|_1 \\
&\leqslant \gamma \cdot \alpha\epsilon \cdot \frac{2\alpha}{(1-\gamma)^2} \\
&= \alpha^2 \frac{2\gamma\epsilon}{(1-\gamma)^2}
\end{aligned}$$

□

3.12 EFFICIENTLY SOLVING THE TRUST-REGION CONSTRAINED OPTIMIZATION PROBLEM

This section describes how to efficiently approximately solve the following constrained optimization problem, which we must solve at each iteration of TRPO:

$$\text{maximize}\, L(\theta) \quad \text{subject to}\ \overline{D}_{KL}(\theta_{old}, \theta) \leqslant \delta.$$

The method we will describe involves two steps: (1) compute a search direction, using a linear approximation to objective and quadratic approximation to the constraint; and (2) perform a line search in that direction, ensuring that we improve the nonlinear objective while satisfying the nonlinear constraint.

The search direction is computed by approximately solving the equation $Ax = g$, where A is the Fisher information matrix, i.e., the quadratic approximation to the KL divergence constraint: $\overline{D}_{KL}(\theta_{old}, \theta) \approx \frac{1}{2}(\theta - \theta_{old})^T A(\theta - \theta_{old})$, where $A_{ij} = \frac{\partial}{\partial\theta_i}\frac{\partial}{\partial\theta_j}\overline{D}_{KL}(\theta_{old}, \theta)$. In large-scale problems, it is prohibitively costly (with respect to computation and memory) to form the full matrix A (or A^{-1}). However, the conjugate gradient algorithm allows us to approximately solve the equation $Ax = b$ without forming this full matrix, when we merely have access to a function that computes matrix-vector products $y \to Ay$. Section 3.12.1 describes the most efficient way to compute matrix-vector products with the Fisher information matrix. For additional exposition on the use of Hessian-vector products for optimizing neural network objectives, see [MS12] and [PB13].

Having computed the search direction $s \approx A^{-1}g$, we next need to compute the maximal step length β such that $\theta + \beta s$ will satisfy the KL divergence constraint. To do this, let $\delta = \overline{D}_{KL} \approx \frac{1}{2}(\beta s)^T A(\beta s) = \frac{1}{2}\beta^2 s^T A s$. From this, we obtain $\beta = \sqrt{2\delta / s^T A s}$, where δ is the desired KL divergence. The term $s^T A s$ can be computed through a single Hessian vector product, and it is also an intermediate result produced by the conjugate gradient algorithm.

Last, we use a line search to ensure improvement of the surrogate objective and satisfaction of the KL divergence constraint, both of which are nonlinear in the parameter vector θ (and thus depart from the linear and quadratic approximations used to compute the step). We perform the line search on the objective $L_{\theta_{old}}(\theta) - \mathcal{X}[\overline{D}_{KL}(\theta_{old}, \theta) \leqslant \delta]$, where $\mathcal{X}[\dots]$ equals zero when its argument is true and $+\infty$ when it is false. Starting with the maximal value of the step length β computed in the previous paragraph, we shrink β exponentially until the objective improves. Without this line search, the algorithm occasionally computes large steps that cause a catastrophic degradation of performance.

3.12.1 *Computing the Fisher-Vector Product*

Here we will describe how to compute the matrix-vector product between the averaged Fisher information matrix and arbitrary vectors; this calculation is also described in other references such as [PB13], but we include it here for self-containedness. This matrix-vector product enables us to perform the conjugate gradient algorithm. Suppose that the parameterized policy maps from the input x to "distribution parameter" vector $\mu_\theta(x)$, which parameterizes the distribution $\pi(u \mid x)$. (For example, for a Gaussian distribution, μ could be the mean and standard deviation concatenated; for a categorical distribution, it could be the vector of probabilities or log-probabilities.) Now the KL divergence for a given input x can be written as follows:

$$D_{KL}(\pi_{\theta_{old}}(\cdot \mid x) \parallel \pi_\theta(\cdot \mid x)) = \mathrm{kl}(\mu_\theta(x), \mu_{old}(x))$$

where kl is the KL divergence between the distributions corresponding to the two mean parameter vectors. Let us assume we can compute kl analytically in terms of its arguments. Differentiating kl twice with respect to θ, we obtain

$$\underbrace{\frac{\partial \mu_a(x)}{\partial \theta_i} \mathrm{kl}''_{ab}(\mu_\theta(x), \mu_{old}(x)) \frac{\partial \mu_b(x)}{\partial \theta_j}}_{J^T M J} + \underbrace{\frac{\partial^2 \mu_a(x)}{\partial \theta_i \partial \theta_j} \mathrm{kl}'_a(\mu_\theta(x), \mu_{old}(x))}_{=0 \text{ at } \mu_\theta = \mu_{old}} \qquad (21)$$

where the primes ($'$) indicate differentiation with respect to the first argument, and there is an implied summation over indices a, b. The second term vanishes because the KL divergence is minimized at $\mu_\theta = \mu_{old}$, and the derivative is zero at a minimum. Let $J := \frac{\partial \mu_a(x)}{\partial \theta_i}$ (the Jacobian), then the Fisher information matrix can be written in matrix form as $J^T M J$, where $M = \mathrm{kl}''_{ab}(\mu_\theta(x), \mu_{old})$ is the Fisher information matrix of the distribution in terms of the mean parameter μ (as opposed to the parameter θ). M has a simple form for most parameterized distributions of interest.

The Fisher-vector product can now be written as a function $y \to J^T M J y$. Multiplication by J^T and J can be performed by automatic differentiation software such as Theano [Ber+10], and the matrix M (the Fisher matrix with respect to μ) can be computed analytically for the distribution of interest. Note that multiplication by J^T is the well-known *backpropagation* operation, whereas multiplication by J is *tangent-propagation* [Gri+89] or the R-Op (in Theano).

There is a simpler but (slightly) less efficient way to calculate the Fisher-vector products using only reverse mode automatic differentiation. This technique is described in [WN99], chapter 8. Let $f(\theta) = \mathrm{kl}(\mu_\theta(x), \mu_{old})$, then we want to compute the Hessian-vector product Hy, where y is a vector, and H is the Hessian of $f(\theta)$. We can first form the expression for the gradient-vector product $\nabla_\theta f(\theta) \cdot p$, then we differentiate this expression to get the Hessian-vector product. This method is slightly less efficient than the one above as it does not exploit the fact that the second derivatives of $\mu(x)$ (i.e., the second term in Equation (21)) can be ignored, but may be substantially easier to implement.

We have described a procedure for computing the Fisher-vector product $y \to Ay$, where the Fisher information matrix is averaged over a set of inputs to the function μ.

Computing the Fisher-vector product is typically about as expensive as computing the gradient of an objective that depends on $\mu(x)$ [WN99]. Furthermore, we need to compute k of these Fisher-vector products per gradient, where k is the number of iterations of the conjugate gradient algorithm we perform. We found $k = 10$ to be quite effective, and using higher k did not result in faster policy improvement. Hence, a naïve implementation would spend more than 90% of the computational effort on these Fisher-vector products. However, we can greatly reduce this burden by subsampling the data for the computation of Fisher-vector product. Since the Fisher information matrix merely acts as a metric, it can be computed on a subset of the data without severely degrading the quality of the final step. Hence, we can compute it on 10% of the data, and the total cost of Hessian-vector products will be about the same as computing the gradient. With this optimization, the computation of a natural gradient step $A^{-1}g$ does not incur a significant extra computational cost beyond computing the gradient g.

3.13 APPROXIMATING FACTORED POLICIES WITH NEURAL NETWORKS

The policy, which is a conditional probability distribution $\pi_\theta(a \mid s)$, can be parameterized with a neural network. The most straightforward way to do so is to have the neural network map (deterministically) from the state vector s to a vector μ that specifies a distribution over action space. Then we can compute the likelihood $p(a \mid \mu)$ and sample $a \sim p(a \mid \mu)$.

For our experiments with continuous state and action spaces, we used a Gaussian distribution, where the covariance matrix was diagonal and independent of the state. A neural network with several fully-connected (dense) layers maps from the input features to the mean of a Gaussian distribution. A separate set of parameters specifies the log standard deviation of each element. More concretely, the parameters include a set of weights and biases for the neural network computing the mean, $\{W_i, b_i\}_{i=1}^L$, and a vector r (log standard deviation) with the same dimension as a. Then, the policy is defined by the normal distribution $\mathcal{N}\left(\text{mean} = \text{NeuralNet}\left(s; \{W_i, b_i\}_{i=1}^L\right), \text{stdev} = \exp(r)\right)$. Here, $\mu = [\text{mean}, \text{stdev}]$.

For the experiments with discrete actions (Atari), we use a factored discrete action space, where each factor is parameterized as a categorical distribution. These factors correspond to the action components (left, no-op, right), (up, no-op, down), (fire, no-fire). Thus, the neural network output a vector of dimension $3 + 3 + 2 = 8$, where each of the components was normalized. The process for computing the factored probability distribution is shown in Figure 5 below.

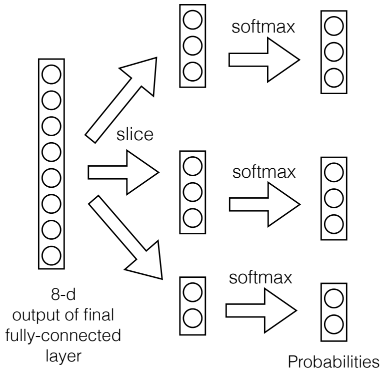


Figure 5: Computation of factored discrete probability distribution in Atari domain

3.14 EXPERIMENT PARAMETERS

	Swimmer	Hopper	Walker
State space dim.	10	12	20
Control space dim.	2	3	6
Total num. policy params	364	4806	8206
Sim. steps per iter.	50K	1M	1M
Policy iter.	200	200	200
Stepsize ($\overline{D}_{KL}$)	0.01	0.01	0.01
Hidden layer size	30	50	50
Discount (γ)	0.99	0.99	0.99
Vine: rollout length	50	100	100
Vine: rollouts per state	4	4	4
Vine: Q-values per batch	500	2500	2500
Vine: num. rollouts for sampling	16	16	16
Vine: len. rollouts for sampling	1000	1000	1000
Vine: computation time (minutes)	2	14	40
SP: num. path	50	1000	10000
SP: path len.	1000	1000	1000
SP: computation time	5	35	100

Table 2: Parameters for continuous control tasks, vine and single path (SP) algorithms.

	All games
Total num. policy params	33500
Vine: Sim. steps per iter.	400K
SP: Sim. steps per iter.	100K
Policy iter.	500
Stepsize ($\overline{D}_{KL}$)	0.01
Discount (γ)	0.99
Vine: rollouts per state	$\approx$ 4
Vine: computation time	$\approx$ 30 hrs
SP: computation time	$\approx$ 30 hrs

Table 3: Parameters used for Atari domain.

3.15 LEARNING CURVES FOR THE ATARI DOMAIN

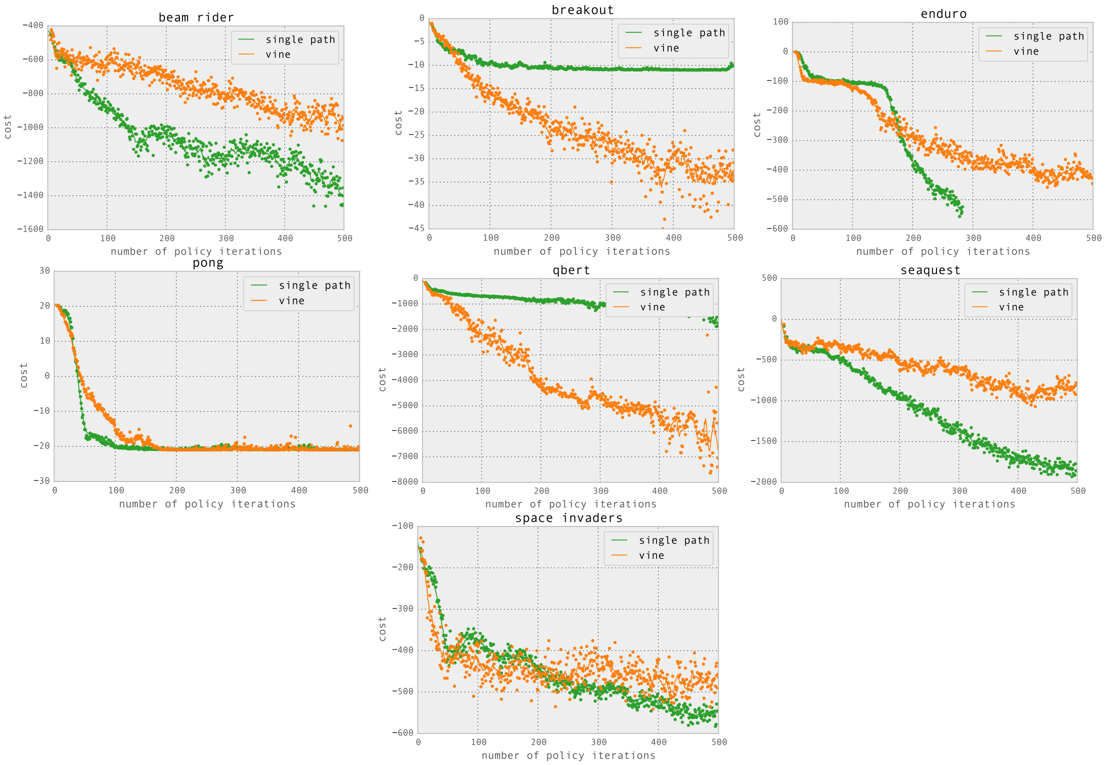


Figure 6: Learning curves for the Atari domain. For historical reasons, the plots show cost = negative reward.

4

GENERALIZED ADVANTAGE ESTIMATION

4.1 OVERVIEW

The two main challenges with policy gradient methods are the large number of samples typically required, and the difficulty of obtaining monotonic improvement despite the nonstationarity of the incoming data. The previous chapter addressed the monotonicity issue and provided some improvement in sample complexity due to the use of the natural gradient step, which was theoretically justified. This chapter provides further improvements to sample complexity issue, by reducing the variance of the policy gradient estimates—the techniques of this chapter are equally applicable to other policy gradient methods such as the vanilla policy gradient algorithm.

In this chapter, we propose a family of policy gradient estimators that significantly reduce variance of the policy gradient estimators while maintaining a tolerable level of bias. We call this estimation scheme, parameterized by $\gamma \in [0,1]$ and $\lambda \in [0,1]$, the generalized advantage estimator (GAE). Related methods have been proposed in the context of online actor-critic methods [KK98; Waw09]. We provide a more general analysis, which is applicable in both the online and batch settings, and discuss an interpretation of our method as an instance of reward shaping [NHR99], where the approximate value function is used to shape the reward.

We present experimental results on a number of highly challenging 3D locomotion tasks, where we show that our approach can learn complex gaits using high-dimensional, general purpose neural network function approximators for both the policy and the value function, each with over 10^4 parameters. The policies perform torque-level control of simulated 3D robots with up to 33 state dimensions and 10 actuators.

The contributions of this chapter are summarized as follows:

1. We provide justification and intuition for an effective variance reduction scheme for policy gradients, which we call generalized advantage estimation (GAE). While

the formula has been proposed in prior work [KK98; Waw09], our analysis is novel and enables GAE to be applied with a more general set of algorithms, including the batch trust-region algorithm we use for our experiments.

2. We propose the use of a trust region optimization method for the value function, which we find is a robust and efficient way to train neural network value functions with thousands of parameters.
3. By combining (1) and (2) above, we obtain an algorithm that empirically is effective at learning neural network policies for challenging control tasks. The results extend the state of the art in using reinforcement learning for high-dimensional continuous control. Videos are available at `https://sites.google.com/site/gaepapersupp`.

4.2 Preliminaries

In this chapter, we consider an undiscounted formulation of the policy optimization problem. The initial state s_0 is sampled from distribution ρ_0. A trajectory $(s_0, a_0, s_1, a_1, \dots)$ is generated by sampling actions according to the policy $a_t \sim \pi(a_t \mid s_t)$ and sampling the states according to the dynamics $s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$, until a terminal (absorbing) state is reached. A reward $r_t = r(s_t, a_t, s_{t+1})$ is received at each timestep. The goal is to maximize the expected total reward $\sum_{t=0}^{\infty} r_t$, which is assumed to be finite for all policies. Note that we are not using a discount as part of the problem specification; it will appear below as an algorithm parameter that adjusts a bias-variance tradeoff. But the discounted problem (maximizing $\sum_{t=0}^{\infty} \gamma^t r_t$) can be handled as an instance of the undiscounted problem in which we absorb the discount factor into the reward function, making it time-dependent.

Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $\mathbf{g} := \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$. There are several different related expressions for the policy gradient, which have the form

$$\mathbf{g} = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right], \tag{22}$$

where Ψ_t may be one of the following:

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^{\pi}(s_t, a_t)$: state-action value function.
5. $A^{\pi}(s_t, a_t)$: advantage function.
6. $r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$: TD residual.

The latter formulas use the definitions

$$V^{\pi}(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \qquad Q^{\pi}(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t+1:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right]$$

$$A^{\pi}(s_t, a_t) := Q^{\pi}(s_t, a_t) - V^{\pi}(s_t), \quad \text{(Advantage function).}$$

Here, the subscript of $\mathbb{E}$ enumerates the variables being integrated over, where states and actions are sampled sequentially from the dynamics model $P(s_{t+1} \mid s_t, a_t)$ and policy $\pi(a_t \mid s_t)$, respectively. The colon notation $a : b$ refers to the inclusive range $(a, a+1, \ldots, b)$. These formulas are well known and straightforward to obtain; they follow directly from Proposition 1, which will be stated shortly.

The choice $\Psi_t = A^{\pi}(s_t, a_t)$ yields almost the lowest possible variance, though in practice, the advantage function is not known and must be estimated. This statement can be intuitively justified by the following interpretation of the policy gradient: that a step in the policy gradient direction should increase the probability of better-than-average actions and decrease the probability of worse-than-average actions. The advantage function, by it's definition $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$, measures whether or not the action is better or worse than the policy's default behavior. Hence, we should choose Ψ_t to be the advantage function $A^{\pi}(s_t, a_t)$, so that the gradient term $\Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)$ points in the direction of increased $\pi_\theta(a_t \mid s_t)$ if and only if $A^{\pi}(s_t, a_t) > 0$. See [GBB04] for a more rigorous analysis of the variance of policy gradient estimators and the effect of using a baseline.

We will introduce a parameter γ that allows us to reduce variance by downweighting rewards corresponding to delayed effects, at the cost of introducing bias. This parameter corresponds to the discount factor used in discounted formulations of MDPs, but we treat it as a variance reduction parameter in an undiscounted problem; this technique was analyzed theoretically by [MT03; Kak01b; Tho14]. The discounted value functions are given by:

$$V^{\pi,\gamma}(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t:\infty}}}\left[\sum_{l=0}^{\infty} \gamma^l r_{t+l}\right] \qquad Q^{\pi,\gamma}(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t+1:\infty}}}\left[\sum_{l=0}^{\infty} \gamma^l r_{t+l}\right]$$

$$A^{\pi,\gamma}(s_t, a_t) := Q^{\pi,\gamma}(s_t, a_t) - V^{\pi,\gamma}(s_t).$$

The discounted approximation to the policy gradient is defined as follows:

$$\mathbf{g}^{\gamma} := \mathbb{E}_{\substack{s_{0:\infty}\\ a_{0:\infty}}}\left[\sum_{t=0}^{\infty} A^{\pi,\gamma}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]. \tag{23}$$

The following section discusses how to obtain biased (but not too biased) estimators for $A^{\pi,\gamma}$, giving us noisy estimates of the discounted policy gradient in Equation (23).

Before proceeding, we will introduce the notion of a γ-just estimator of the advantage function, which is an estimator that does not introduce bias when we use it in place of $A^{\pi,\gamma}$ (which is not known and must be estimated) in Equation (23) to estimate $\mathbf{g}^\gamma$.[1] Consider an advantage estimator $\hat{A}_t(s_{0:\infty}, a_{0:\infty})$, which may in general be a function of the entire trajectory.

Definition 2. *The estimator $\hat{A}_t$ is γ-just if*

$$\mathbb{E}_{\substack{s_{0:\infty}\\a_{0:\infty}}}\left[\hat{A}_t(s_{0:\infty}, a_{0:\infty})\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] = \mathbb{E}_{\substack{s_{0:\infty}\\a_{0:\infty}}}\left[A^{\pi,\gamma}(s_t, a_t)\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right].$$

It follows immediately that if $\hat{A}_t$ is γ-just for all t, then

$$\mathbb{E}_{\substack{s_{0:\infty}\\a_{0:\infty}}}\left[\sum_{t=0}^{\infty}\hat{A}_t(s_{0:\infty}, a_{0:\infty})\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] = \mathbf{g}^\gamma \tag{24}$$

One sufficient condition for $\hat{A}_t$ to be γ-just is that $\hat{A}_t$ decomposes as the difference between two functions Q_t and b_t, where Q_t can depend on any trajectory variables but gives an unbiased estimator of the γ-discounted Q-function, and b_t is an arbitrary function of the states and actions sampled before a_t.

Proposition 2. *Suppose that $\hat{A}_t$ can be written in the form* $\hat{A}_t(s_{0:\infty}, a_{0:\infty}) = Q_t(s_{0:\infty}, a_{0:\infty}) - b_t(s_{0:t}, a_{0:t-1})$ *such that for all* (s_t, a_t), $\mathbb{E}_{s_{t+1:\infty}, a_{t+1:\infty} \mid s_t, a_t}\left[Q_t(s_{t:\infty}, a_{t:\infty})\right] = Q^{\pi,\gamma}(s_t, a_t)$. *Then $\hat{A}$ is γ-just.*

The proof is provided in Section 4.9. It is easy to verify that the following expressions are γ-just advantage estimators for $\hat{A}_t$:

- $\sum_{l=0}^{\infty} \gamma^l r_{t+l}$
- $Q^{\pi,\gamma}(s_t, a_t)$
- $A^{\pi,\gamma}(s_t, a_t)$
- $r_t + \gamma V^{\pi,\gamma}(s_{t+1}) - V^{\pi,\gamma}(s_t)$.

[1] Note, that we have already introduced bias by using $A^{\pi,\gamma}$ in place of A^π; here we are concerned with obtaining an unbiased estimate of $\mathbf{g}^\gamma$, which is a biased estimate of the policy gradient of the undiscounted MDP.

4.3 ADVANTAGE FUNCTION ESTIMATION

This section will be concerned with producing an accurate estimate $\hat{A}_t$ of the discounted advantage function $A^{\pi,\gamma}(s_t, a_t)$, which will then be used to construct a policy gradient estimator of the following form:

$$\hat{g} = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{\infty} \hat{A}_t^n \nabla_\theta \log \pi_\theta(a_t^n \mid s_t^n) \tag{25}$$

where n indexes over a batch of episodes.

Let V be an approximate value function. Define $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$, i.e., the TD residual of V with discount γ [SB98]. Note that δ_t^V can be considered as an estimate of the advantage of the action a_t. In fact, if we have the correct value function $V = V^{\pi,\gamma}$, then it is a γ-just advantage estimator, and in fact, an unbiased estimator of $A^{\pi,\gamma}$:

$$\begin{aligned}
\mathbb{E}_{s_{t+1}}\left[\delta_t^{V^{\pi,\gamma}}\right] &= \mathbb{E}_{s_{t+1}}\left[r_t + \gamma V^{\pi,\gamma}(s_{t+1}) - V^{\pi,\gamma}(s_t)\right] \\
&= \mathbb{E}_{s_{t+1}}\left[Q^{\pi,\gamma}(s_t, a_t) - V^{\pi,\gamma}(s_t)\right] = A^{\pi,\gamma}(s_t, a_t).
\end{aligned}$$

However, this estimator is only γ-just for $V = V^{\pi,\gamma}$, otherwise it will yield biased policy gradient estimates.

Next, let us consider taking the sum of k of these δ terms, which we will denote by $\hat{A}_t^{(k)}$

$$\begin{aligned}
\hat{A}_t^{(1)} &:= \delta_t^V &&= -V(s_t) + r_t + \gamma V(s_{t+1}) \\
\hat{A}_t^{(2)} &:= \delta_t^V + \gamma \delta_{t+1}^V &&= -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \\
\hat{A}_t^{(3)} &:= \delta_t^V + \gamma \delta_{t+1}^V + \gamma^2 \delta_{t+2}^V &&= -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3})
\end{aligned}$$

$$\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_t^V = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k})$$

These equations result from a telescoping sum, and we see that $\hat{A}_t^{(k)}$ involves a k-step estimate of the returns, minus a baseline term $-V(s_t)$. Analogously to the case of $\delta_t^V = \hat{A}_t^{(1)}$, we can consider $\hat{A}_t^{(k)}$ to be an estimator of the advantage function, which is only γ-just when $V = V^{\pi,\gamma}$. However, note that the bias generally becomes smaller as $k \to \infty$, since the term $\gamma^k V(s_{t+k})$ becomes more heavily discounted, and the term $-V(s_t)$ does not affect the bias. Taking $k \to \infty$, we get

$$\hat{A}_t^{(\infty)} = \sum_{l=0}^{k-1} \gamma^l \delta_t^V = -V(s_t) + \sum_{l=0}^{\infty} \gamma^l r_{t+l},$$

which is simply the empirical returns minus the value function baseline.

The generalized advantage estimator $\mathrm{GAE}(\gamma, \lambda)$ is defined as the exponentially-weighted average of these k-step estimators:

$$\begin{aligned}
\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} &:= (1-\lambda)\Big(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \dots\Big) \\
&= (1-\lambda)\Big(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + \dots\Big) \\
&= (1-\lambda)(\delta_t^V(1+\lambda+\lambda^2+\dots) + \gamma\delta_{t+1}^V(\lambda+\lambda^2+\lambda^3+\dots) \\
&\qquad + \gamma\delta_{t+2}^V(\lambda^2+\lambda^3+\lambda^4+\dots) + \dots) \\
&= (1-\lambda)\left(\delta_t^V\left(\frac{1}{1-\lambda}\right) + \gamma\delta_{t+1}^V\left(\frac{\lambda}{1-\lambda}\right) + \gamma^2\delta_{t+2}^V\left(\frac{\lambda^2}{1-\lambda}\right) + \dots\right) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned} \tag{26}$$

From Equation (26), we see that the advantage estimator has a remarkably simple formula involving a discounted sum of Bellman residual terms. Section 4.4 discusses an interpretation of this formula as the returns in an MDP with a modified reward function. The construction we used above is closely analogous to the one used to define TD(λ) [SB98], however TD(λ) is an estimator of the value function, whereas here we are estimating the advantage function.

There are two notable special cases of this formula, obtained by setting $\lambda = 0$ and $\lambda = 1$.

$$\mathrm{GAE}(\gamma, 0): \quad \hat{A}_t := \delta_t \qquad = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{27}$$

$$\mathrm{GAE}(\gamma, 1): \quad \hat{A}_t := \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V(s_t) \tag{28}$$

$\mathrm{GAE}(\gamma, 1)$ is γ-just regardless of the accuracy of V, but it has high variance due to the sum of terms. $\mathrm{GAE}(\gamma, 0)$ is γ-just for $V = V^{\pi,\gamma}$ and otherwise induces bias, but it typically has much lower variance. The generalized advantage estimator for $0 < \lambda < 1$ makes a compromise between bias and variance, controlled by parameter λ.

We've described an advantage estimator with two separate parameters γ and λ, both of which contribute to the bias-variance tradeoff when using an approximate value function. However, they serve different purposes and work best with different ranges of values. γ most importantly determines the scale of the value function $V^{\pi,\gamma}$, which does not depend on λ. Taking $\gamma < 1$ introduces bias into the policy gradient estimate, regardless of the value function's accuracy. On the other hand, $\lambda < 1$ introduces bias only when the value function is inaccurate. Empirically, we find that the best value of λ is much lower than the best value of γ, likely because λ introduces far less bias than γ for a reasonably accurate value function.

Using the generalized advantage estimator, we can construct a biased estimator of $\mathbf{g}^{\gamma}$, the discounted policy gradient from Equation (23):

$$\mathbf{g}^{\gamma} \approx \mathbb{E}\left[\sum_{t=0}^{\infty} \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t) \hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)}\right] = \mathbb{E}\left[\sum_{t=0}^{\infty} \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t) \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V\right], \quad (29)$$

where equality holds when $\lambda = 1$.

4.4 INTERPRETATION AS REWARD SHAPING

In this section, we discuss how one can interpret λ as an extra discount factor applied after performing a reward shaping transformation on the MDP. We also introduce the notion of a response function to help understand the bias introduced by γ and λ.

Reward shaping [NHR99] refers to the following transformation of the reward function of an MDP: let $\Phi : \mathcal{S} \to \mathbb{R}$ be an arbitrary scalar-valued function on state space, and define the transformed reward function $\tilde{r}$ by

$$\tilde{r}(s, a, s') = r(s, a, s') + \gamma\Phi(s') - \Phi(s), \quad (30)$$

which in turn defines a transformed MDP. This transformation leaves the discounted advantage function $A^{\pi,\gamma}$ unchanged for any policy π. To see this, consider the discounted sum of rewards of a trajectory starting with state s_t:

$$\sum_{l=0}^{\infty} \gamma^l \tilde{r}(s_{t+l}, a_t, s_{t+l+1}) = \sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, a_{t+l}, s_{t+l+1}) - \Phi(s_t). \quad (31)$$

Letting $\tilde{Q}^{\pi,\gamma}, \tilde{V}^{\pi,\gamma}, \tilde{A}^{\pi,\gamma}$ be the value and advantage functions of the transformed MDP, one obtains from the definitions of these quantities that

$$
\begin{aligned}
&\tilde{Q}^{\pi,\gamma}(s,a) = Q^{\pi,\gamma}(s,a) - \Phi(s)\\
&\tilde{V}^{\pi,\gamma}(s,a) = V^{\pi,\gamma}(s,a) - \Phi(s)\\
&\tilde{A}^{\pi,\gamma}(s,a) = (Q^{\pi,\gamma}(s,a) - \Phi(s)) - (V^{\pi}(s) - \Phi(s)) = A^{\pi,\gamma}(s,a).
\end{aligned}
$$

Note that if Φ happens to be the state-value function $V^{\pi,\gamma}$ from the original MDP, then the transformed MDP has the interesting property that $\tilde{V}^{\pi,\gamma}(s)$ is zero at every state.

Note that [NHR99] showed that the reward shaping transformation leaves the policy gradient and optimal policy unchanged when our objective is to maximize the discounted sum of rewards $\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t,s_{t+1})$. In contrast, this chapter is concerned with maximizing the undiscounted sum of rewards, where the discount γ is used as a variance-reduction parameter.

Having reviewed the idea of reward shaping, let us consider how we could use it to get a policy gradient estimate. The most natural approach is to construct policy gradient estimators that use discounted sums of shaped rewards $\tilde{r}$. However, Equation (31) shows that we obtain the discounted sum of the original MDP's rewards r minus a baseline term. Next, let's consider using a "steeper" discount $\gamma\lambda$, where $0 \leqslant \lambda \leqslant 1$. It's easy to see that the shaped reward $\tilde{r}$ equals the Bellman residual term δ^V, introduced in Section 4.3, where we set $\Phi = V$. Letting $\Phi = V$, we see that

$$
\sum_{l=0}^{\infty}(\gamma\lambda)^l \tilde{r}(s_{t+l},a_t,s_{t+l+1}) = \sum_{l=0}^{\infty}(\gamma\lambda)^l \delta^V_{t+l} = \hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)}.
$$

Hence, by considering the $\gamma\lambda$-discounted sum of shaped rewards, we exactly obtain the generalized advantage estimators from Section 4.3. As shown previously, $\lambda = 1$ gives an unbiased estimate of $\mathbf{g}^\gamma$, whereas $\lambda < 0$ gives a biased estimate.

To further analyze the effect of this shaping transformation and parameters γ and λ, it will be useful to introduce the notion of a response function χ, which we define as follows:

$$
\chi(l;s_t,a_t) = \mathbb{E}\left[r_{t+l} \mid s_t,a_t\right] - \mathbb{E}\left[r_{t+l} \mid s_t\right].
$$

Note that $A^{\pi,\gamma}(s,a) = \sum_{l=0}^{\infty}\gamma^l\chi(l;s,a)$, hence the response function decomposes the advantage function across timesteps. The response function lets us quantify the temporal credit assignment problem: long range dependencies between actions and rewards correspond to nonzero values of the response function for $l \gg 0$.

Next, let us revisit the discount factor γ and the approximation we are making by using $A^{\pi,\gamma}$ rather than $A^{\pi,1}$. The discounted policy gradient estimator from Equation (23) has a sum of terms of the form

$$\nabla_\theta \log \pi_\theta(a_t \mid s_t) A^{\pi,\gamma}(s_t, a_t) = \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{l=0}^{\infty} \gamma^l \chi(l; s, a).$$

Using a discount $\gamma < 1$ corresponds to dropping the terms with $l \gg 1/(1-\gamma)$. Thus, the error introduced by this approximation will be small if χ rapidly decays as l increases, i.e., if the effect of an action on rewards is "forgotten" after $\approx 1/(1-\gamma)$ timesteps.

If the reward function $\tilde{r}$ were obtained using $\Phi = V^{\pi,\gamma}$, we would have $\mathbb{E}\left[\tilde{r}_{t+l} \mid s_t, a_t\right] = \mathbb{E}\left[\tilde{r}_{t+l} \mid s_t\right] = 0$ for $l > 0$, i.e., the response function would only be nonzero at $l = 0$. Therefore, this shaping transformation would turn temporally extended response into an immediate response. Given that $V^{\pi,\gamma}$ completely reduces the temporal spread of the response function, we can hope that a good approximation $V \approx V^{\pi,\gamma}$ partially reduces it. This observation suggests an interpretation of Equation (26): reshape the rewards using V to shrink the temporal extent of the response function, and then introduce a "steeper" discount $\gamma\lambda$ to cut off the noise arising from long delays, i.e., ignore terms $\nabla_\theta \log \pi_\theta(a_t \mid s_t) \delta^V_{t+l}$ where $l \gg 1/(1-\gamma\lambda)$.

4.5 VALUE FUNCTION ESTIMATION

A variety of different methods can be used to estimate the value function (see, e.g., [Ber12]). When using a nonlinear function approximator to represent the value function, the simplest approach is to solve a nonlinear regression problem:

$$\underset{\phi}{\text{minimize}} \sum_{n=1}^{N} \|V_\phi(s_n) - \hat{V}_n\|^2, \tag{32}$$

where $\hat{V}_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l}$ is the discounted sum of rewards, and n indexes over all timesteps in a batch of trajectories. This is sometimes called the Monte Carlo or TD(1) approach for estimating the value function [SB98].[2]

[2] Another natural choice is to compute target values with an estimator based on the TD(λ) backup [Ber12; SB98], mirroring the expression we use for policy gradient estimation: $\hat{V}^\lambda_t = V_{\phi_{\text{old}}}(s_n) + \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$. While we experimented with this choice, we did not notice a difference in performance from the $\lambda = 1$ estimator in Equation (32).

For the experiments in this work, we used a trust region method to optimize the value function in each iteration of a batch optimization procedure. The trust region helps us to avoid overfitting to the most recent batch of data. To formulate the trust region problem, we first compute $\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}\|V_{\phi_{\text{old}}}(s_n) - \hat{V}_n\|^2$, where ϕ_{old} is the parameter vector before optimization. Then we solve the following constrained optimization problem:

$$\begin{aligned} &\underset{\phi}{\text{minimize}} && \sum_{n=1}^{N}\|V_\phi(s_n) - \hat{V}_n\|^2 \\ &\text{subject to} && \frac{1}{N}\sum_{n=1}^{N}\frac{\|V_\phi(s_n) - V_{\phi_{\text{old}}}(s_n)\|^2}{2\sigma^2} \leqslant \epsilon. \end{aligned} \tag{33}$$

This constraint is equivalent to constraining the average KL divergence between the previous value function and the new value function to be smaller than ϵ, where the value function is taken to parameterize a conditional Gaussian distribution with mean $V_\phi(s)$ and variance σ^2.

We compute an approximate solution to the trust region problem using the conjugate gradient algorithm [WN99]. Specifically, we are solving the quadratic program

$$\begin{aligned} &\underset{\phi}{\text{minimize}} && g^T(\phi - \phi_{\text{old}}) \\ &\text{subject to} && \frac{1}{N}\sum_{n=1}^{N}(\phi - \phi_{\text{old}})^T H(\phi - \phi_{\text{old}}) \leqslant \epsilon. \end{aligned} \tag{34}$$

where g is the gradient of the objective, and $H = \frac{1}{N}\sum_n j_n j_n^T$, where $j_n = \nabla_\phi V_\phi(s_n)$. Note that H is the "Gauss-Newton" approximation of the Hessian of the objective, and it is (up to a σ^2 factor) the Fisher information matrix when interpreting the value function as a conditional probability distribution. Using matrix-vector products $v \to Hv$ to implement the conjugate gradient algorithm, we compute a step direction $s \approx -H^{-1}g$. Then we rescale $s \to \alpha s$ such that $\frac{1}{2}(\alpha s)^T H(\alpha s) = \epsilon$ and take $\phi = \phi_{\text{old}} + \alpha s$. This procedure is analogous to the procedure we use for updating the policy, which is described further in Section 4.6 and based on [Sch+15c].

4.6 experiments

We designed a set of experiments to investigate the following questions:

1. What is the empirical effect of varying $\lambda \in [0, 1]$ and $\gamma \in [0, 1]$ when optimizing episodic total reward using generalized advantage estimation?

2. Can generalized advantage estimation, along with trust region algorithms for policy and value function optimization, be used to optimize large neural network policies for challenging control problems?

4.6.1 *Policy Optimization Algorithm*

While generalized advantage estimation can be used along with a variety of different policy gradient methods, for these experiments, we performed the policy updates using trust region policy optimization (TRPO) [Sch+15c]. TRPO updates the policy by approximately solving the following constrained optimization problem each iteration:

$$\begin{aligned}
&\underset{\theta}{\text{minimize}}\, L_{\theta_{\text{old}}}(\theta) \\
&\text{subject to } \overline{D}_{\text{KL}}^{\theta_{\text{old}}}(\pi_{\theta_{\text{old}}}, \pi_\theta) \leqslant \epsilon \\
&\text{where } L_{\theta_{\text{old}}}(\theta) = \frac{1}{N}\sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n \\
&\qquad \overline{D}_{\text{KL}}^{\theta_{\text{old}}}(\pi_{\theta_{\text{old}}}, \pi_\theta) = \frac{1}{N}\sum_{n=1}^{N} D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot \mid s_n) \parallel \pi_\theta(\cdot \mid s_n))
\end{aligned} \tag{35}$$

As described in [Sch+15c], we approximately solve this problem by linearizing the objective and quadraticizing the constraint, which yields a step in the direction $\theta - \theta_{\text{old}} \propto -F^{-1}g$, where F is the average Fisher information matrix, and g is a policy gradient estimate. This policy update yields the same step direction as the natural policy gradient [Kak01a] and natural actor-critic [PS08], however it uses a different stepsize determination scheme and numerical procedure for computing the step.

Since prior work [Sch+15c] compared TRPO to a variety of different policy optimization algorithms, we will not repeat these comparisons; rather, we will focus on varying the γ, λ parameters of policy gradient estimator while keeping the underlying algorithm fixed.

For completeness, the whole algorithm for iteratively updating policy and value function is given below:

Note that the policy update $\theta_i \to \theta_{i+1}$ is performed using the value function V_{ϕ_i} for advantage estimation, not $V_{\phi_{i+1}}$. Additional bias would have been introduced if we updated the value function first. To see this, consider the extreme case where we overfit the value function, and the Bellman residual $r_t + \gamma V(s_{t+1}) - V(s_t)$ becomes zero at all timesteps—the policy gradient estimate would be zero.

```
Initialize policy parameter $\theta_0$ and value function parameter $\phi_0$.
for $i = 0, 1, 2, \ldots$ do
    Simulate current policy $\pi_{\theta_i}$ until $N$ timesteps are obtained.
    Compute $\delta_t^V$ at all timesteps $t \in \{1, 2, \ldots, N\}$, using $V = V_{\phi_i}$.
    Compute $\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V$ at all timesteps.
    Compute $\theta_{i+1}$ with TRPO update, Equation (35).
    Compute $\phi_{i+1}$ with Equation (34).
end for
```

4.6.2 *Experimental Setup*

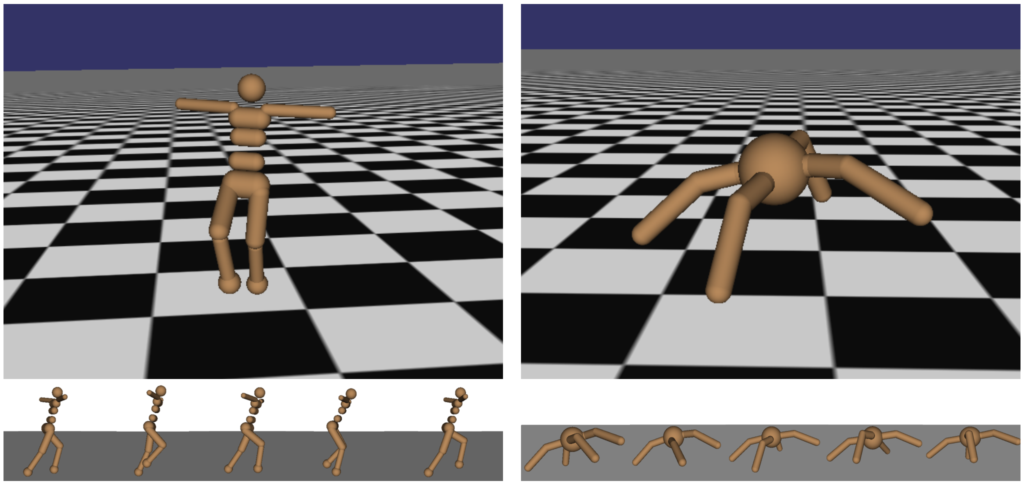


Figure 7: Top figures: robot models used for 3D locomotion. Bottom figures: a sequence of frames from the learned gaits. Videos are available at `https://sites.google.com/site/gaepapersupp`.

We evaluated our approach on the classic cart-pole balancing problem, as well as several challenging 3D locomotion tasks: (1) bipedal locomotion; (2) quadrupedal locomotion; (3) dynamically standing up, for the biped, which starts off laying on its back. The models are shown in Figure 7.

Architecture

We used the same neural network architecture for all of the 3D robot tasks, which was a feedforward network with three hidden layers, with 100, 50 and 25 tanh units respectively. The same architecture was used for the policy and value function. The final output layer had linear activation. The value function estimator used the same architecture, but with only one scalar output. For the simpler cart-pole task, we used a linear policy, and a neural network with one 20-unit hidden layer as the value function.

Task details

For the cart-pole balancing task, we collected 20 trajectories per batch, with a maximum length of 1000 timesteps, using the physical parameters from Barto, Sutton, and Anderson [BSA83].

The simulated robot tasks were simulated using the MuJoCo physics engine [TET12]. The humanoid model has 33 state dimensions and 10 actuated degrees of freedom, while the quadruped model has 29 state dimensions and 8 actuated degrees of freedom. The initial state for these tasks consisted of a uniform distribution centered on a reference configuration. We used 50000 timesteps per batch for bipedal locomotion, and 200000 timesteps per batch for quadrupedal locomotion and bipedal standing. Each episode was terminated after 2000 timesteps if the robot had not reached a terminal state beforehand. The timestep was 0.01 seconds.

The reward functions are provided in the table below.

Task	Reward
3D biped locomotion	$v_{\text{fwd}} - 10^{-5}\|u\|^2 - 10^{-5}\|f_{\text{impact}}\|^2 + 0.2$
Quadruped locomotion	$v_{\text{fwd}} - 10^{-6}\|u\|^2 - 10^{-3}\|f_{\text{impact}}\|^2 + 0.05$
Biped getting up	$-(h_{\text{head}} - 1.5)^2 - 10^{-5}\|u\|^2$

Here, $v_{\text{fwd}} :=$ forward velocity, $u :=$ vector of joint torques, $f_{\text{impact}} :=$ impact forces, $h_{\text{head}} :=$ height of the head.

In the locomotion tasks, the episode is terminated if the center of mass of the actor falls below a predefined height: .8 m for the biped, and .2 m for the quadruped. The constant offset in the reward function encourages longer episodes; otherwise the quadratic reward terms might lead lead to a policy that ends the episodes as quickly as possible.

4.6.3 *Experimental Results*

All results are presented in terms of the cost, which is defined as negative reward and is minimized. Videos of the learned policies are available at https://sites.google.com/site/gaepapersupp. In plots, "No VF" means that we used a time-dependent baseline that did not depend on the state, rather than an estimate of the state value function. The time-dependent baseline was computed by averaging the return at each timestep over the trajectories in the batch.

Cart-pole

The results are averaged across 21 experiments with different random seeds. Results are shown in Figure 8, and indicate that the best results are obtained at intermediate values of the parameters: $\gamma \in [0.96, 0.99]$ and $\lambda \in [0.92, 0.99]$.

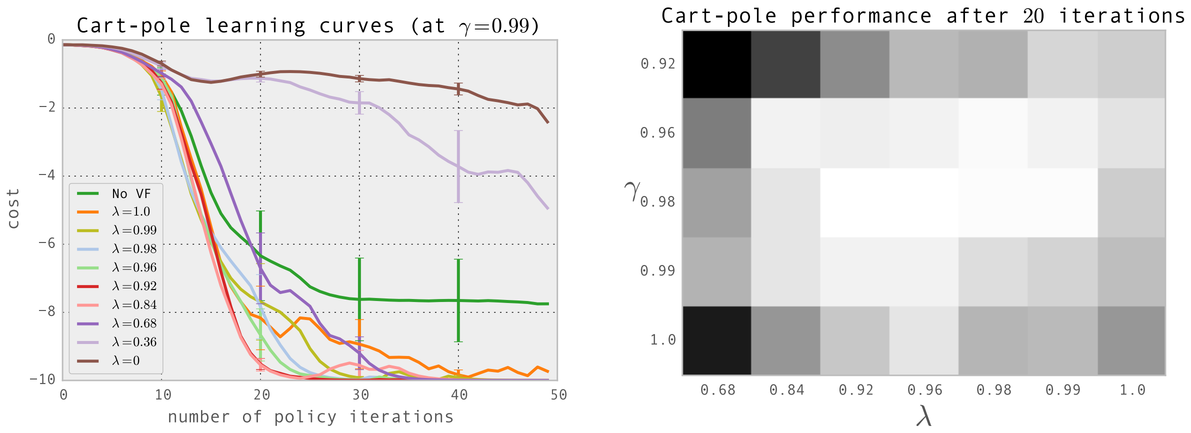


Figure 8: Left: learning curves for cart-pole task, using generalized advantage estimation with varying values of λ at $\gamma = 0.99$. The fastest policy improvement is obtain by intermediate values of λ in the range $[0.92, 0.98]$. Right: performance after 20 iterations of policy optimization, as γ and λ are varied. White means higher reward. The best results are obtained at intermediate values of both.

3D bipedal locomotion

Each trial took about 2 hours to run on a 16-core machine, where the simulation rollouts were parallelized, as were the function, gradient, and matrix-vector-product evaluations used when optimizing the policy and value function. Here, the results are averaged across 9 trials with different random seeds. The best performance is again obtained using intermediate values of $\gamma \in [0.99, 0.995], \lambda \in [0.96, 0.99]$. The result after 1000 iterations is a fast, smooth, and stable gait that is effectively completely stable. We can compute how much "real time" was used for this learning process: $0.01\,\text{seconds}/\text{timestep} \times 50000\,\text{timesteps}/\text{batch} \times 1000\,\text{batches}/3600 \cdot 24\,\text{seconds}/\text{day} = 5.8\,\text{days}$. Hence, it is plausible

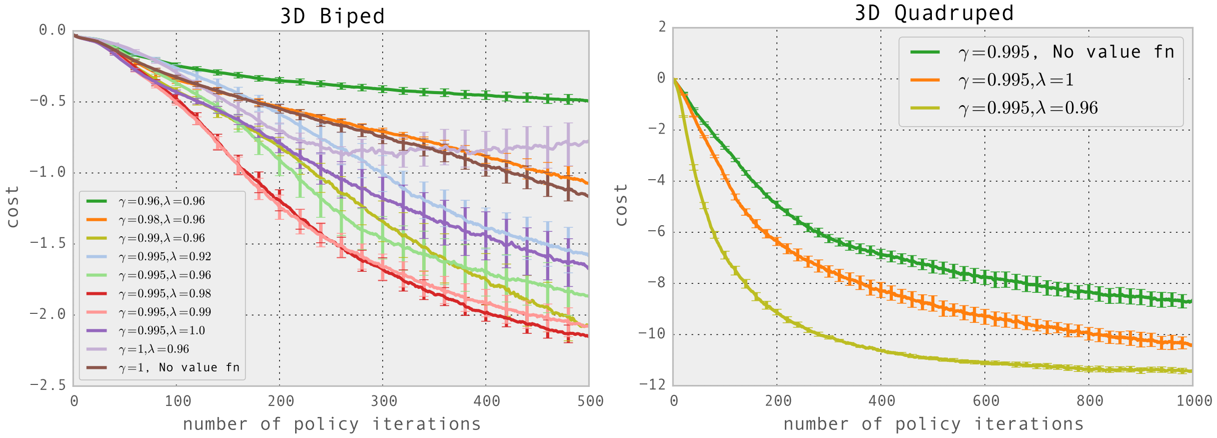


Figure 9: Left: Learning curves for 3D bipedal locomotion, averaged across nine runs of the algorithm. Right: learning curves for 3D quadrupedal locomotion, averaged across five runs.

that this algorithm could be run on a real robot, or multiple real robots learning in parallel, if there were a way to reset the state of the robot and ensure that it doesn't damage itself.

Other 3D robot tasks

The other two motor behaviors considered are quadrupedal locomotion and getting up off the ground for the 3D biped. Again, we performed 5 trials per experimental condition, with different random seeds (and initializations). The experiments took about 4 hours per trial on a 32-core machine. We performed a more limited comparison on these domains (due to the substantial computational resources required to run these experiments), fixing $\gamma = 0.995$ but varying $\lambda = \{0, 0.96\}$, as well as an experimental condition with no value function. For quadrupedal locomotion, the best results are obtained using a value function with $\lambda = 0.96$ Section 4.6.3. For 3D standing, the value function always helped, but the results are roughly the same for $\lambda = 0.96$ and $\lambda = 1$.

4.7 DISCUSSION

Policy gradient methods provide a way to reduce reinforcement learning to stochastic gradient descent, by providing unbiased gradient estimates. However, so far their success at solving difficult control problems has been limited, largely due to their high sample

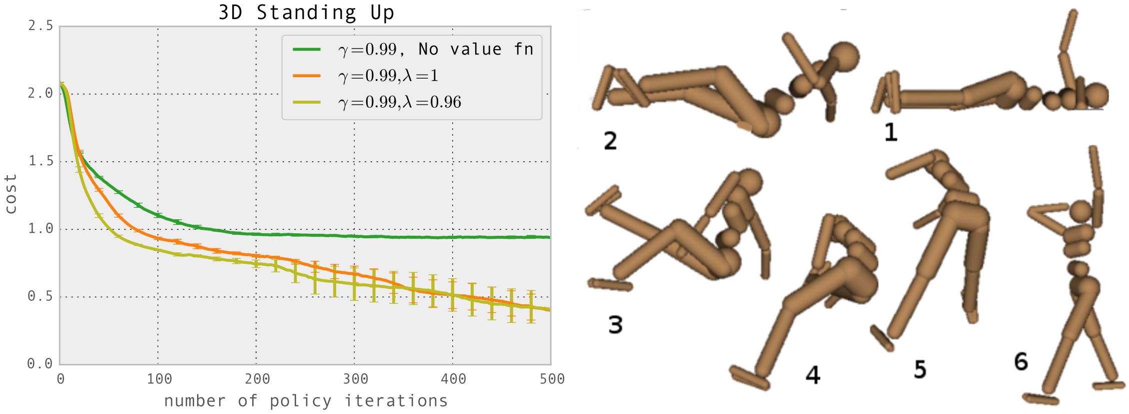


Figure 10: Left: curve from 3D standing, right: clips from 3D standing up.

complexity. We have argued that the key to variance reduction is to obtain good estimates of the advantage function.

We have provided an intuitive but informal analysis of the problem of advantage function estimation, and justified the generalized advantage estimator, which has two parameters γ, λ which adjust the bias-variance tradeoff. We described how to combine this idea with trust region policy optimization and a trust region algorithm that optimizes a value function, both represented by neural networks. Combining these techniques, we are able to learn to solve difficult control tasks that have previously been out of reach for generic reinforcement learning methods.

Our main experimental validation of generalized advantage estimation is in the domain of simulated robotic locomotion. In these domains, the $\lambda = 0$ As shown in our experiments, choosing an appropriate intermediate value of λ in the range $[0.9, 0.99]$ usually results in the best performance. A possible topic for future work is how to adjust the estimator parameters γ, λ in an adaptive or automatic way.

One question that merits future investigation is the relationship between value function estimation error and policy gradient estimation error. If this relationship were known, we could choose an error metric for value function fitting that is well-matched to the quantity of interest, which is typically the accuracy of the policy gradient estimation. Some candidates for such an error metric might include the Bellman error or projected Bellman error, as described in [Bha+09].

Another enticing possibility is to use a shared function approximation architecture for the policy and the value function, while optimizing the policy using generalized advantage estimation. While formulating this problem in a way that is suitable for numerical optimization and provides convergence guarantees remains an open question, such an approach could allow the value function and policy representations to share useful features of the input, resulting in even faster learning.

In concurrent work, researchers have been developing policy gradient methods that involve differentiation with respect to the continuous-valued action [Lil+15; Hee+15]. While we found empirically that the one-step return ($\lambda = 0$) leads to excessive bias and poor performance, those papers show that such methods can work when tuned appropriately. However, note that those papers consider control problems with substantially lower-dimensional state and action spaces than the ones considered here. A comparison between both classes of approach would be useful for future work.

4.8 FREQUENTLY ASKED QUESTIONS

4.8.1 *What's the Relationship with Compatible Features?*

Compatible features are often mentioned in relation to policy gradient algorithms that make use of a value function, and the idea was proposed in the paper *On Actor-Critic Methods* by Konda and Tsitsiklis [KT03]. These authors pointed out that due to the limited representation power of the policy, the policy gradient only depends on a certain subspace of the space of advantage functions. This subspace is spanned by the compatible features $\nabla_{\theta_i} \log \pi_\theta(a_t \mid s_t)$, where $i \in \{1, 2, \dots, \dim \theta\}$. This theory of compatible features provides no guidance on how to exploit the temporal structure of the problem to obtain better estimates of the advantage function, making it mostly orthogonal to the ideas in this chapter.

The idea of compatible features motivates an elegant method for computing the natural policy gradient [Kak01a; PS08]. Given an empirical estimate of the advantage function $\hat{A}_t$ at each timestep, we can project it onto the subspace of compatible features by solving the following least squares problem:

$$\underset{\mathbf{r}}{\text{minimize}} \sum_t \|\mathbf{r} \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t) - \hat{A}_t\|^2.$$

If $\hat{A}$ is γ-just, the least squares solution is the natural policy gradient [Kak01a]. Note that any estimator of the advantage function can be substituted into this formula, including the ones we derive in this paper. For our experiments, we also compute natural policy gradient steps, but we use the more computationally efficient numerical procedure from [Sch+15c], as discussed in Section 4.6.

4.8.2 *Why Don't You Just Use a Q-Function?*

Previous actor critic methods, e.g. in [KT03], use a Q-function to obtain potentially low-variance policy gradient estimates. Recent papers, including [Hee+15; Lil+15], have shown that a neural network Q-function approximator can used effectively in a policy gradient method. However, there are several advantages to using a state-value function in the manner of this paper. First, the state-value function has a lower-dimensional input and is thus easier to learn than a state-action value function. Second, the method of this paper allows us to smoothly interpolate between the high-bias estimator ($\lambda = 0$) and the low-bias estimator ($\lambda = 1$). On the other hand, using a parameterized Q-function only allows us to use a high-bias estimator. We have found that the bias is prohibitively large when using a one-step estimate of the returns, i.e., the $\lambda = 0$ estimator, $\hat{A}_t = \delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$. We expect that similar difficulty would be encountered when using an advantage estimator involving a parameterized Q-function, $\hat{A}_t = Q(s, a) - V(s)$. There is an interesting space of possible algorithms that would use a parameterized Q-function and attempt to reduce bias, however, an exploration of these possibilities is beyond the scope of this work.

4.9 PROOFS

Proof of Proposition 1: First we can split the expectation into terms involving Q and b,

$$\begin{aligned}
&\mathbb{E}_{s_{0:\infty},a_{0:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)(Q_t(s_{0:\infty}, a_{0:\infty}) + b_t(s_{0:t}, a_{0:t-1}))\right] \\
&\quad = \mathbb{E}_{s_{0:\infty},a_{0:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)(Q_t(s_{0:\infty}, a_{0:\infty}))\right] \\
&\quad\quad + \mathbb{E}_{s_{0:\infty},a_{0:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)(b_t(s_{0:t}, a_{0:t-1}))\right]
\end{aligned}$$

We'll consider the terms with Q and b in turn.

$$\begin{aligned}
&\mathbb{E}_{s_{0:\infty},a_{0:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) Q_t(s_{0:\infty}, a_{0:\infty})\right] \\
&\quad = \mathbb{E}_{s_{0:t},a_{0:t}}\left[\mathbb{E}_{s_{t+1:\infty},a_{t+1:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) Q_t(s_{0:\infty}, a_{0:\infty})\right]\right] \\
&\quad = \mathbb{E}_{s_{0:t},a_{0:t}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \mathbb{E}_{s_{t+1:\infty},a_{t+1:\infty}}\left[Q_t(s_{0:\infty}, a_{0:\infty})\right]\right] \\
&\quad = \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) A^\pi(s_t, a_t)\right]
\end{aligned}$$

Next,

$$\begin{aligned}
&\mathbb{E}_{s_{0:\infty},a_{0:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) b_t(s_{0:t}, a_{0:t-1})\right] \\
&\quad = \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\mathbb{E}_{s_{t+1:\infty},a_{t:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) b_t(s_{0:t}, a_{0:t-1})\right]\right] \\
&\quad = \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\mathbb{E}_{s_{t+1:\infty},a_{t:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] b_t(s_{0:t}, a_{0:t-1})\right] \\
&\quad = \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[0 \cdot b_t(s_{0:t}, a_{0:t-1})\right] \\
&\quad = 0.
\end{aligned}$$

5

STOCHASTIC COMPUTATION GRAPHS

5.1 OVERVIEW

The great success of neural networks is due in part to the simplicity of the backpropagation algorithm, which allows one to efficiently compute the gradient of any loss function defined as a composition of differentiable functions. This simplicity has allowed researchers to search in the space of architectures for those that are both highly expressive and conducive to optimization; yielding, for example, convolutional neural networks in vision [LeC+98] and LSTMs for sequence data [HS97]. However, the backpropagation algorithm is only sufficient when the loss function is a deterministic, differentiable function of the parameter vector.

A rich class of problems arising throughout machine learning requires optimizing loss functions that involve an expectation over random variables. Two broad categories of these problems are (1) likelihood maximization in probabilistic models with latent variables [Nea90; NH98], and (2) policy gradients in reinforcement learning [Gly90; Sut+99; Wil92]. Combining ideas from from those two perennial topics, recent models of attention [Mni+14] and memory [ZS15] have used networks that involve a combination of stochastic and deterministic operations.

In most of these problems, from probabilistic modeling to reinforcement learning, the loss functions and their gradients are intractable, as they involve either a sum over an exponential number of latent variable configurations, or high-dimensional integrals that have no analytic solution. Prior work (see Section 5.6) has provided problem-specific derivations of Monte-Carlo gradient estimators, however, to our knowledge, no previous work addresses the general case.

Section 5.10 recalls several classic and recent techniques in variational inference [MG14; KW13; RMW14] and reinforcement learning [Sut+99; Wie+10; Mni+14], where the loss functions can be straightforwardly described using the formalism of stochastic computation graphs that we introduce. For these examples, the variance-reduced gradient estimators derived in prior work are special cases of the results in Sections 5.3 and 5.4.

The contributions of this chapter are as follows:

- We introduce a formalism of stochastic computation graphs, and in this general setting, we derive unbiased estimators for the gradient of the expected loss.
- We show how this estimator can be computed as the gradient of a certain differentiable function (which we call the *surrogate loss*), hence, it can be computed efficiently using the backpropagation algorithm. This observation enables a practitioner to write an efficient implementation using automatic differentiation software.
- We describe variance reduction techniques that can be applied to the setting of stochastic computation graphs, generalizing prior work from reinforcement learning and variational inference.
- We briefly describe how to generalize some other optimization techniques to this setting: majorization-minimization algorithms, by constructing an expression that bounds the loss function; and quasi-Newton / Hessian-free methods [Mar10], by computing estimates of Hessian-vector products.

The main practical result of this chapter is that to compute the gradient estimator, one just needs to make a simple modification to the backpropagation algorithm, where extra gradient signals are introduced at the stochastic nodes. Equivalently, the resulting algorithm is *just* the backpropagation algorithm, applied to the surrogate loss function, which has extra terms introduced at the stochastic nodes. The modified backpropagation algorithm is presented in Section 5.5.

5.2 preliminaries

5.2.1 *Gradient Estimators for a Single Random Variable*

This section will discuss computing the gradient of an expectation taken over a single random variable—the estimators described here will be the building blocks for more complex cases with multiple variables. Suppose that x is a random variable, f is a function (say, the cost), and we are interested in computing $\frac{\partial}{\partial\theta}\mathbb{E}_x\left[f(x)\right]$. There are a few different ways that the process for generating x could be parameterized in terms of θ, which lead to different gradient estimators.

- We might be given a parameterized probability distribution $x \sim p(\cdot;\ \theta)$. In this case, we can use the *score function* (SF) estimator [Fu06]:

$$\frac{\partial}{\partial\theta}\mathbb{E}_x\left[f(x)\right] = \mathbb{E}_x\left[f(x)\frac{\partial}{\partial\theta}\log p(x;\ \theta)\right]. \tag{36}$$

This classic equation is derived as follows:

$$\begin{aligned}\frac{\partial}{\partial\theta}\mathbb{E}_x\left[f(x)\right] &= \frac{\partial}{\partial\theta}\int dx\ p(x;\ \theta)f(x) = \int dx\ \frac{\partial}{\partial\theta}p(x;\ \theta)f(x)\\ &= \int dx\ p(x;\ \theta)\frac{\partial}{\partial\theta}\log p(x;\ \theta)f(x) = \mathbb{E}_x\left[f(x)\frac{\partial}{\partial\theta}\log p(x;\ \theta)\right].\end{aligned} \tag{37}$$

This equation is valid if and only if $p(x;\ \theta)$ is a continuous function of θ; however, it does not need to be a continuous function of x [Gla03].

- x may be a deterministic, differentiable function of θ and another random variable z, i.e., we can write $x(z,\theta)$. Then, we can use the *pathwise derivative* (PD) estimator, defined as follows.

$$\frac{\partial}{\partial\theta}\mathbb{E}_z\left[f(x(z,\theta))\right] = \mathbb{E}_z\left[\frac{\partial}{\partial\theta}f(x(z,\theta))\right].$$

This equation, which merely swaps the derivative and expectation, is valid if and only if $f(x(z,\theta))$ is a continuous function of θ for all z [Gla03]. [1] That is not true if, for example, f is a step function.

- Finally θ might appear both in the probability distribution and inside the expectation, e.g., in $\frac{\partial}{\partial\theta}\mathbb{E}_{z\sim p(\cdot;\ \theta)}\left[f(x(z,\theta))\right]$. Then the gradient estimator has two terms:

$$\frac{\partial}{\partial\theta}\mathbb{E}_{z\sim p(\cdot;\ \theta)}\left[f(x(z,\theta))\right] = \mathbb{E}_{z\sim p(\cdot;\ \theta)}\left[\frac{\partial}{\partial\theta}f(x(z,\theta)) + \left(\frac{\partial}{\partial\theta}\log p(z;\theta)\right)f(x(z,\theta))\right].$$

This formula can be derived by writing the expectation as an integral and differentiating, as in Equation (37).

In some cases, it is possible to *reparameterize* a probabilistic model—moving θ from the distribution to inside the expectation or vice versa. See [Fu06] for a general discussion, and see [KW13; RMW14] for a recent application of this idea to variational inference.

The SF and PD estimators are applicable in different scenarios and have different properties.

[1] Note that for the pathwise derivative estimator, $f(x(z,\theta))$ merely needs to be a *continuous* function of θ—it is sufficient that this function is almost-everywhere differentiable. A similar statement can be made about $p(x;\theta)$ and the score function estimator. See Glasserman [Gla03] for a detailed discussion of the technical requirements for these gradient estimators to be valid.

1. SF is valid under more permissive mathematical conditions than PD. SF can be used if f is discontinuous, or if x is a discrete random variable.
2. SF only requires sample values $f(x)$, whereas PD requires the derivatives $f'(x)$. In the context of control (reinforcement learning), SF can be used to obtain unbiased policy gradient estimators in the "model-free" setting where we have no model of the dynamics, we only have access to sample trajectories.
3. SF tends to have higher variance than PD, when both estimators are applicable (see for instance [Fu06; RMW14]). The variance of SF increases (often linearly) with the dimensionality of the sampled variables. Hence, PD is usually preferable when x is high-dimensional. On the other hand, PD has high variance if the function f is rough, which occurs in many time-series problems due to an "exploding gradient problem" / "butterfly effect".
4. PD allows for a deterministic limit, SF does not. This idea is exploited by the deterministic policy gradient algorithm [Sil+14].

NOMENCLATURE. The methods of estimating gradients of expectations have been independently proposed in several different fields, which use differing terminology. What we call the *score function* estimator (via [Fu06]) is alternatively called the *likelihood ratio* estimator [Gly90] and REINFORCE [Wil92]. We chose this term because the score function is a well-known object in statistics. What we call the *pathwise derivative* estimator (from the mathematical finance literature [Gla03] and reinforcement learning [Mun06]) is alternatively called *infinitesimal perturbation analysis* and *stochastic backpropagation* [RMW14]. We chose this term because pathwise derivative is evocative of propagating a derivative through a sample path.

5.2.2 *Stochastic Computation Graphs*

The results of this chapter will apply to stochastic computation graphs, which are defined as follows:

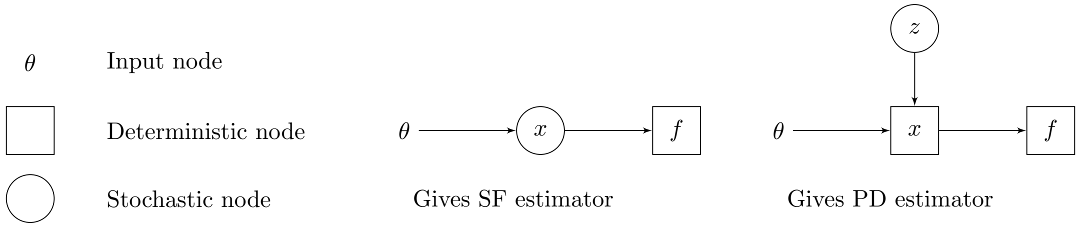
Definition 3 (Stochastic Computation Graph). *A directed, acyclic graph, with three types of nodes:*

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

3. Stochastic nodes, which are distributed conditionally on their parents.

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

In the subsequent diagrams of this chapter, we will use circles to denote stochastic nodes and squares to denote deterministic nodes, as illustrated below. The structure of the graph fully specifies what estimator we will use: SF, PD, or a combination thereof. This graphical notation is shown below, along with the single-variable estimators from Section 5.2.1.



5.2.3 *Simple Examples*

Several simple examples that illustrate the stochastic computation graph formalism are shown below. The gradient estimators can be described by writing the expectations as integrals and differentiating, as with the simpler estimators from Section 5.2.1. However, they are also implied by the general results that we will present in Section 5.3.

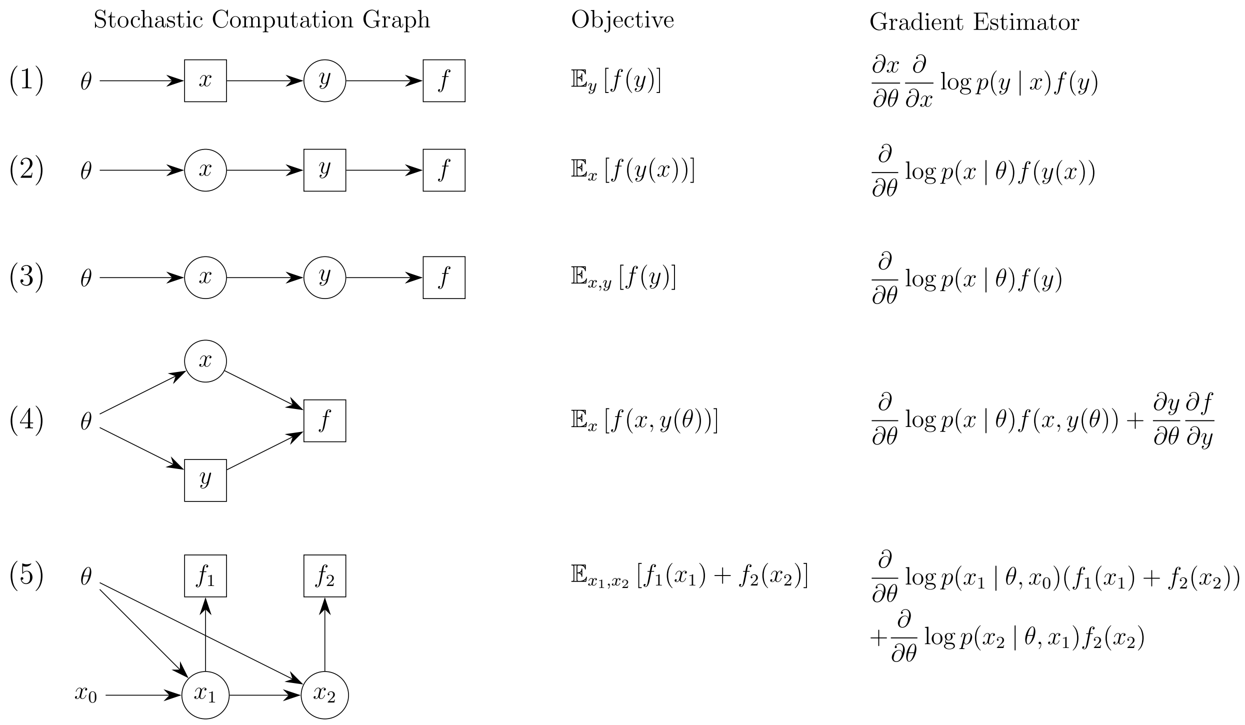
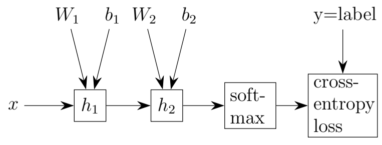

Stochastic Computation Graph
Objective
Gradient Estimator
(1)
θ
x
y
f
$\mathbb{E}_y\left[f(y)\right]$
$\frac{\partial x}{\partial \theta}\frac{\partial}{\partial x}\log p(y \mid x) f(y)$
(2)
θ
x
y
f
$\mathbb{E}_x\left[f(y(x))\right]$
$\frac{\partial}{\partial \theta}\log p(x \mid \theta) f(y(x))$
(3)
θ
x
y
f
$\mathbb{E}_{x,y}\left[f(y)\right]$
$\frac{\partial}{\partial \theta}\log p(x \mid \theta) f(y)$
(4)
x
θ
f
y
$\mathbb{E}_x\left[f(x, y(\theta))\right]$
$\frac{\partial}{\partial \theta}\log p(x \mid \theta) f(x, y(\theta)) + \frac{\partial y}{\partial \theta}\frac{\partial f}{\partial y}$
(5)
θ
f_1
f_2
x_0
x_1
x_2
$\mathbb{E}_{x_1,x_2}\left[f_1(x_1) + f_2(x_2)\right]$
$\frac{\partial}{\partial \theta}\log p(x_1 \mid \theta, x_0)(f_1(x_1) + f_2(x_2))$
$+\frac{\partial}{\partial \theta}\log p(x_2 \mid \theta, x_1) f_2(x_2)$

Figure 11: Simple stochastic computation graphs

These simple examples illustrate several important motifs, where stochastic and deterministic nodes are arranged in series or in parallel. For example, note that in (2) the derivative of y does not appear in the estimator, since the path from θ to f is "blocked" by x. Similarly, in (3), $p(y \mid x)$ does not appear (this type of behavior is particularly useful if we only have access to a simulator of a system, but not access to the actual likelihood function). On the other hand, (4) has a direct path from θ to f, which contributes a term to the gradient estimator. (5) resembles a parameterized Markov reward process, and it illustrates that we'll obtain score function terms of the form *grad log-probability* × *future costs*.

The examples above all have one input θ, but the formalism accommodates models with multiple inputs, for example a stochastic neural network with multiple layers of weights and biases, which may influence different subsets of the stochastic and cost nodes. See Section 5.10 for nontrivial examples with stochastic nodes and multiple inputs. The figure on the right shows a de-



terministic computation graph representing classification loss for a two-layer neural network, which has four parameters (W_1, b_1, W_2, b_2) (weights and biases). Of course, this deterministic computation graph is a special type of stochastic computation graph.

5.3 main results on stochastic computation graphs

5.3.1 *Gradient Estimators*

This section will consider a general stochastic computation graph, in which a certain set of nodes are designated as *costs*, and we would like to compute the gradient of the sum of costs with respect to some input node θ.

Notation Glossary

Θ: Input nodes

$\mathcal{D}$: Deterministic nodes

$\mathcal{S}$: Stochastic nodes

$\mathcal{C}$: Cost nodes

$v \prec w$: v influences w

$v \prec^D w$: v deterministically influences w

DEPS_v: "dependencies", $\{w \in \Theta \cup \mathcal{S} \mid w \prec^D v\}$

$\hat{Q}_v$: sum of cost nodes influenced by v.

$\hat{v}$: denotes the sampled value of the node v.

In brief, the main results of this section are as follows:

1. We derive a gradient estimator for an expected sum of costs in a stochastic computation graph. This estimator contains two parts (1) a *score function* part, which is a sum of terms *grad log-prob of variable* $\times$ *sum of costs influenced by variable*; and (2) a *pathwise derivative* term, that propagates the dependence through differentiable functions.
2. This gradient estimator can be computed efficiently by differentiating an appropriate "surrogate" objective function.

Let Θ denote the set of input nodes, $\mathcal{D}$ the set of deterministic nodes, and $\mathcal{S}$ the set of stochastic nodes. Further, we will designate a set of cost nodes $\mathcal{C}$, which are scalar-valued and deterministic. (Note that there is no loss of generality in assuming that the costs are deterministic—if a cost is stochastic, we can simply append a deterministic node that applies the identity function to it.) We will use θ to denote an input node ($\theta \in \Theta$) that we differentiate with respect to. In the context of machine learning, we will usually be most concerned with differentiating with respect to a parameter vector (or tensor), however, the theory we present does not make any assumptions about what θ represents.

For the results that follow, we need to define the notion of "influence", for which we will introduce two relations $\prec$ and $\prec^D$. The relation $v \prec w$ ("v influences w") means that there exists a sequence of nodes $a_1, a_2, \dots, a_K$, with $K \geqslant 0$, such that
$(v, a_1), (a_1, a_2), \dots, (a_{K-1}, a_K), (a_K, w)$ are edges in the graph. The relation $v \prec^D w$ ("v *deterministically* influences w") is defined similarly, except that now we require that each a_k is a deterministic node. For example, in Figure 11, diagram (5) above, θ influences $\{x_1, x_2, f_1, f_2\}$, but it only deterministically influences $\{x_1, x_2\}$.

Next, we will establish a condition that is sufficient for the existence of the gradient. Namely, we will stipulate that every edge (v, w) with w lying in the "influenced" set of θ corresponds to a differentiable dependency: if w is deterministic, then the Jacobian $\frac{\partial w}{\partial v}$ must exist; if w is stochastic, then the probability mass function $p(w \mid v, \dots)$ must be differentiable with respect to v.

More formally:

Condition 1 (Differentiability Requirements)**.** *Given input node $\theta \in \Theta$, for all edges (v, w) which satisfy $\theta \prec^D v$ and $\theta \prec^D w$, then the following condition holds: if w is deterministic, Jacobian $\frac{\partial w}{\partial v}$ exists, and if w is stochastic, then the derivative of the probability mass function $\frac{\partial}{\partial v} p(w \mid \textsc{parents}_w)$ exists.*

Note that Condition 1 does not require that all the functions in the graph are differentiable. If the path from an input θ to deterministic node v is blocked by stochastic nodes, then v may be a nondifferentiable function of its parents. If a path from input θ to stochastic node v is blocked by other stochastic nodes, the likelihood of v given its parents need not be differentiable; in fact, it does not need to be known[2].

We need a few more definitions to state the main theorems. Let $\textsc{deps}_v := \{w \in \Theta \cup \mathcal{S} \mid w \prec^D v\}$, the "dependencies" of node v, i.e., the set of nodes that deterministically influence it. Note the following:

- If $v \in \mathcal{S}$, the probability mass function of v is a function of $\textsc{deps}_v$, i.e., we can write $p(v \mid \textsc{deps}_v)$.
- If $v \in \mathcal{D}$, v is a deterministic function of $\textsc{deps}_v$, so we can write $v(\textsc{deps}_v)$.

Let $\hat{Q}_v := \sum_{\substack{c \succ v, \\ c \in \mathcal{C}}} \hat{c}$, i.e., the sum of costs downstream of node v. These costs will be treated as constant, fixed to the values obtained during sampling. In general, we will use the hat symbol $\hat{v}$ to denote a sample value of variable v, which will be treated as constant in the gradient formulae.

[2] This fact is particularly important for reinforcement learning, allowing us to compute policy gradient estimates despite having a discontinuous dynamics function or reward function.

Now we can write down a general expression for the gradient of the expected sum of costs in a stochastic computation graph:

THEOREM 1. *Suppose that $\theta \in \Theta$ satisfies Condition 1. Then the following two equivalent equations hold:*

$$\frac{\partial}{\partial\theta}\mathbb{E}\left[\sum_{c\in\mathcal{C}} c\right] = \mathbb{E}\left[\sum_{\substack{w\in\mathcal{S},\\ \theta\prec^{D} w}}\left(\frac{\partial}{\partial\theta}\log p(w \mid \text{DEPS}_w)\right)\hat{Q}_w + \sum_{\substack{c\in\mathcal{C}\\ \theta\prec^{D} c}}\frac{\partial}{\partial\theta}c(\text{DEPS}_c)\right] \tag{38}$$

$$= \mathbb{E}\left[\sum_{c\in\mathcal{C}}\hat{c}\sum_{\substack{w\prec c,\\ \theta\prec^{D} w}}\frac{\partial}{\partial\theta}\log p(w \mid \text{DEPS}_w) + \sum_{\substack{c\in\mathcal{C},\\ \theta\prec^{D} c}}\frac{\partial}{\partial\theta}c(\text{DEPS}_c)\right]. \tag{39}$$

Proof: See Section 5.8.

The estimator expressions above have two terms. The first term is due to the influence of θ on probability distributions. The second term is due to the influence of θ on the cost variables through a chain of differentiable functions. The distribution term involves a sum of gradients times "downstream" costs. The first term in Equation (38) involves a sum of gradients times "downstream" costs, whereas the first term in Equation (39) has a sum of costs times "upstream" gradients.

5.3.2 *Surrogate Loss Functions*

The next corollary lets us write down a "surrogate" objective L, which is a function of the inputs that we can differentiate to obtain an unbiased gradient estimator.

Corollary 1. *Let* $L(\Theta, \mathcal{S}) := \sum_w \log p(w \mid \text{DEPS}_w)\hat{Q}_w + \sum_{c\in\mathcal{C}} c(\text{DEPS}_c)$. *Then differentiation of* L *gives us an unbiased gradient estimate:* $\frac{\partial}{\partial\theta}\mathbb{E}\left[\sum_{c\in\mathcal{C}} c\right] = \mathbb{E}\left[\frac{\partial}{\partial\theta}L(\Theta, \mathcal{S})\right]$.

One practical consequence of this result is that we can apply a standard automatic differentiation procedure to L to obtain an unbiased gradient estimator. In other words, we convert the stochastic computation graph into a deterministic computation graph, to which we can apply the backpropagation algorithm.

There are several alternative ways to define the surrogate objective function that give the same gradient as L from Corollary 1. We could also write $L(\Theta, \mathcal{S}) := \sum_w \frac{p(\hat{w} \mid \textsc{deps}_w)}{\hat{P}_v} \hat{Q}_w + \sum_{c \in \mathcal{C}} c(\textsc{deps}_c)$, where $\hat{P}_w$ is the probability $p(w \mid \textsc{deps}_w)$ obtained during sampling, which is viewed as a constant.

The surrogate objective from Corollary 1 is actually an upper bound on the true objective in the case that (1) all costs $c \in \mathcal{C}$ are negative, (2) the the costs are not deterministically influenced by the parameters Θ. This construction allows from majorization-minimization algorithms (similar to EM) to be applied to general stochastic computation graphs. See Section 5.9 for details.

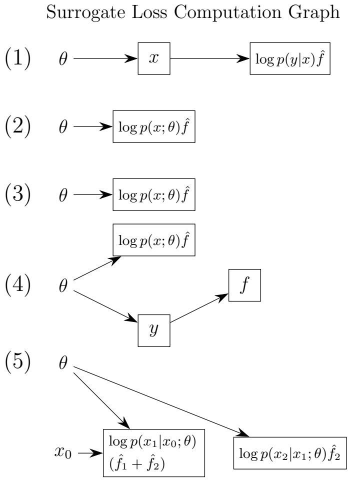

Surrogate Loss Computation Graph
(1)
θ
x
$\log p(y|x)\hat{f}$
(2)
θ
$\log p(x;\theta)\hat{f}$
(3)
θ
$\log p(x;\theta)\hat{f}$
(4)
θ
$\log p(x;\theta)\hat{f}$
f
y
(5)
θ
x_0
$\log p(x_1|x_0;\theta)$ $(\hat{f}_1 + \hat{f}_2)$
$\log p(x_2|x_1;\theta)\hat{f}_2$

Figure 12: Deterministic computation graphs obtained as surrogate loss functions of stochastic computation graphs from Figure 11.

5.3.3 *Higher-Order Derivatives.*

The gradient estimator for a stochastic computation graph is itself a stochastic computation graph. Hence, it is possible to compute the gradient yet again (for each component of the gradient vector), and get an estimator of the Hessian. For most problems of interest, it is not efficient to compute this dense Hessian. On the other hand, one can also differentiate the gradient-vector product to get a Hessian-vector product—this computation is usually not much more expensive than the gradient computation itself. The Hessian-vector product can be used to implement a quasi-Newton algorithm via the conjugate gradient algorithm [WN99]. A variant of this technique, called Hessian-free optimization [Mar10], has been used to train large neural networks.

5.4 VARIANCE REDUCTION

Consider estimating $\frac{\partial}{\partial \theta} \mathbb{E}_{x \sim p(\cdot;\, \theta)}[f(x)]$. Clearly this expectation is unaffected by subtracting a constant b from the integrand, which gives $\frac{\partial}{\partial \theta} \mathbb{E}_{x \sim p(\cdot;\, \theta)}[f(x) - b]$. Taking the score function estimator, we get $\frac{\partial}{\partial \theta} \mathbb{E}_{x \sim p(\cdot;\, \theta)}[f(x)] = \mathbb{E}_{x \sim p(\cdot;\, \theta)}\left[\frac{\partial}{\partial \theta} \log p(x;\, \theta)(f(x) - b)\right]$. Taking $b = \mathbb{E}_x[f(x)]$ generally leads to substantial variance reduction—b is often called a *base-*

line[3] (see [GBB04] for a more thorough discussion of baselines and their variance reduction properties).

We can make a general statement for the case of stochastic computation graphs—that we can add a baseline to every stochastic node, which depends all of the nodes it doesn't influence. Let $\textsc{NonInfluenced}(v) := \{w \mid v \not\prec w\}$.

THEOREM 2.

$$\frac{\partial}{\partial\theta}\mathbb{E}\left[\sum_{c\in\mathcal{C}} c\right] = \mathbb{E}\left[\sum_{\substack{v\in\mathcal{S}\\ v\succ\theta}}\left(\frac{\partial}{\partial\theta}\log p(v \mid \textsc{parents}_v)\right)(\hat{Q}_v - b(\textsc{NonInfluenced}(v)) + \sum_{c\in\mathcal{C}\succeq\theta}\frac{\partial}{\partial\theta}c\right]$$

Proof: See Section 5.8.

5.5 ALGORITHMS

As shown in Section 5.3, the gradient estimator can be obtained by differentiating a surrogate objective function L. Hence, this derivative can be computed by performing the backpropagation algorithm on L. That is likely to be the most practical and efficient method, and can be facilitated by automatic differentiation software.

Algorithm 4 shows explicitly how to compute the gradient estimator in a backwards pass through the stochastic computation graph. The algorithm will recursively compute $g_v := \frac{\partial}{\partial v}\mathbb{E}\left[\sum_{\substack{c\in\mathcal{C}\\ v\prec c}} c\right]$ at every deterministic and input node v.

5.6 RELATED WORK

As discussed in Section 5.2, the score function and pathwise derivative estimators have been used in a variety of different fields, under different names. See [Fu06] for a review of gradient estimation, mostly from the simulation optimization literature. Glasserman's textbook provides an extensive treatment of various gradient estimators and Monte Carlo estimators in general. Griewank and Walther's textbook [GW08] is a comprehensive reference on computation graphs and automatic differentiation (of deterministic programs.) The notation and nomenclature we use is inspired by Bayes nets and influence diagrams

[3] The optimal baseline for scalar θ is in fact the weighted expectation $\frac{\mathbb{E}_x[f(x)s(x)^2]}{\mathbb{E}_x[s(x)^2]}$ where $s(x) = \frac{\partial}{\partial\theta}\log p(x;\, \theta)$.

Algorithm 4 Compute Gradient Estimator for Stochastic Computation Graph

```
for $v \in$ Graph do                                          ▷ Initialization at output nodes
    $\mathbf{g}_v = \begin{cases} \mathbb{1}_{\dim v} & \text{if } v \in \mathcal{C} \\ 0_{\dim v} & \text{otherwise} \end{cases}$
end for
Compute $\hat{Q}_w$ for all nodes $w \in$ Graph
for $v$ in ReverseTopologicalSort(NonInputs) do               ▷ Reverse traversal
    for $w \in$ parents$_v$ do
        if not IsStochastic($w$) then
            if IsStochastic($v$) then
                $\mathbf{g}_w \mathrel{+}= (\frac{\partial}{\partial w} \log p(v \mid \textsc{parents}_v))\hat{Q}_w$
            else
                $\mathbf{g}_w \mathrel{+}= (\frac{\partial v}{\partial w})^{\mathsf{T}} \mathbf{g}_v$
            end if
        end if
    end for
end for
return $[\mathbf{g}_\theta]_{\theta \in \Theta}$
```

[Pea14]. (In fact, a stochastic computation graph is a type of Bayes network; where the deterministic nodes correspond to degenerate probability distributions.)

The topic of gradient estimation has drawn significant recent interest in machine learning. Gradients for networks with stochastic units was investigated in Bengio et al. [BLC13], though they are concerned with differentiating through individual units and layers; not how to deal with arbitrarily structured models and loss functions. Kingma and Welling [KW14] consider a similar framework, although only with continuous latent variables, and point out that reparameterization can be used to to convert hierarchical Bayesian models into neural networks, which can then be trained by backpropagation.

The score function method is used to perform variational inference in general models (in the context of probabilistic programming) in Wingate and Weber [WW13], and similarly in Ranganath et al. [RGB13]; both papers mostly focus on mean-field approximations without amortized inference. It is used to train generative models using neural networks with discrete stochastic units in Mnih and Gregor [MG14] and Gregor et al. in [Gre+13]; both amortize inference by using an inference network.

Generative models with continuous valued latent variables networks are trained (again using an inference network) with the reparametrization method by Rezende, Mohamed, and Wierstra [RMW14] and by Kingma and Welling [KW13]. Rezende et al. also provide a detailed discussion of reparameterization, including a discussion comparing the variance of the SF and PD estimators.

Bengio, Leonard, and Courville [BLC13] have recently written a paper about gradient estimation in neural networks with stochastic units or non-differentiable activation functions—including Monte Carlo estimators and heuristic approximations. The notion that policy gradients can be computed in multiple ways was pointed out in early work on policy gradients by Williams [Wil92]. However, all of this prior work deals with specific structures of the stochastic computation graph and does not address the general case.

5.7 conclusion

The reinforcement learning is extremely general and lies at the heart of artificial intelligence, and corresponds to the ability for decision making and motor control. The core idea in deep learning is that by reducing learning into optimization, it is possible to learn function approximators that perform computation.

We have developed a framework for describing a computation with stochastic and deterministic operations, called a stochastic computation graph. Given a stochastic computation graph, we can automatically obtain a gradient estimator, given that the graph satisfies the appropriate conditions on differentiability of the functions at its nodes. The gradient can be computed efficiently in a backwards traversal through the graph: one approach is to apply the standard backpropagation algorithm to one of the surrogate loss functions from Section 5.3; another approach (which is roughly equivalent) is to apply a modified backpropagation procedure shown in Algorithm 4. The results we have presented are sufficiently general to automatically reproduce a variety of gradient estimators that have been derived in prior work in reinforcement learning and probabilistic modeling, as we show in Section 5.10. We hope that this work will facilitate further development of interesting and expressive models.

5.8 proofs

Theorem 1

We will consider the case that all of the random variables are continuous-valued, thus the expectations can be written as integrals. For discrete random variables, the integrals should be changed to sums.

Recall that we seek to compute $\frac{\partial}{\partial\theta}\mathbb{E}\left[\sum_{c\in\mathcal{C}} c\right]$. We will differentiate the expectation of a single cost term; summing over these terms yields Equation (39).

$$\begin{align*}
\mathbb{E}_{\substack{v\in\mathcal{S},\\ v\prec c}}[c] &= \int \prod_{\substack{v\in\mathcal{S},\\ v\prec c}} p(v\,|\,\textsc{deps}_v)dv \;\; c(\textsc{deps}_c) \\
\frac{\partial}{\partial\theta}\mathbb{E}_{\substack{v\in\mathcal{S},\\ v\prec c}}[c] &= \frac{\partial}{\partial\theta}\int \prod_{\substack{v\in\mathcal{S},\\ v\prec c}} p(v\,|\,\textsc{deps}_v)dv \;\; c(\textsc{deps}_c) \\
&= \int \prod_{\substack{v\in\mathcal{S},\\ v\prec c}} p(v\,|\,\textsc{deps}_v)dv \left[\sum_{\substack{w\in\mathcal{S},\\ w\prec c}} \frac{\frac{\partial}{\partial\theta}p(w\,|\,\textsc{deps}_w)}{p(w\,|\,\textsc{deps}_w)} c(\textsc{deps}_c) + \frac{\partial}{\partial\theta}c(\textsc{deps}_c)\right] \tag{40} \\
&= \int \prod_{\substack{v\in\mathcal{S},\\ v\prec c}} p(v\,|\,\textsc{deps}_v)dv \left[\sum_{\substack{w\in\mathcal{S},\\ w\prec c}} \left(\frac{\partial}{\partial\theta}\log p(w\,|\,\textsc{deps}_w)\right) c(\textsc{deps}_c) + \frac{\partial}{\partial\theta}c(\textsc{deps}_c)\right] \\
&= \mathbb{E}_{\substack{v\in\mathcal{S},\\ v\prec c}}\left[\sum_{\substack{w\in\mathcal{S},\\ w\prec c}} \frac{\partial}{\partial\theta}\log p(w\,|\,\textsc{deps}_w)\hat{c} + \frac{\partial}{\partial\theta}c(\textsc{deps}_c)\right].
\end{align*}$$

Equation (40) requires that the integrand is differentiable, which is satisfied if all of the PDFs and $c(\textsc{deps}_c)$ are differentiable. Equation (39) follows by summing over all costs $c \in \mathcal{C}$. Equation (38) follows from rearrangement of the terms in this equation.

Theorem 2

It suffices to show that for a particular node $v \in \mathcal{S}$, the following expectation (taken over all variables) vanishes

$$\mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\log p(v \mid \textsc{parents}_v)\right) b(\textsc{NonInfluenced}(v))\right].$$

Analogously to $\textsc{NonInfluenced}(v)$, define $\textsc{Influenced}(v) := \{w \mid w \succ v\}$. Note that the nodes can be ordered so that $\textsc{NonInfluenced}(v)$ all come before v in the ordering. Thus, we can write

$$\begin{aligned}
&\mathbb{E}_{\textsc{NonInfluenced}(v)}\left[\mathbb{E}_{\textsc{Influenced}(v)}\left[\left(\frac{\partial}{\partial\theta}\log p(v \mid \textsc{parents}_v)\right) b(\textsc{NonInfluenced}(v))\right]\right] \\
&= \mathbb{E}_{\textsc{NonInfluenced}(v)}\left[\mathbb{E}_{\textsc{Influenced}(v)}\left[\left(\frac{\partial}{\partial\theta}\log p(v \mid \textsc{parents}_v)\right)\right] b(\textsc{NonInfluenced}(v))\right] \\
&= \mathbb{E}_{\textsc{NonInfluenced}(v)}\left[0 \cdot b(\textsc{NonInfluenced}(v))\right] \\
&= 0
\end{aligned}$$

where we used $\mathbb{E}_{\textsc{Influenced}(v)}\left[\left(\frac{\partial}{\partial\theta}\log p(v \mid \textsc{parents}_v)\right)\right] = \mathbb{E}_v\left[\left(\frac{\partial}{\partial\theta}\log p(v \mid \textsc{parents}_v)\right)\right] = 0$.

5.9 SURROGATE AS AN UPPER BOUND, AND MM ALGORITHMS

L has additional significance besides allowing us to estimate the gradient of the expected sum of costs. Under certain conditions, L is a upper bound on on the true objective (plus a constant).

We shall make two restrictions on the stochastic computation graph: (1) first, that all costs $c \in \mathcal{C}$ are negative. (2) the the costs are not deterministically influenced by the parameters Θ. First, let us use importance sampling to write down the expectation of a given cost node, when the sampling distribution is different from the distribution we are evaluating: for parameter $\theta \in \Theta$, $\theta = \theta_{\text{old}}$ is used for sampling, but we are evaluating at

$\theta = \theta_{new}$.

$$\begin{aligned}
\mathbb{E}_{v \prec c \,|\, \theta_{new}}[\hat{c}] &= \mathbb{E}_{v \prec c \,|\, \theta_{old}}\left[\hat{c} \prod_{\substack{v \prec c,\\ \theta \prec^D v}} \frac{P_v(v \,|\, \text{DEPS}_v \backslash \theta, \theta_{new})}{P_v(v \,|\, \text{DEPS}_v \backslash \theta, \theta_{old})}\right] \\
&\leqslant \mathbb{E}_{v \prec c \,|\, \theta_{old}}\left[\hat{c}\left(\log\left(\prod_{\substack{v \prec c,\\ \theta \prec^D v}} \frac{P_v(v \,|\, \text{DEPS}_v \backslash \theta, \theta_{new})}{P_v(v \,|\, \text{DEPS}_v \backslash \theta, \theta_{old})}\right) + 1\right)\right]
\end{aligned}$$

where the second line used the inequality $x \geqslant \log x + 1$, and the sign is reversed since $\hat{c}$ is negative. Summing over $c \in \mathcal{C}$ and rearranging we get

$$\begin{aligned}
\mathbb{E}_{\mathcal{S} \,|\, \theta_{new}}\left[\sum_{c \in \mathcal{C}} \hat{c}\right] &\leqslant \mathbb{E}_{\mathcal{S} \,|\, \theta_{old}}\left[\sum_{c \in \mathcal{C}} \hat{c} + \sum_{v \in \mathcal{S}} \log\left(\frac{p(v \,|\, \text{DEPS}_v \backslash \theta, \theta_{new})}{p(v \,|\, \text{DEPS}_v \backslash \theta, \theta_{old})}\right)\hat{Q}_v\right] \\
&= \mathbb{E}_{\mathcal{S} \,|\, \theta_{old}}\left[\sum_{v \in \mathcal{S}} \log p(v \,|\, \text{DEPS}_v \backslash \theta, \theta_{new})\hat{Q}_v\right] + \text{const.} \qquad (41)
\end{aligned}$$

Equation (41) allows for majorization-minimization algorithms (like the EM algorithm) to be used to optimize with respect to θ. In fact, similar equations have been derived by interpreting rewards (negative costs) as probabilities, and then taking the variational lower bound on log-probability (e.g., [Vla+09]).

5.10 EXAMPLES

This section considers two settings where the formalism of stochastic computation graphs can be applied. First, we consider the generalized EM algorithm for maximum likelihood estimation in probabilistic models with latent variables. Second, we consider reinforcement learning in Markov Decision Processes. In both cases, the objective function is given by an expectation; writing it out as a composition of stochastic and deterministic steps yields a stochastic computation graph.

5.10.1 *Generalized EM Algorithm and Variational Inference.*

The generalized EM algorithm maximizes likelihood in a probabilistic model with latent variables [NH98]. We start with a parameterized probability density $p(x, z; \theta)$ where x is observed, z is a latent variable, and θ is a parameter of the distribution. The generalized EM algorithm maximizes the *variational lower bound*, which is defined by an expectation over z for each sample x:

$$L(\theta, q) = \mathbb{E}_{z \sim q}\left[\log\left(\frac{p(x, z; \theta)}{q(z)}\right)\right].$$

As parameters will appear both in the probability density and inside the expectation, stochastic computation graphs provide a convenient route for deriving the gradient estimators.

Neural variational inference.

Mnih and Gregor [MG14] propose a generalized EM algorithm for multi-layered latent variable models that employs an *inference network*, an explicit parameterization of the posterior $q_\phi(z \mid x) \approx p(z \mid x)$, to allow for fast approximate inference. The generative model and inference network take the form

$$\begin{aligned} p_\theta(x) &= \sum_{h_1, h_2} p_{\theta_1}(x|h_1) p_{\theta_2}(h_1|h_2) p_{\theta_3}(h_2|h_3) p_{\theta_3}(h_3) \\ q_\phi(h_1, h_2|x) &= q_{\phi_1}(h_1|x) q_{\phi_2}(h_2|h_1) q_{\phi_3}(h_3|h_2). \end{aligned}$$

The inference model q_ϕ is used for sampling, i.e., we sample $h_1 \sim q_{\phi_1}(\cdot \mid x), h_2 \sim q_{\phi_2}(\cdot \mid h_1), h_3 \sim q_{\phi_3}(\cdot \mid h_2)$. The stochastic computation graph is shown above.

$$L(\theta, \phi) = \mathbb{E}_{h \sim q_\phi}\left[\underbrace{\log \frac{p_{\theta_1}(x|h_1)}{q_{\phi_1}(h_1|x)}}_{=r_1} + \underbrace{\log \frac{p_{\theta_2}(h_1|h_2)}{q_{\phi_2}(h_2|h_1)}}_{=r_2} + \underbrace{\log \frac{p_{\theta_3}(h_2|h_3) p_{\theta_3}(h_3)}{q_{\phi_3}(h_3|h_2)}}_{=r_3}\right].$$

Given a sample $h \sim q_\phi$ an unbiased estimate of the gradient is given by Theorem 2 as

$$\frac{\partial L}{\partial \theta} \approx \frac{\partial}{\partial \theta} \log p_{\theta_1}(x|h_1) + \frac{\partial}{\partial \theta} \log p_{\theta_2}(h_1|h_2) + \frac{\partial}{\partial \theta} \log p_{\theta_3}(h_2) \tag{42}$$

$$\begin{aligned} \frac{\partial L}{\partial \phi} \approx{}& \frac{\partial}{\partial \phi} \log q_{\phi_1}(h_1|x)(\hat{Q}_1 - b_1(x)) \\ &+ \frac{\partial}{\partial \phi} \log q_{\phi_2}(h_2|h_1)(\hat{Q}_2 - b_2(h_1)) + \frac{\partial}{\partial \phi} \log q_{\phi_3}(h_3|h_2)(\hat{Q}_3 - b_3(h_2)) \end{aligned} \tag{43}$$

where $\hat{Q}_1 = r_1 + r_2 + r_3$; $\hat{Q}_2 = r_2 + r_3$; and $\hat{Q}_3 = r_3$, and b_1, b_2, b_3 are baseline functions. The stochastic computation graph is shown in Figure 13.

Variational Autoencoder, Deep Latent Gaussian Models and Reparameterization. Here we'll note out that in some cases, the stochastic computation graph can be transformed to give the same probability distribution for the observed variables, but one obtains a different gradient estimator. Kingma and Welling [KW13] and Rezende et al. [RMW14] consider a model that is similar to the one proposed by Mnih et al. [MG14] but with continuous latent variables, and they re-parameterize their inference network to enable the use of the PD estimator. The original objective, the variational lower bound, is

$$L_{\text{orig}}(\theta, \phi) = \mathbb{E}_{h \sim q_\phi}\left[\log \frac{p_\theta(x|h)p_\theta(h)}{q_\phi(h|x)}\right].$$

The second term, the entropy of q_ϕ, can be computed analytically for the parametric forms of q considered in the paper (Gaussians). For q_ϕ being conditionally Gaussian, i.e. $q_\phi(h|x) = N(h|\mu_\phi(x), \sigma_\phi(x))$ re-parameterizing leads to $h = h_\phi(\epsilon; x) = \mu_\phi(x) + \epsilon\sigma_\phi(x)$, giving

$$\begin{aligned} L_{\text{re}}(\theta, \phi) = \mathbb{E}_{\epsilon \sim \rho}&\left[\log p_\theta(x|h_\phi(\epsilon, x)) + \log p_\theta(h_\phi(\epsilon, x))\right] \\ &+ H[q_\phi(\cdot|x)]. \end{aligned}$$

The stochastic computation graph before and after reparameterization is shown in Figure 13. Given $\epsilon \sim \rho$ an estimate of the gradient is obtained as

$$\begin{aligned} \frac{\partial L_{\text{re}}}{\partial \theta} &\approx \frac{\partial}{\partial \theta}\left[\log p_\theta(x|h_\phi(\epsilon, x)) + \log p_\theta(h_\phi(\epsilon, x))\right], \\ \frac{\partial L_{\text{re}}}{\partial \phi} &\approx \left[\frac{\partial}{\partial h}\log p_\theta(x|h_\phi(\epsilon, x)) + \frac{\partial}{\partial h}\log p_\theta(h_\phi(\epsilon, x))\right]\frac{\partial h}{\partial \phi} + \frac{\partial}{\partial \phi}H[q_\phi(\cdot|x)]. \end{aligned}$$

5.10.2 *Policy Gradients in Reinforcement Learning.*

In reinforcement learning, an agent interacts with an environment according to its policy π, and the goal is to maximize the expected sum of rewards, called the *return*. Policy gradient methods seek to directly estimate the gradient of expected return with respect to the policy parameters [Wil92; BB01; Sut+99]. In reinforcement learning, we typically assume that the environment dynamics are not available analytically and can only be sampled. Below we distinguish two important cases: the *Markov decision process* (MDP) and the *partially observable Markov decision process* (POMDP).

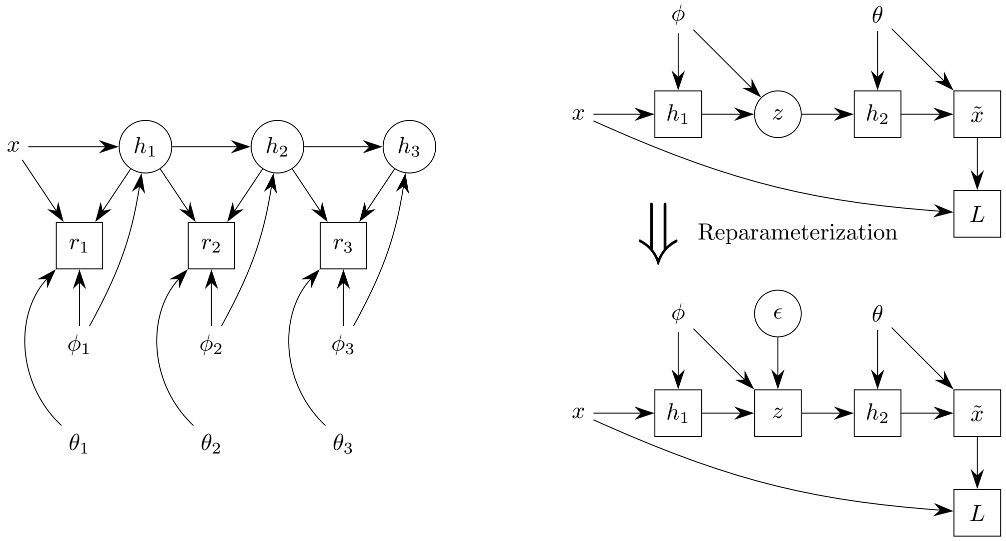


Figure 13: Stochastic computation graphs for NVIL (left) and VAE (right) models

MDPs. In the MDP case, the expectation is taken with respect to the distribution over state (s) and action (a) sequences

$$L(\theta) = \mathbb{E}_{\tau \sim p_\theta}\left[\sum_{t=1}^{T} r(s_t, a_t)\right],$$

where $\tau = (s_1, a_1, s_2, a_2, \dots)$ are trajectories and the distribution over trajectories is defined in terms of the environment dynamics $p_E(s_{t+1} \mid s_t, a_t)$ and the policy π_θ: $p_\theta(\tau) = p_E(s_1)\prod_t \pi_\theta(a_t \mid s_t) p_E(s_{t+1} \mid s_t, a_t)$. r are rewards (negative costs in the terminology of the rest of the paper). The classic *REINFORCE* [Wil92] estimate of the gradient is given by

$$\frac{\partial}{\partial\theta} L = \mathbb{E}_{\tau \sim p_\theta}\left[\sum_{t=1}^{T} \frac{\partial}{\partial\theta} \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) - b_t(s_t)\right)\right], \tag{44}$$

where $b_t(s_t)$ is an arbitrary baseline which is often chosen to approximate $V_t(s_t) = \mathbb{E}_{\tau \sim p_\theta}\left[\sum_{t'=t}^{T} r(s_{t'}, a_{t'})\right]$, i.e. the state-value function. Note that the stochastic action nodes a_t "block" the differentiable path from θ to rewards, which eliminates the need to differentiate through the unknown environment dynamics. The stochastic computation graph is shown in Figure 14.

POMDPs. POMDPs differ from MDPs in that the state s_t of the environment is not observed directly but, as in latent-variable time series models, only through stochastic observations o_t, which depend on the latent states s_t via $p_E(o_t \mid s_t)$. The policy therefore has to be a function of the history of past observations $\pi_\theta(a_t \mid o_1 \dots o_t)$. Applying Theorem 2, we obtain a gradient estimator:

$$\frac{\partial}{\partial \theta} L = \mathbb{E}_{\tau \sim p_\theta}\left[\sum_{t=1}^{T} \frac{\partial}{\partial \theta} \log \pi_\theta(a_t \mid o_1 \dots o_t)) \left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) - b_t(o_1 \dots o_t) \right) \right]. \tag{45}$$

Here, the baseline b_t and the policy π_θ can depend on the observation history through time t, and these functions can be parameterized as recurrent neural networks [Wie+10; Mni+14]. The stochastic computation graph is shown in Figure 14.

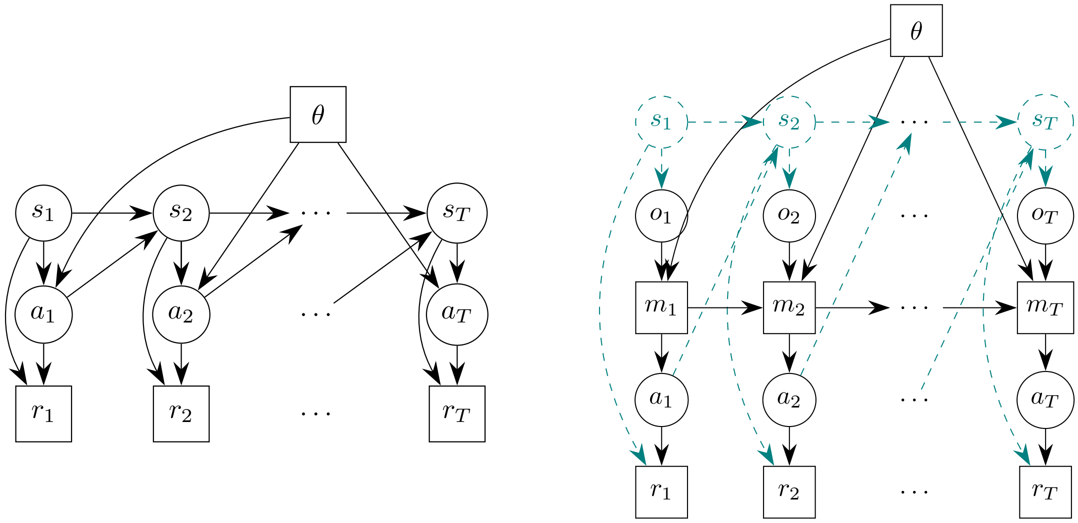


Figure 14: Stochastic Computation Graphs for MDPs (left) and POMDPs (right)

6

CONCLUSION

The reinforcement learning problem, of maximizing reward in a POMDP, is extremely general and lies at the core of artificial intelligence. Historically, most work in reinforcement learning has used function approximators with limited expressivity, but recent work in deep reinforcement learning (including this thesis) studies how to use expressive function approximators such as deep neural networks. These function approximators are capable of performing multi-step computations, but they are also tractable to learn gradient-based optimization. Nevertheless, deep reinforcement learning brings many challenges, in how to develop reinforcement learning algorithms that are reliable, scalable, and reasonably sample efficient.

This thesis is mostly concerned with developing deep reinforcement learning algorithms that are more reliable and sample-efficient than the algorithms that were available previously. In this work, we focus on using stochastic policies, for which it is possible to obtain estimators of the gradient of performance. We developed an algorithm called trust region policy optimization (TRPO), which is theoretically justified, and empirically performs well in the challenging domains of Atari and 2D simulated robotic locomotion. Recently, Duan et al. [Dua+16] found TRPO to perform the best overall out of the algorithms considered on a benchmark of continuous control problems. We also studied variance reduction for policy gradient methods, unifying and expending on several some previous statements of this idea, and obtaining strong empirical results in the domain of 3D simulated robotic locomotion, which exceed previous results obtained with reinforcement learning.

The last work discussed, on *stochastic computation graphs*, makes the point that policy gradient methods for reinforcement learning are an instance of a more general class of techniques for optimizing objectives defined as expectations. We expect this to be useful for deriving optimization procedures in reinforcement learning or other probabilistic modeling problems; also, the unifying view motivates using RL algorithms like TRPO in non-RL problems.

6.1 frontiers

Many open problems remain, which relate to and could build on this thsis work. Below, we describe some of the frontiers that we consider to be the most exciting, mostly in the field of deep reinforcement learning.

1. *Shared representations for control and prediction*. In domains with high-dimensional observations (for example, robotics using camera input, or games like Atari), two different mappings need to be learned: first, we need to map the raw input into more useful representations (for example, parse the image into a set of objects and their locations); second, we need to map these representations to the actions. When using policy gradient methods, this learning is driven by the advantage function, which is a noisy one-dimensional signal—i.e., it is a slow source of information about the environment. It should be possible to learn representations faster by solving prediction problems involving the observations themselves—that way, we are using much more information from the environment. To speed up learning this way, we would need to use an architecture that shares parameters between a prediction part and an action-selection part.
2. *Hierarchy*: animals and (prospectively) intelligent robots need to carry out behaviors that unfold over a range of different timescales: fractions of a second for low-level motor control; hours or days for various high-level behaviors. But traditional reinforcement learning methods have fundamental difficulties learning any behaviors that require more than $100 - 1000$ timesteps. Learning can proceed if the MDP is augmented with high-level actions that unfold over a long period of time: some versions of this idea include *hierarchical abstract machines* [PR98] and *options* [SPS99]. The persistent difficulty is how to automatically learn these high-level actions, or what kind of optimization objective will encourage the policy to be more "hierarchical".
3. *Exploration*: the principle of exploration is to actively encourage the agent to reach unfamiliar parts of state space, avoiding convergence to a suboptimal policy. Policy gradient methods are prone to converging to suboptimal policies, as we observed many times while doing the empirical work in this thesis. While a body of theoretical work answers the question of how to explore optimally in an finite MDP (e.g., [Str+06]), there is a need for exploration methods that can be applied in challenging real-world settings such as robotics. Some preliminary work towards making

exploration work in the deep RL setting includes methods based on Thompson sampling [Osb+16] and exploration bonuses [Hou+16].

4. *Using learned models*: model-based reinforcement learning methods seek to speed up learning by fitting a dynamics model and using it for planning or speeding up learning. It is known that in certain low-dimensional continuous control problems, it is possible to learn good controllers in an extremely small number of samples (e.g., [DR11; Mol+15]); however, this success has not yet been extended to problems with high-dimensional state spaces. More generally, many have found that model based methods learn faster (in fewer samples) than model-free methods such as policy gradients and Q-learning when they work; however, no method has yet emerged that can perform as well as model-free methods on challenging high-dimensional tasks, such as the Atari and MuJoCo tasks considered in this thesis. Guided policy search, which uses a model for trajectory optimization [Lev+16], was used to learn some behaviors efficiently on a physical robot. These methods also have yet to be extended to problems that require controlling a high-dimensional state.
5. *Finer-grained credit assignment*: the policy gradient estimator performs credit assignment in a crude way, since it credits an action with all rewards that follow the action. However, often it is possible to do better credit assignment based on some knowledge of the system. For example, when one serves a tennis ball, the result does not depend on any action he takes after his racket hits the ball; however, that sort of inference is not included in any of our reinforcement learning algorithms. It should be possible to do better credit assignment with the help of a model of the system. Heess et al. [Hee+15] tried model-based credit assignment and obtained a negative result; however, other possible instantiations of the idea might be more successful. Another technique for variance reduction was proposed in [LCR02]; however, this technique only provides a moderate amount of variance reduction. If there were a generic method for approximating the unknown or non-differentiable components in a stochastic computation graph (e.g., the dynamics model in reinforcement learning) and using them to obtain better gradient estimates, this method could provide significant benefits in reinforcement learning and probabilistic modeling problems that involve “hard” decisions.

BIBLIOGRAPHY

That which has been is that which will be,
And that which has been done is that which will be done.
So there is nothing new under the sun.
— Ecclesiastes 1:9, NASB

[BS03] J. A. Bagnell and J. Schneider. "Covariant policy search." In: IJCAI. 2003 (cit. on pp. 6, 24).

[BB11] P. L. Bartlett and J. Baxter. "Infinite-horizon policy-gradient estimation." In: *arXiv preprint arXiv:1106.0665* (2011) (cit. on p. 25).

[BSA83] A. G. Barto, R. S. Sutton, and C. W. Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems." In: *Systems, Man and Cybernetics, IEEE Transactions on* 5 (1983), pp. 834–846 (cit. on pp. 31, 57).

[BB01] J. Baxter and P. L. Bartlett. "Infinite-horizon policy-gradient estimation." In: *Journal of Artificial Intelligence Research* (2001), pp. 319–350 (cit. on p. 81).

[Bel+13] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. "The Arcade Learning Environment: An Evaluation Platform for General Agents." In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279 (cit. on p. 32).

[BLC13] Y. Bengio, N. Léonard, and A. Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation." In: *arXiv preprint arXiv:1308.3432* (2013) (cit. on p. 76).

[Ber+10] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. "Theano: a CPU and GPU math expression compiler." In: *Proceedings of the Python for scientific computing conference (SciPy)*. Vol. 4. Austin, TX. 2010, p. 3 (cit. on p. 41).

[Ber05] D. Bertsekas. *Dynamic programming and optimal control*. Vol. 1. 2005 (cit. on p. 26).

[Ber12] D. P. Bertsekas. *Dynamic programming and optimal control*. Vol. 2. 2. Athena Scientific, 2012 (cit. on p. 53).

[Bha+09] S. Bhatnagar, D. Precup, D. Silver, R. S. Sutton, H. R. Maei, and C. Szepesvári. "Convergent temporal-difference learning with arbitrary smooth function approximation." In: *Advances in Neural Information Processing Systems*. 2009, pp. 1204–1212 (cit. on p. 60).

[Dah+12] G. E. Dahl, D. Yu, L. Deng, and A. Acero. "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition." In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.1 (2012), pp. 30–42 (cit. on pp. 1, 2).

[DILM09] H. Daumé Iii, J. Langford, and D. Marcu. "Search-based structured prediction." In: *Machine learning* 75.3 (2009), pp. 297–325 (cit. on p. 1).

[DNP13] M. Deisenroth, G. Neumann, and J. Peters. "A Survey on Policy Search for Robotics." In: *Foundations and Trends in Robotics* 2.1-2 (2013), pp. 1–142 (cit. on p. 18).

[DR11] M. Deisenroth and C. E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search." In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*. 2011, pp. 465–472 (cit. on p. 86).

[Dua+16] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. "Benchmarking Deep Reinforcement Learning for Continuous Control." In: *arXiv preprint arXiv:1604.06778* (2016) (cit. on p. 84).

[Fu06] M. C. Fu. "Gradient estimation." In: *Handbooks in operations research and management science* 13 (2006), pp. 575–616 (cit. on pp. 66, 67, 74).

[GGS13] V. Gabillon, M. Ghavamzadeh, and B. Scherrer. "Approximate Dynamic Programming Finally Performs Well in the Game of Tetris." In: *Advances in Neural Information Processing Systems*. 2013 (cit. on p. 25).

[GPW06] T. Geng, B. Porr, and F. Wörgötter. "Fast biped walking with a reflexive controller and realtime policy searching." In: *Advances in Neural Information Processing Systems (NIPS)*. 2006 (cit. on p. 32).

[Gla03] P. Glasserman. *Monte Carlo methods in financial engineering*. Vol. 53. Springer Science & Business Media, 2003 (cit. on pp. 66, 67).

[Gly90] P. W. Glynn. "Likelihood ratio gradient estimation for stochastic systems." In: *Communications of the ACM* 33.10 (1990), pp. 75–84 (cit. on pp. 64, 67).

[GBB04] E. Greensmith, P. L. Bartlett, and J. Baxter. "Variance reduction techniques for gradient estimates in reinforcement learning." In: *The Journal of Machine Learning Research* 5 (2004), pp. 1471–1530 (cit. on pp. 14, 47, 74).

[Gre+13] K. Gregor, I. Danihelka, A. Mnih, C. Blundell, and D. Wierstra. "Deep autoregressive networks." In: *arXiv preprint arXiv:1310.8499* (2013) (cit. on p. 76).

[GW08] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Siam, 2008 (cit. on p. 74).

[Gri+89] A. Griewank et al. "On automatic differentiation." In: *Mathematical Programming: recent developments and applications* 6.6 (1989), pp. 83–107 (cit. on p. 41).

[Guo+14] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning." In: *Advances in Neural Information Processing Systems*. 2014, pp. 3338–3346 (cit. on pp. 32, 33).

[HO96] N. Hansen and A. Ostermeier. "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation." In: *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. IEEE. 1996, pp. 312–317 (cit. on p. 31).

[Hau+12] M. Hausknecht, P. Khandelwal, R. Miikkulainen, and P. Stone. "HyperNEAT-GGP: A HyperNEAT-based Atari general game player." In: *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM. 2012, pp. 217–224 (cit. on pp. 3, 4).

[Hee+15] N. Heess, G. Wayne, D. Silver, T. Lillicrap, Y. Tassa, and T. Erez. "Learning Continuous Control Policies by Stochastic Value Gradients." In: *arXiv preprint arXiv:1510.09142* (2015) (cit. on pp. 3, 5, 61, 62, 86).

[HS97] S. Hochreiter and J. Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 64).

[Hou+16] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. "Curiosity-driven Exploration in Deep Reinforcement Learning via Bayesian Neural Networks." In: *arXiv preprint arXiv:1605.09674* (2016) (cit. on p. 86).

[HL04] D. R. Hunter and K. Lange. "A tutorial on MM algorithms." In: *The American Statistician* 58.1 (2004), pp. 30–37 (cit. on p. 22).

[JJS94] T. Jaakkola, M. I. Jordan, and S. P. Singh. "On the convergence of stochastic iterative dynamic programming algorithms." In: *Neural computation* 6.6 (1994), pp. 1185–1201 (cit. on pp. 4, 6, 16).

[Kak01a] S. Kakade. "A Natural Policy Gradient." In: *NIPS*. Vol. 14. 2001, pp. 1531–1538 (cit. on pp. 55, 61).

[Kak01b] S. Kakade. "Optimizing average reward using discounted rewards." In: *Computational Learning Theory*. Springer. 2001, pp. 605–615 (cit. on p. 47).

[Kak02] S. Kakade. "A Natural Policy Gradient." In: *Advances in Neural Information Processing Systems*. MIT Press, 2002, pp. 1057–1063 (cit. on pp. 4, 6, 28, 31).

[KL02] S. Kakade and J. Langford. “Approximately optimal approximate reinforcement learning.” In: *ICML*. Vol. 2. 2002, pp. 267–274 (cit. on pp. 19–21, 28, 34).

[KK98] H. Kimura and S. Kobayashi. “An Analysis of Actor/Critic Algorithms Using Eligibility Traces: Reinforcement Learning with Imperfect Value Function.” In: *ICML*. 1998, pp. 278–286 (cit. on pp. 45, 46).

[KB14] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization.” In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 16).

[KW13] D. P. Kingma and M. Welling. “Auto-encoding variational Bayes.” In: *arXiv:1312.6114* (2013) (cit. on pp. 64, 66, 76, 81).

[KW14] D. P. Kingma and M. Welling. “Efficient gradient-based inference through transformations between bayes nets and neural nets.” In: *arXiv preprint arXiv:1402.0480* (2014) (cit. on p. 76).

[KBP13] J. Kober, J. A. Bagnell, and J. Peters. “Reinforcement learning in robotics: A survey.” In: *The International Journal of Robotics Research* (2013), p. 0278364913495721 (cit. on p. 1).

[KT03] V. R. Konda and J. N. Tsitsiklis. “On Actor-Critic Algorithms.” In: *SIAM journal on Control and Optimization* 42.4 (2003), pp. 1143–1166 (cit. on pp. 61, 62).

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “Imagenet classification with deep convolutional neural networks.” In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on pp. 1, 2).

[LP03] M. G. Lagoudakis and R. Parr. “Reinforcement learning as classification: Leveraging modern classifiers.” In: *ICML*. Vol. 3. 2003, pp. 424–431 (cit. on p. 25).

[LCR02] G. Lawrence, N. Cowan, and S. Russell. “Efficient gradient estimation for motor control learning.” In: *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc. 2002, pp. 354–361 (cit. on p. 86).

[LeC+98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition.” In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 64).

[LPW09] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2009 (cit. on p. 37).

[LA14] S. Levine and P. Abbeel. “Learning neural network policies with guided policy search under unknown dynamics.” In: *Advances in Neural Information Processing Systems*. 2014, pp. 1071–1079 (cit. on p. 29).

[Lev+16] S. Levine, C. Finn, T. Darrell, and P. Abbeel. “End-to-end training of deep visuomotor policies.” In: *Journal of Machine Learning Research* 17.39 (2016), pp. 1–40 (cit. on pp. 3, 86).

[Lil+15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. “Continuous control with deep reinforcement learning.” In: *arXiv preprint arXiv:1509.02971* (2015) (cit. on pp. 3, 5, 61, 62).

[Lin93] L.-J. Lin. *Reinforcement learning for robots using neural networks*. Tech. rep. DTIC Document, 1993 (cit. on p. 2).

[MT03] P. Marbach and J. N. Tsitsiklis. “Approximate gradient methods in policy-space optimization of Markov reward processes.” In: *Discrete Event Dynamic Systems* 13.1-2 (2003), pp. 111–148 (cit. on p. 47).

[MS12] J. Martens and I. Sutskever. “Training deep and recurrent networks with Hessian-free optimization.” In: *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 479–535 (cit. on p. 40).

[Mar10] J. Martens. “Deep learning via Hessian-free optimization.” In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 735–742 (cit. on pp. 65, 73).

[MG14] A. Mnih and K. Gregor. “Neural variational inference and learning in belief networks.” In: *arXiv:1402.0030* (2014) (cit. on pp. 64, 76, 80, 81).

[Mni+13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. “Playing Atari with Deep Reinforcement Learning.” In: *arXiv preprint arXiv:1312.5602* (2013) (cit. on pp. 3, 5, 32, 33).

[Mni+14] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu. “Recurrent models of visual attention.” In: *Advances in Neural Information Processing Systems*. 2014, pp. 2204–2212 (cit. on pp. 64, 83).

[Mni+15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. “Human-level control through deep reinforcement learning.” In: *Nature* 518.7540 (2015), pp. 529–533 (cit. on p. 4).

[Mni+16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. “Asynchronous methods for deep reinforcement learning.” In: *arXiv preprint arXiv:1602.01783* (2016) (cit. on pp. 3, 17).

[Mol+15] T. M. Moldovan, S. Levine, M. I. Jordan, and P. Abbeel. “Optimism-driven exploration for nonlinear systems.” In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 3239–3246 (cit. on p. 86).

[Muno6] R. Munos. “Policy gradient in continuous time.” In: *The Journal of Machine Learning Research* 7 (2006), pp. 771–791 (cit. on p. 67).

[NP90] K. S. Narendra and K. Parthasarathy. “Identification and control of dynamical systems using neural networks.” In: *IEEE Transactions on neural networks* 1.1 (1990), pp. 4–27 (cit. on p. 2).

[Nea90] R. M. Neal. “Learning stochastic feedforward networks.” In: *Department of Computer Science, University of Toronto* (1990) (cit. on p. 64).

[NH98] R. M. Neal and G. E. Hinton. “A view of the EM algorithm that justifies incremental, sparse, and other variants.” In: *Learning in graphical models*. Springer, 1998, pp. 355–368 (cit. on pp. 64, 79).

[NJoo] A. Y. Ng and M. Jordan. “PEGASUS: A policy search method for large MDPs and POMDPs.” In: *Uncertainty in artificial intelligence (UAI)*. 2000 (cit. on p. 26).

[NHR99] A. Y. Ng, D. Harada, and S. Russell. “Policy invariance under reward transformations: Theory and application to reward shaping.” In: *ICML*. Vol. 99. 1999, pp. 278–287 (cit. on pp. 45, 51, 52).

[Osb+16] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. “Deep Exploration via Bootstrapped DQN.” In: *arXiv preprint arXiv:1602.04621* (2016) (cit. on pp. 3, 86).

[Owe13] A. B. Owen. *Monte Carlo theory, methods and examples*. 2013 (cit. on p. 26).

[PR98] R. Parr and S. Russell. “Reinforcement learning with hierarchies of machines.” In: *Advances in neural information processing systems* (1998), pp. 1043–1049 (cit. on p. 85).

[PB13] R. Pascanu and Y. Bengio. “Revisiting natural gradient for deep networks.” In: *arXiv preprint arXiv:1301.3584* (2013). arXiv: `1301.3584 [cs.DG]` (cit. on p. 40).

[Pea14] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014 (cit. on p. 76).

[PMA10] J. Peters, K. Mülling, and Y. Altün. “Relative Entropy Policy Search.” In: *AAAI Conference on Artificial Intelligence*. 2010 (cit. on pp. 24, 29).

[PSo8] J. Peters and S. Schaal. “Natural actor-critic.” In: *Neurocomputing* 71.7 (2008), pp. 1180–1190 (cit. on pp. 6, 24, 27, 55, 61).

[Pir+13] M. Pirotta, M. Restelli, A. Pecorino, and D. Calandriello. “Safe policy iteration.” In: *Proceedings of The 30th International Conference on Machine Learning*. 2013, pp. 307–315 (cit. on p. 29).

[Poloo] D. Pollard. *Asymptopia: an exposition of statistical asymptotic theory*. 2000. URL: `http://www.stat.yale.edu/~pollard/Books/Asymptopia` (cit. on p. 22).

[RGB13] R. Ranganath, S. Gerrish, and D. M. Blei. “Black box variational inference.” In: *arXiv preprint arXiv:1401.0118* (2013) (cit. on p. 76).

[RMW14] D. J. Rezende, S. Mohamed, and D. Wierstra. “Stochastic backpropagation and approximate inference in deep generative models.” In: *arXiv:1401.4082* (2014) (cit. on pp. 64, 66, 67, 76, 81).

[Sch+15a] J. Schulman, N. Heess, T. Weber, and P. Abbeel. “Gradient estimation using stochastic computation graphs.” In: *Advances in Neural Information Processing Systems*. 2015, pp. 3528–3536 (cit. on p. 7).

[Sch+15b] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. “High-dimensional continuous control using generalized advantage estimation.” In: *arXiv preprint arXiv:1506.02438* (2015) (cit. on p. 7).

[Sch+15c] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. “Trust region policy optimization.” In: *CoRR, abs/1502.05477* (2015) (cit. on pp. 7, 54, 55, 61).

[Sil+14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. “Deterministic Policy Gradient Algorithms.” In: *ICML*. 2014 (cit. on pp. 3, 67).

[Sil+16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. “Mastering the game of Go with deep neural networks and tree search.” In: *Nature* 529.7587 (2016), pp. 484–489 (cit. on p. 3).

[Str+06] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman. “PAC model-free reinforcement learning.” In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 881–888 (cit. on p. 85).

[SB98] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*. MIT Press, 1998 (cit. on pp. 2, 7, 49, 50, 53).

[SPS99] R. S. Sutton, D. Precup, and S. Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning.” In: *Artificial intelligence* 112.1 (1999), pp. 181–211 (cit. on p. 85).

[Sut+99] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al. “Policy gradient methods for reinforcement learning with function approximation.” In: *NIPS*. Vol. 99. Citeseer. 1999, pp. 1057–1063 (cit. on pp. 4, 16, 64, 81).

[Sze10] C. Szepesvári. “Algorithms for reinforcement learning.” In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 4.1 (2010), pp. 1–103 (cit. on p. 2).

[SL06] I. Szita and A. Lörincz. “Learning Tetris using the noisy cross-entropy method.” In: *Neural computation* 18.12 (2006), pp. 2936–2941 (cit. on pp. 4, 11, 31).

[TZS04] R. Tedrake, T. Zhang, and H. Seung. "Stochastic policy gradient reinforcement learning on a simple 3D biped." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2004 (cit. on p. 32).

[Tes95] G. Tesauro. "Temporal difference learning and TD-Gammon." In: *Communications of the ACM* 38.3 (1995), pp. 58–68 (cit. on p. 2).

[Tho14] P. Thomas. "Bias in natural actor-critic algorithms." In: *Proceedings of The 31st International Conference on Machine Learning*. 2014, pp. 441–448 (cit. on p. 47).

[TET12] E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A physics engine for model-based control." In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 5026–5033 (cit. on pp. 30, 57).

[VHGS15] H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double Q-learning." In: *CoRR, abs/1509.06461* (2015) (cit. on p. 3).

[VR+97] B. Van Roy, D. P. Bertsekas, Y. Lee, and J. N. Tsitsiklis. "A neuro-dynamic programming approach to retailer inventory management." In: *Decision and Control, 1997., Proceedings of the 36th IEEE Conference on*. Vol. 4. IEEE. 1997, pp. 4052–4057 (cit. on p. 1).

[Vla+09] N. Vlassis, M. Toussaint, G. Kontes, and S. Piperidis. "Learning model-free robot control by a Monte Carlo EM algorithm." In: *Autonomous Robots* 27.2 (2009), pp. 123–130 (cit. on p. 79).

[WP09] K. Wampler and Z. Popović. "Optimal gait and form for animal locomotion." In: *ACM Transactions on Graphics (TOG)*. Vol. 28. 3. ACM. 2009, p. 60 (cit. on pp. 4, 32).

[Waw09] P. Wawrzyński. "Real-time reinforcement learning by sequential actor–critics and experience replay." In: *Neural Networks* 22.10 (2009), pp. 1484–1497 (cit. on pp. 45, 46).

[Wie+08] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. "Natural evolution strategies." In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, pp. 3381–3387 (cit. on p. 4).

[Wie+10] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. "Recurrent policy gradients." In: *Logic Journal of IGPL* 18.5 (2010), pp. 620–634 (cit. on pp. 64, 83).

[Wil92] R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Machine learning* 8.3-4 (1992), pp. 229–256 (cit. on pp. 4, 16, 64, 67, 76, 81, 82).

[WW13] D. Wingate and T. Weber. "Automated variational inference in probabilistic programming." In: *arXiv preprint arXiv:1301.1299* (2013) (cit. on p. 76).

[WN99] S. J. Wright and J. Nocedal. *Numerical optimization*. Vol. 2. Springer New York, 1999 (cit. on pp. 41, 42, 54, 73).

[ZS15] W. Zaremba and I. Sutskever. "Reinforcement Learning Neural Turing Machines." In: *arXiv preprint arXiv:1505.00521* (2015) (cit. on p. 64).