McGill University
Department of Computer Science
Ph.D. Comprehensive Exam
Policy Gradient Methods for Reinforcement Learning

Due on Monday, Aug 17, 2015

Eric Crawford
260609418

1 Introduction

Finding an optimal strategy for acting in an environment based on past interactions with that environment is a thread that links a wide array of research fields. It is important to neuroscientists and psychologists, for example, because it is a problem brains must be able to solve in order to adapt to new environments. It is important to computer scientists because machines that can solve this problem robustly would be highly useful. Consequently, its mathematical formulation, known as reinforcement learning (RL), has been a focus of intense academic study (sometimes under other names such as "trial-and-error learning", "neuro-dynamic programming" or "optimal control") since as far back as the 1950's [1].

Many useful algorithms have been derived, often with strong theoretical guarantees, since that time. However, most of these algorithms and their theoretical results apply only to discrete environments; not until the relatively recent proliferation of cheap computers and robotics has it been necessary (or even possible) to study algorithms that are capable of scaling up to the high-dimensional, continuous state spaces most often encountered in real world problems. A recent class of algorithms, inspired by the success of gradient-based optimization in other areas of machine learning, have recently been shown, both empirically and theoretically, to scale gracefully to these difficult environments [2]. Such *policy gradient* algorithms are the focus of this study.

To give some brief intuition, suppose we are interested in obtaining an agent that is capable of safely driving a car. The environment constitutes the car and the surrounding world (the road, traffic signs, pedestrians, other cars, etc). We seek a policy for acting in this environment; that is, we seek a strategy for moving the steering wheel and pedals in response to the current state of the environment. In particular, we seek a policy which results in high reward, where reward is assigned for events like staying in between the lines of the road, avoiding obstacles, and getting to the goal location without causing an accident. We will see that given an environment, it is often possible to define a function that maps each policy to the expected reward when acting according to that policy in the environment, and that certain tricks can be used to make this function differentiable. We can then find a policy with locally optimal performance (expected reward) by simply following the gradient of this function. The difficult part, it turns out, is coming up with tractable ways of computing this gradient.

This study is organized as follows. Section 2 covers the basics of Reinforcement Learning. Section 3 introduces the basics of policy gradient algorithms. Sections 4 and 5 outline a core class of policy gradient algorithms based on the so-called "likelihood-ratio" trick. Section 6 covers algorithms which follow the *natural gradient*, an alternate formulation of the concept of gradient which has several advantages over the standard gradient. Section 7 investigates off-policy policy gradient algorithms, which allow learning about one policy while following another. Section 8 looks at policy gradient algorithms for classes of *deterministic* policies, in contrast to the stochastic policy classes handled by most other methods. Section 9 briefly covers finite-difference methods, a largely separate class of algorithms that originated in the stochastic simulation community. Finally, Section 10 summarizes modern improvements which build on this foundational work. We conclude by providing a summary of policy gradient methods and briefly discussing open problems in the field.

2 Preliminaries

2.1 Markov Decision Processes

Throughout this study an environment is modelled as a Markov Decision Processes (MDP) [3]. An MDP is a tuple $\langle S, A, T, r, d_0 \rangle$. Here S is a set of states and A is a set of actions. $T : S \times S \times A \to \mathbb{R}$ is a function such that $T(s'|s, a)$ gives the probability of transitioning to state s' given that the current state is s and the agent executes action a. Note that the next state is conditionally independent of all other variables given the current state and action, which is known as the Markov property. $r : S \times A \to \mathbb{R}$ is a function such

that $r(s, a)$ gives the reward yielded to the agent upon taking action a in state s, and d_0 is a probability distribution over states from which the initial state is chosen.

The agent that interacts with the environment is modelled as a policy. A policy is a function $\pi : A \times S \to \mathbb{R}$ where $\pi(a|s)$ gives the probability that the agent chooses action a in state s. We require that $\pi(a|s) \in (0, 1)$ for all s and a and $\int_A \pi(a|s)da = 1$ so that $\pi(\cdot|s)$ is a probability distribution for all s. We require $\pi(a|s) > 0$ because it will often appear as a denominator.

Interaction between the agent and the environment proceeds in discrete time steps beginning with $t = 0$. The initial state s_0 is chosen according to d_0. The agent then selects action a_0 according to $\pi(\cdot|s_0)$ and receives a reward $r(s_0, a_0)$. Next, s_1 is selected according to $T(\cdot|s_0, a_0)$, after which the agent selects a_1 according to $\pi(\cdot|s_1)$, and so on. A sequence of H interactions $\tau = (s_0, a_0, \ldots, s_{H-1}, a_{H-1}, s_H)$ is called a trajectory. For a given environment, the trajectory length or horizon H is fixed. The *return* associated with a trajectory is given by $R(\tau) = \sum_{t=0}^{H-1} \gamma^t r(s_t, a_t)$. $\gamma \in (0, 1]$ is called a *discount factor* and controls the extent to which value is placed on future rewards; smaller discount factors place progressively less value on temporally distant rewards. H is allowed to be infinite only if $\gamma < 1$, a restriction which ensures that the return converges. Tasks with finite H are called *episodic*, while tasks with infinite H are called *continuing*.

Let $J(\pi)$ be the expected return using policy π in an MDP $M = \langle S, A, T, r, d_0 \rangle$:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t)\right]$$

Now, given some fixed policy class Π, the general reinforcement learning problem is to find π^* such that $\pi^* = \text{argmax}_{\pi \in \Pi} J(\pi)$. All algorithms covered in this study assume that agent does not know T or r; it must make due with the indirect information about T and r provided by interactions with the environment.

2.2 Value and Action-value Functions

The value and action-value functions are core components of most algorithms for finding and characterizing high-return policies when H is infinite. The value function for a policy π maps each $s \in S$ to the expected return when following π given we start in state s:

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s\right] = \mathbb{E}\left[r(s_0, a_0) + \gamma V(s_1) | s_0 = s\right]$$

The action-value function is defined similarly:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s, a_0 = a\right] = r(s, a) + \gamma \mathbb{E}\left[Q(s_1, a_1) | s_0 = s, a_0 = a\right]$$

From these definitions, a number of useful relations immediately follow. For instance, $J(\pi) = \mathbb{E}_{s \sim d_0}\left[V^\pi(s)\right]$ and $V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)}\left[Q^\pi(s, a)\right]$.

2.3 Temporal-Difference Learning

Temporal-difference (TD) learning is a popular way to learn value and action-value functions for a given environment/policy combination when T and r are not known. The central idea of TD learning is improving past predictions based on current predictions [4]. In Reinforcement Learning, we are interested in predicting return based on the current state (when learning the value function) or the current state-action pair (when learning the action-value function).

We first consider learning the value function. Suppose that at time t an agent is in state s_t, and it executes action a_t according to its policy $\pi(\cdot|s_t)$. This yields a reward $r(s_t, a_t)$ and causes a transition into

a new state s_{t+1}. The agent now has access to two temporally separated predictions of the value of state s, namely $V(s)$ and $r(s_t, a_t) + \gamma V(s_{t+1})$. The latter of these should be more accurate as it is based on more data, so we should modify $V(s)$ so that it is more like $r(s_t, a_t) + \gamma V(s_{t+1})$. When S and A are discrete and the value function is represented as a lookup table, we can use the equation:

$$V_{t+1}(s_t) = V_t(s_t) + \beta_t(r(s_t, a_t) + \gamma V_t(s_{t+1}) - V_t(s_t)) \tag{1}$$

where $V_t(s)$ gives our estimate of $V(s)$ at time t, the quantity inside the brackets is called the *temporal-difference error* or TD error, and $\{\beta_t\}$ is a sequence of learning rates which satisfy the *stochastic approximation (SA)* conditions [1]:

$$\beta_t > 0 \qquad \sum_t \beta_t = \infty \qquad \sum_t \beta_t^2 < \infty \tag{2}$$

A similar update can be used to learn the action-value function:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t) + \beta_t(r(s_t, a_t) + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t))$$

When S and A are continuous, V and Q cannot be represented by a lookup table as there are now uncountably many states and action. Instead, we must use parameterized function approximators V_u and Q_w, parameterized by vectors u and w respectively. TD learning then consists of updating the parameter vectors based on the TD error:

$$\begin{aligned} u_{t+1} &= u_t + \beta_t(r(s_t, a_t) + \gamma V_{u_t}(s_{t+1}) - V_{u_t}(s_t))\frac{\partial V_{u_t}(s_t)}{\partial u} \\ w_{t+1} &= w_t + \beta_t(r(s_t, a_t) + \gamma Q_{w_t}(s_{t+1}, a_{t+1}) - Q_{w_t}(s_t, a_t))\frac{\partial Q_{w_t}(s_t, a_t)}{\partial w} \end{aligned} \tag{3}$$

2.4 Greedy Reinforcement Learning

Most RL algorithms operate on the principle of *Generalized Policy Iteration (GPI)* [1]. Such algorithms work by repeatedly alternating between *policy evaluation* and *policy improvement* steps. The policy evaluation step entails finding the value or action-value function for the policy currently under consideration, usually using TD learning. The policy improvement step uses the information gleaned during the evaluation step to find a policy that improves upon the current one.

Many of the original RL algorithms employ a *greedy* policy improvement step [1]. That is, they choose a new policy which, in every state, picks the action that maximizes the action-value function of the current policy in that state. Using this scheme, the policy for the next GPI iteration is completely determined by the action-value function for the current GPI iteration. Consequently, we do not need to explicitly maintain a policy; we can view it as implicitly defined by the action-value function. SARSA (state-action-reward-state-action) is a popular algorithm along these lines [1]. Each time step, SARSA updates its estimate of the action-value function using Equation (3), and then chooses a new action by selecting the action which has the highest estimated Q value in the current state (possibly with a small chance of choosing other actions to provide exploration).

When the first policy gradient algorithms came out, it was not known whether it could be guaranteed that SARSA will eventually converge to a fixed policy when using function approximation. Hence, policy gradient algorithms, which can be shown to converge with linear function approximation, were seen as a major improvement. However, Perkins and Precup [5] showed that SARSA is convergent with linear function approximation, somewhat mitigating that advantage. Fortunately, policy gradient techniques still have the advantage of connecting RL with the wealth of gradient-based innovations coming from the field of supervised learning (such as the natural gradient, see Section 6). Policy gradient methods also provide increased flexibility in the range of policies that are permitted compared to greedy techniques, and provide

a means of including prior knowledge about the environment by selecting the policy class appropriately.

2.5 Actor-Critic Algorithms

Actor-critic algorithms constitute another set of GPI algorithms [1, 6, 7]. Similar to greedy methods, actor-critic methods use temporal-difference learning to maintain an estimate of the action-value function for the current policy. They differ from greedy methods in how they implement their policy improvement step. Instead of defining the policy for the next iteration implicitly in terms of the action-value function learned on the current iteration, actor-critic algorithms represent the current policy explicitly, separate from the action-value function. In particular, actor-critic algorithms work with policies that are parameterized by a real-valued vector, θ, with each value for θ mapping to a policy π_θ. The name "actor-critic" comes from this dichotomy: the actor is the policy itself, and the critic is the module which maintains an estimate of the actor's performance, providing information that the actor can use to update itself and improve its performance.

If state and action spaces are assumed to be discrete, then a common class of parameterized policies for use in actor-critic architectures is the lookup table softmax (a.k.a. Gibbs or Boltzmann) [1]. Policies in this class have the form:

$$\pi_\theta(a|s) = \frac{e^{\theta(s,a)}}{\sum_{b \in A} e^{\theta(s,b)}}$$

assuming that the parameter vector θ is used as a lookup table which can be indexed by state-action pairs.

In most actor-critic algorithms, the role of the critic is to maintain an estimate of the value function for the actor's current policy using temporal-difference learning. The critic's learning is thus accomplished using Equation (1). Different actor-critic algorithms use different techniques for updating the actor's policy. If we are in state s and take action a, arriving in state s' and receiving reward r, then if s' seems like a "good" state, we should update θ such that the actor is more likely to choose a next time it is in state s, and vice versa for "bad" states. Consequently, one of the earliest actor-critic algorithms used the update:

$$\theta(s,a) = \theta(s,a) + \alpha_t \left(r + \gamma V(s')\right)$$

where $\{\alpha_t\}$ is the actor's learning rate sequence which satisfies the SA conditions (2). Subsequently, Sutton [7] (which coined the term "actor-critic") explored using the update:

$$\theta(s,a) = \theta(s,a) + \alpha_t \left(r + \gamma V(s') - V(s)\right)$$

and found it to have much better performance than the original update method. An intuitive explanation for this is that $r + \gamma V(s') - V(s)$ can be regarded as an unbiased estimate of $Q(s,a) - V(s)$. It thus matches more closely with what we want to quantify, namely how much better it is to take action a in state s than the other actions. Both actor and critic perform a learning update on every time step.

Actor-critic algorithms are two-time scale algorithms, which means that each of the two components uses a different learning rate sequence ($\{\alpha_t\}$ and $\{\beta_t\}$ for the actor and critic, respectively). Many actor critic algorithms (e.g. Konda and Tsitsiklis [8]) require that the critic learn faster than the actor, and, in particular, that $\frac{\alpha_t}{\beta_t} \to 0$. With this satisfied, when the actor changes its policy the critic is able to learn the value function for the new policy quickly, allowing the actor to make its next update using (approximately) correct information [6].

We introduce actor-critic algorithms here because many of the policy gradient algorithms we will encounter *are* in fact actor-critic algorithms. However, the policy gradient algorithms generally update the parameters of the actor in more principled ways (i.e. following the gradient of the objective with respect to θ), and may TD-learn different quantities in the critic.

3 The Policy Gradient

With the preliminaries out of the way, we are finally in a position to introduce policy gradient algorithms. As in the case of actor-critic algorithms, we will always work with policy classes parameterized by a real-valued vector θ, i.e. $\Pi = \{\pi_\theta | \theta \in \mathbb{R}^{|\theta|}\}$.

We have established that the goal in reinforcement learning is to find the policy which yields the largest expected return, i.e. find $\pi^* = \text{argmax}_{\pi \in \Pi} J(\pi)$. Since each parameter value maps to a policy, we can treat our objective function J as a function of θ instead of π_θ. The RL problem is then:

$$\theta^* = \text{argmax}_\theta J(\theta)$$

$J(\pi)$ is a differentiable function, so if π_θ is a differentiable function of θ, then $J(\theta)$ is a differentiable function. We can thus optimize this function using the standard method of (first-order) gradient ascent. Given some initial θ_0, we choose successive values of θ according to the rule:

$$\theta_{t+1} = \theta_t + \alpha_t \frac{\partial J(\theta_t)}{\partial \theta}$$

where $\{\alpha_t\}$ satisfies the usual SA conditions (2). This leaves the question of calculating $\frac{\partial J(\theta_t)}{\partial \theta}$. One approach is to write $J(\theta)$ as an expectation over trajectories of length H:

$$J(\theta) = \mathbb{E}_{\tau \sim P_{\pi_\theta}} [R(\tau)] = \int P_{\pi_\theta}(\tau) R(\tau) d\tau$$

where $P_\pi(\tau) = d_0(s_0^\tau) \prod_{t=0}^{H-1} \pi(a_t^\tau | s_t^\tau) T(s_{t+1}^\tau | s_t^\tau, a_t^\tau)$. We could then attempt to differentiate straightforwardly:

$$\frac{\partial J(\theta)}{\partial \theta} = \int \frac{\partial P_{\pi_\theta}(\tau)}{\partial \theta} R(\tau) d\tau \tag{4}$$

Unfortunately, calculating this exactly would require summing/integrating over all trajectories for every gradient computation, which is clearly infeasible (e.g. there are $|S|^H |A|^H$ trajectories in the discrete case). Even more problematically, it requires having acces to the environment model T which is usually not available. The bulk of the research on policy gradient algorithms involves finding computationally tractable ways of approximating this gradient.

4 REINFORCE and the Likelihood-Ratio Trick

Williams [9] was among the first to explicitly consider using gradient methods in reinforcement learning. The product of his efforts was the REINFORCE algorithm, which is still widely used today for its simplicity and wide applicability. REINFORCE uses what is known as the likelihood-ratio trick, combined with the weak law of large numbers, to approximate the intractable integral in Equation (4) using sample trajectories obtained by acting according to policy π_θ, assuming we are interested in evaluating the gradient at parameter value θ.

We begin by deriving two identities that will be used repeatedly throughout the remainder of the study. Both are elementary, but it will be useful to state them explicitly. The first follows from simple properties of derivatives:

$$\frac{\partial \log f(\theta)}{\partial \theta} = \frac{1}{f(\theta)} \frac{\partial f(\theta)}{\partial \theta}$$

where log denotes the natural logarithm and f is a function such that both f and $\log f$ are differentiable.

The second identity is:

$$\begin{aligned}\frac{\partial \log P_{\pi_\theta}(\tau)}{\partial \theta} &= \frac{\partial}{\partial \theta}\left(\log\left(d_0(s_0^\tau)\prod_{t=0}^{H-1} T(s_{t+1}^\tau|s_t^\tau,a_t^\tau)\pi_\theta(a_t^\tau|s_t^\tau)\right)\right)\\ &= \frac{\partial}{\partial \theta}\left(\log d_0(s_0^\tau) + \sum_{t=0}^{H-1}\log T(s_{t+1}^\tau|s_t^\tau,a_t^\tau) + \sum_{t=0}^{H-1}\log \pi_\theta(a_t^\tau|s_t^\tau)\right)\\ &= \sum_{t=0}^{H-1}\frac{\partial \log \pi_\theta(a_t^\tau|s_t^\tau)}{\partial \theta}\end{aligned}$$

where the last equality uses the fact that neither T nor d_0 depends on θ. We can now use the likelihood-ratio trick to turn Equation (4), the naive expression of the gradient of $J(\theta)$, into an expression that can be estimated using sampled trajectories:

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta} &= \int \frac{\partial P_{\pi_\theta}(\tau)}{\partial \theta} R(\tau) d\tau\\ &= \int \frac{P_{\pi_\theta}(\tau)}{P_{\pi_\theta}(\tau)}\frac{\partial P_{\pi_\theta}(\tau)}{\partial \theta} R(\tau) d\tau &(5)\\ &= \int P_{\pi_\theta}(\tau)\frac{\partial \log P_\theta(\tau)}{\partial \theta} R(\tau) d\tau &(6)\\ &= \int P_{\pi_\theta}(\tau)\left(\sum_{t=0}^{H-1}\frac{\partial \log \pi_\theta(a_t^\tau|s_t^\tau)}{\partial \theta}\right) R(\tau) d\tau &(7)\\ &= \mathbb{E}_{\tau\sim P_{\pi_\theta}}\left[\left(\sum_{t=0}^{H-1}\frac{\partial \log \pi_\theta(a_t^\tau|s_t^\tau)}{\partial \theta}\right)\left(\sum_{t=0}^{H-1}\gamma^t r(s_t^\tau,a_t^\tau)\right)\right] &(8)\end{aligned}$$

Line (5) is the likelihood-ratio trick itself and lines (6) and (7) follow from applying identities 1 and 2, respectively. We can now use the weak law of large numbers to estimate this expectation from trajectories sampled following policy π_θ:

$$\frac{\partial J(\theta)}{\partial \theta} \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{H-1}\frac{\partial \log \pi_\theta(a_t^{\tau_i}|s_t^{\tau_i})}{\partial \theta}\right)\left(\sum_{t=0}^{H-1}\gamma^t r(s_t^{\tau_i},a_t^{\tau_i})\right) \qquad \text{where } \tau_i \sim P_{\pi_\theta} \tag{9}$$

All quantities in this estimator are readily available since we observe the reward $r(s,a)$ as we interact with the environment, and we have assumed that π_θ is a differentiable function of θ.

We note that, in principle, the likelihood-ratio trick can be applied using a sampling distribution P other than P_{π_θ}. However, one must choose P with care because, as we will see in Section 7, the variance of gradient estimator can grow quickly as P and P_{π_θ} diverge.

5 The Policy Gradient Theorem

The Policy Gradient Theorem (PGT) improves upon the REINFORCE algorithm in a number of ways. For one, it applies to infinite-horizon problems, unlike REINFORCE. Furthermore, PGT takes advantage of the fact that actions only affect rewards encountered at later times (not earlier ones) to obtain a gradient estimator with lower variance. This can be derived formally from the REINFORCE gradient expression. We

first note that we can rearrange (8) to obtain:

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta} &= \mathbb{E}_{\tau \sim P_{\pi_\theta}}\left[\sum_{k=0}^{H-1}\sum_{\ell=0}^{H-1}\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}\gamma^k r(s_k^\tau, a_k^\tau)\right] \\ &= \sum_{k=0}^{H-1}\sum_{\ell=0}^{H-1}\mathbb{E}_{\tau\sim P_{\pi_\theta}}\left[\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}\gamma^k r(s_k^\tau, a_k^\tau)\right]\end{aligned} \tag{10}$$

Now we consider terms of this expression wherein $k < \ell$ (meaning the action comes after the reward):

$$\begin{aligned}&\mathbb{E}_{\tau\sim P_{\pi_\theta}}\left[\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}\gamma^k r(s_k^\tau, a_k^\tau)\right] \\ &= \mathbb{E}_{\tau_{k:\ell}}\left[\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}\gamma^k r(s_k^\tau, a_k^\tau)\right] \\ &= \int P_{\pi_\theta}(\tau_{k:\ell})\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}\gamma^k r(s_k^\tau, a_k^\tau)\ d\tau_{k:\ell} \\ &= \int \pi_\theta(a_\ell|s_\ell)T(s_\ell|s_{\ell-1}, a_{\ell-1})P_{\pi_\theta}(\tau_{k:\ell-1})\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}\gamma^k r(s_k^\tau, a_k^\tau)\ d\tau_{k:\ell} \\ &= \int T(s_\ell|s_{\ell-1}, a_{\ell-1})P_{\pi_\theta}(\tau_{k:\ell-1})\gamma^k r(s_k^\tau, a_k^\tau)\left(\int \pi_\theta(a_\ell|s_\ell)\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}da_\ell\right)\ ds_\ell\ d\tau_{k:\ell-1}\end{aligned}$$

But notice that making using of identity 1, we have:

$$\int \pi_\theta(a_\ell|s_\ell)\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}da_\ell = \int \frac{\partial \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}da_\ell = \frac{\partial}{\partial\theta}\left(\int \pi_\theta(a_\ell^\tau|s_\ell^\tau)da_\ell\right) = \frac{\partial}{\partial\theta}(1) = 0$$

Thus, all terms in Equation (10) wherein $k < \ell$ have mean 0, and we get:

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\tau\sim P_{\pi_\theta}}\left[\sum_{\ell=0}^{H-1}\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}\left(\sum_{k=\ell}^{H-1}\gamma^k r(s_k^\tau, a_k^\tau)\right)\right]$$

This can be used once again with the weak law of large numbers to derive an estimator with lower variance than the one given to us by REINFORCE. However, we can also go one further and derive an algorithm that is more appropriate for infinite-horizon problems. Assume $H = \infty$. Let $d_\ell^{\pi_\theta}(s)$ be the probability that the agent is in state s after acting according to policy π_θ for ℓ steps. Then $d^{\pi_\theta}(s) = (1-\gamma)\sum_{\ell=0}^{\infty}\gamma^\ell d_\ell^{\pi_\theta}(s)$ is the long term discounted distribution over states when following policy π_θ. Now:

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta} &= \mathbb{E}_{\tau\sim P_{\pi_\theta}}\left[\sum_{\ell=0}^{\infty}\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}\left(\sum_{k=\ell}^{\infty}\gamma^k r(s_k^\tau, a_k^\tau)\right)\right] \\ &= \mathbb{E}_{\tau\sim P_{\pi_\theta}}\left[\sum_{\ell=0}^{\infty}\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}\gamma^\ell\left(\sum_{k=\ell}^{\infty}\gamma^{k-\ell} r(s_k^\tau, a_k^\tau)\right)\right] \\ &= \mathbb{E}_{\tau\sim P_{\pi_\theta}}\left[\sum_{\ell=0}^{\infty}\frac{\partial \log \pi_\theta(a_\ell^\tau|s_\ell^\tau)}{\partial\theta}\gamma^\ell Q(s_\ell, a_\ell)\right] \\ &= \sum_{\ell=0}^{\infty}\mathbb{E}_{(s_\ell,a_\ell)\sim P_{\pi_\theta}}\left[\frac{\partial \log \pi_\theta(a_\ell|s_\ell)}{\partial\theta}\gamma^\ell Q(s_\ell, a_\ell)\right] \\ &= \sum_{\ell=0}^{\infty}\int_S\int_A \gamma^\ell d_\ell^{\pi_\theta}(s)\pi_\theta(a|s)\frac{\partial \log \pi_\theta(a|s)}{\partial\theta}Q(s,a)\ da\ ds \\ &\propto \int_S d^{\pi_\theta}(s)\int_A \frac{\partial \pi_\theta(a|s)}{\partial\theta}Q(s,a)\ da\ ds\end{aligned} \tag{11}$$

This is the content of the Policy Gradient Theorem [10], and Konda and Tsitsiklis [8] arrived at a similar result independently.

5.1 Compatible Function Approximation

The formula for the policy gradient given by the PGT requires the action-value function $Q(s,a)$ to be known, which will typically not be the case; it will usually have to be estimated from experience. If working in a continuous state space, function approximation will be necessary. In general, function approximation introduces a source of error, and it is possible that this error could bias our gradient estimates when our approximation $f(s,a)$ is substituted for $Q(s,a)$ in Equation (11).

Fortunately, it can be shown that function approximators that are linear in $\frac{\partial \log \pi_\theta(a|s)}{\partial \theta}$ are compatible with the PGT, in the sense that they will not bias the gradient estimates [8, 10]. In particular, let us define $f_w(s,a) = w^T \frac{\partial \log \pi(s|a)}{\partial \theta}$ and choose w satisfying

$$\int_S d_{\pi_\theta}(s) \int_A \pi_\theta(s,a)[Q_{\pi_\theta}(s,a) - f_w(s,a)] \frac{\partial f_w(s,a)}{\partial w} \, da \, ds = 0$$

which implies that w (locally) minimizes the mean squared error $\frac{1}{2} \int_S d_{\pi_\theta}(s) \int_A \pi_\theta(s,a) \left(Q(s,a) - f_w(s,a)\right)^2 dads$. Then it can be shown that:

$$\frac{\partial J(\theta)}{\partial \theta} = \int_S d_{\pi_\theta}(s) \int_A \frac{\partial \pi_\theta(a|s)}{\partial \theta} f_w(s,a) \, da \, ds \tag{12}$$

We can now derive a gradient based actor-critic algorithm [8]. We assume the critic learns using Equation (3) with $f_w(s,a)$ substituted for $Q_w(s,a)$. In the average reward case, the actor is assumed to update its parameters according to:

$$\theta_{t+1} = \theta_t + \alpha_t \Gamma(w) f_w(s_t, a_t) \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta}$$

where $\Gamma(w)$ is a normalizing function. See Konda and Tsitsiklis [8] for more details.

One caveat is that the restriction that $\frac{\partial \log \pi_\theta(a|s)}{\partial \theta}$ be used as features for $f_w(s,a)$ restricts $f_w(s,a)$ to have mean 0 for all states, since:

$$\begin{aligned}
\mathbb{E}_{a \sim \pi_\theta(\cdot|s)} \left[f_w(s,a) \right] &= w^T \mathbb{E}_{a \sim \pi_\theta(\cdot|s)} \left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} \right] \\
&= w^T \left(\int_A \pi_\theta(a|s) \frac{\partial \log \pi_\theta(a|s)}{\partial \theta} da \right) \\
&= w^T \left(\int_A \frac{\partial \pi_\theta(a|s)}{\partial \theta} da \right) = w^T \frac{\partial}{\partial \theta} \left(\int_A \pi_\theta(a|s) da \right) = 0
\end{aligned}$$

It is possible to directly use $f_w(s,a)$ to approximate $Q(s,a)$, though this 0 mean restriction can make this task difficult since $Q(s,a)$ will generally not be 0 mean [11]. An alternative is to make $f_w(s,a)$ an approximation of the *advantage* function, which does have 0 mean:

$$A(s,a) = Q(s,a) - \mathbb{E}_{a \sim \pi(\cdot|s)}[Q(s,a)] = Q(s,a) - V(s)$$

Sutton et al. [10] provide additional results showing that if $f_w(s,a)$ is used to approximate $A(s,a)$ instead of $Q(s,a)$, then Equation (12) still holds. Estimating $A(s,a)$ using temporal-difference learning is more involved than doing the same for $Q(s,a)$ since it also requires simultaneously estimating the value function; such an algorithm can be found in Peters and Schaal [11], but is beyond the scope of this study.

6 The Natural Policy Gradient

One drawback of standard policy gradient algorithms is that the gradient direction, the direction of steepest ascent of the objective function, is dependent on the parameterization of the policy.

6.1 Motivation

Formally, in finding the gradient of $J(\theta)$ with respect to θ, we are finding a vector $d\theta$ such that $J(\theta + d\theta)$ is maximized, under the restriction that $\|d\theta\|_p^2 = \epsilon$ for ϵ a small positive constant. This means that the value of the gradient is dependent on the domain space of $J(\theta)$, and, in particular, which norm $\|\cdot\|_p$ is used in that space. The standard gradient that we have used thus far uses the Euclidean norm $\|d\theta\|_2 = \sqrt{d\theta^T d\theta}$, and one consequence of this is that the standard gradient direction is dependent on how our policies are parameterized; it is said to be *non-covariant* [12]. More concretely, suppose we have two policy classes Π_1 and Π_2 which contain the same set of policies, but are related to their parameter spaces in different ways. For two policies $\pi_{\theta_1} \in \Pi_1$ and $\pi_{\theta_2} \in \Pi_2$ which induce the same distribution over trajectories (that is, they are the same policy), it will often happen that $\frac{\partial J_1(\theta_1)}{\partial \theta}$ and $\frac{\partial J_2(\theta_2)}{\partial \theta}$ point in different directions (i.e. moving in the direction of $\frac{\partial J_1(\theta_1)}{\partial \theta}$ yields a different set of effective policies than moving in the direction of $\frac{\partial J_2(\theta_2)}{\partial \theta}$). It is thus difficult to stand by the standard gradient as the "direction of steepest ascent" [13].

The key to solving the non-covariance problem is to use a norm in the domain space which only takes into account the differences between the distributions over trajectories P_{π_θ} and $P_{\pi_{\theta+d\theta}}$, and disregards the distance between the parameter vectors θ and $\theta + d\theta$, which is dependent on the parameterization. This makes sense when we recall that our objective function is:

$$J(\theta) = \sum_\tau P_{\pi_\theta}(\tau) R(\tau)$$

The value of the objective function for a policy is completely determined by the probability distribution over trajectories that the policy induces, and it is therefore the size of the change to this distribution which should be taken into account when measuring the size of a policy perturbation $d\theta$.

6.2 Derivation

The natural gradient was first introduced into the machine learning literature by Amari [14] for the case of supervised learning. It was subsequently applied to derive an improved policy gradient algorithm for infinite-horizon average reward reinforcement learning by Kakade [12] (average reward is an alternative to discounted reward for infinite-horizon problems). Bagnell and Schneider [13] put this algorithm on firm theoretical ground (it was originally suggested as a heuristic), and extended it to finite-horizon and infinite-horizon discounted problems. Here we present the derivation for finite-horizon problems. The usual way to derive the natural gradient uses Riemannian probability manifolds and the methods of information geometry [15], which we do not have the space to develop here. Instead, we present a shorter derivation based on a second-order Taylor approximation to the KL-divergence between distributions over trajectories.

We have stated that we would like to define a norm in the domain of $J(\theta)$ which only takes into account the distributions over trajectories induced by the policies. One well known measure of the difference between two distributions is the KL divergence, which, for two distributions P and Q, is defined as:

$$KL(P||Q) = -\int P(\tau) \log \frac{Q(\tau)}{P(\tau)} d\tau$$

Ideally, we would like to use the KL-divergence to define a norm measuring the size of the perturbation $d\theta$, but which only takes into account the difference between the probability distributions P_θ and $P_{\theta+d\theta}$. Unfortunately, the KL-divergence between P_θ and $P_{\theta+d\theta}$ cannot be used as such a norm, because it will not in general satisfy the scalar multiplication requirement of norms. One alternate possibility is to use a second

order approximation to the KL-divergence. Define the function $D_\theta(\theta') = KL(P_{\pi_\theta}||P_{\pi_{\theta'}})$, and consider the second-order Taylor approximation of D_θ about θ:

$$\tilde{D}_\theta(\theta') = D_\theta(\theta) + \frac{\partial D_\theta}{\partial \theta'}(\theta)(\theta' - \theta) + \frac{1}{2}(\theta' - \theta)^T \frac{\partial^2 D_\theta}{\partial \theta'^2}(\theta)(\theta' - \theta)$$

The KL divergence of a distribution with itself is 0, so $D_\theta(\theta) = KL(P_{\pi_\theta}||P_{\pi_\theta}) = 0$. Another basic fact is that the global minimum of $D_\theta(\theta')$ is θ, and since the gradient of a function is 0 at its global minima, we have $\frac{\partial D_\theta}{\partial \theta'}(\theta) = 0$. Thus, letting $G(\theta) = \frac{\partial^2 D_\theta}{\partial \theta'^2}(\theta)$, we have:

$$\widetilde{KL}(P_\theta||P_{\theta'}) = \tilde{D}_\theta(\theta') = (\theta' - \theta)^T G(\theta)(\theta' - \theta) = d\theta^T G(\theta) d\theta$$

It can be shown that $G(\theta)$ is the well-known Fisher information matrix of the probability distribution P_{π_θ}:

$$G(\theta) = E_{\tau \sim P_{\pi_\theta}} \left[\frac{\partial \log P_{\pi_\theta}(\tau)}{\partial \theta} \frac{\partial \log P_{\pi_\theta}(\tau)}{\partial \theta}^T \right]$$

We restrict ourselves now to cases where $G(\theta)$ is positive definite (pathological cases wherein $G(\theta)$ is singular and thus only positive semi-definite are possible), and thus $\|d\theta\|_{G(\theta)} = \sqrt{d\theta^T G(\theta) d\theta}$ is a valid norm. This norm, which takes into account only differences between induced distributions instead of differences between parameter vectors, is the norm used to define the natural gradient in lieu of the Euclidean norm. The fact that the norm is dependent on θ is not an issue as it is only used locally.

There is one final result we need in order to derive an algorithm based on the natural policy gradient, which is how to calculate the gradient using this alternate norm. Denoting the natural gradient as $\widehat{\frac{\partial J(\theta)}{\partial \theta}}$, a Lagrange multiplier argument [13] can be used to show that:

$$\widehat{\frac{\partial J(\theta)}{\partial \theta}} = G(\theta)^{-1} \frac{\partial J(\theta)}{\partial \theta} \tag{13}$$

This result allows the derivation of an algorithm for episodic MDPs which ascends the natural gradient [13]. To estimate the natural gradient at a point θ, we estimate the standard gradient using a REINFORCE-style estimator, and simultaneously estimate the Fisher information matrix $G(\theta)$ from the trajectories sampled as part of the gradient estimation. Finally, we perform the calculation in Equation (13). This algorithm has been shown to learn dramatically faster than the standard gradient [13]. In particular, it eliminates the problem of "plateaus", which are large regions of parameter space wherein the standard gradient is nearly 0, causing standard gradient ascent to progress extremely slowly.

6.3 Natural Actor-Critic

Peters and Schaal [11] develop the Natural Actor-Critic (NAC) algorithm, which does not require the Fisher information matrix to be explicitly estimated. Here we present the version appropriate for infinite horizon problems, and in the next section we present the episodic version. First note that for infinite-horizon discounted reward problems, the Fisher information matrix at a point θ is equal to the integral of the Fisher information matrices of the action distributions for each state (i.e. $\pi_\theta(\cdot|s)$) weighted by the discounted reward distribution d_{π_θ} [13]. That is:

$$G(\theta) = \int_S d_{\pi_\theta}(s) G_{\pi_\theta(\cdot|s)}(\theta)\ ds = \int_S d_{\pi_\theta}(s) \int_A \pi_\theta(a|s) \frac{\partial \log \pi_\theta(a|s)}{\partial \theta} \frac{\partial \log \pi_\theta(a|s)}{\partial \theta}^T\ da\ ds$$

We now use an interesting result that was proved (but not directly used in an algorithm) by Kakade [12]. The result relates to compatible function approximation from our discussion of the Policy Gradient Theorem, and states:

Theorem 1. *Kakade [12]. Assume a vector $w \in \mathbb{R}^{|\theta|}$ is such that $f_w(s,a) = w^T \frac{\partial \log \pi_\theta(s,a)}{\partial \theta}$ minimizes*

$$\epsilon(w;\theta) = \int_S d^\pi(s) \int_A \pi_\theta(a|s) \left(f_w(s,a) - Q(s,a;\theta)\right)^2 \, da \, ds$$

Then $\widehat{\frac{\partial J(\theta)}{\partial \theta}} = w$.

That is, the weight vector of a compatible function approximator (as defined in the section on PGT) *is* the natural gradient. The proof of Theorem 1 is straightforward. We start from the Policy Gradient Theorem with compatible function approximation:

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta} &= \int_S d_{\pi_\theta}(s) \int_A \frac{\partial \pi_\theta(a|s)}{\partial \theta} f_w(s,a) \, da \, ds \\
&= \int_S d_{\pi_\theta}(s) \int_A \pi_\theta(a|s) \frac{\partial \log \pi_\theta(a|s)}{\partial \theta} f_w(s,a) \, ds \, ds \\
&= \int_S d_{\pi_\theta}(s) \int_A \pi_\theta(a|s) \frac{\partial \log \pi_\theta(a|s)}{\partial \theta} \left(\frac{\partial \log \pi_\theta(a|s)}{\partial \theta}^T w \right) da \, ds \\
&= \left(\int_S d_{\pi_\theta}(s) \int_A \pi_\theta(a|s) \frac{\partial \log \pi_\theta(a|s)}{\partial \theta} \frac{\partial \log \pi_\theta(a|s)}{\partial \theta}^T da \, ds \right) w \\
&= G(\theta) w
\end{aligned}$$

But we also know that:

$$\widehat{\frac{\partial J(\theta)}{\partial \theta}} = G(\theta)^{-1} \frac{\partial J(\theta)}{\partial \theta} = G(\theta)^{-1} G(\theta) w = w$$

■

Similar to the situation with PGT, this result still holds if $f_w(s,a)$ approximates the advantage function $A(s,a)$ instead of $Q(s,a)$, and approximating $A(s,a)$ can be done more easily since both $f_w(s,a)$ and $A(s,a)$ have mean 0 because of the restriction on the features used by $f_w(s,a)$. Thus NAC simply learns a vector w such that $f_w(s,a)$ approximates $A(s,a)$ using TD learning, and then moves θ in the direction of w. The algorithm for TD learning the advantage function, which is more involved than TD learning the value or action-value function, can be found in Peters and Schaal [11].

Peters and Schaal [11] also present an episodic version of their algorithm called episodic Natural Actor Critic (eNAC) which is quite straightforward. For an H-length trajectory sampled using policy π_θ, it can be shown that:

$$\sum_{t=0}^{H-1} \gamma^t A(s_t, a_t) = V(s_0) + \sum_{t=0}^{H-1} \gamma^t r(s_t, a_t)$$

This should also be true if $A(s_t, a_t)$ is replaced by $f_w(s_t, a_t)$. Thus, to calculate the natural gradient at a parameter vector θ, eNAC simply samples some number of trajectories N using the current policy π_θ, and then solves for w in the linear regression problem:

$$\left(\sum_{t=0}^{H-1} \gamma^t \frac{\partial \log \pi_\theta(a_t^{\tau_i}|s_t^{\tau_i})}{\partial \theta} \right)^T w = V(s_0^{\tau_i}) + \sum_{t=0}^{H-1} \gamma^t r(s_t^{\tau_i}, a_t^{\tau_i}) \qquad \text{where } \tau_i \sim P_{\pi_\theta}$$

The one requirement is that the initial state s_0 be deterministic, so that $V(s_0)$ can be treated as the bias term of the linear regression problem. There are $|\theta| + 1$ unknowns, so if $N = |\theta| + 1$ trajectories are sampled then the regression problem should have a well-defined solution. In this case, the runtime of eNAC is similar to that of the original natural gradient algorithm which explicitly estimates the Fisher information matrix,

since both algorithms require an inversion of a square matrix with side-length $\approx |\theta|$. Although, Peters and Schaal [11] did not compare the performance of eNAC to that of the original algorithm, a subsequent study in a slightly different setting [16] found NAC/eNAC-style algorithms to converge more quickly and to be less sensitive to hyperparameters than algorithms which explicitly estimate the Fisher information matrix.

7 Off-Policy Policy Gradient

The finite-horizon policy gradient algorithms we have seen so far, such as REINFORCE and eNAC, require new trajectories to be sampled for each new policy candidate. As soon as we move to a new policy candidate (e.g. by taking a step in the direction of the gradient), samples drawn using previous candidates are effectively thrown away. Clearly this is wasteful, as those samples, combined with knowledge about the policies with which they were sampled, contain valuable information about the environment that we should be able to leverage. Fortunately, this situation can be remedied using *importance sampling*.

Importance sampling is a well-understood technique for estimating an expectation with respect to one distribution using samples from another distribution [17]. Suppose we have two distributions P and Q over some set $\mathcal{X}$, and we are interested in estimating the quantity $\mathbb{E}_{X\sim P}[f(X)]$. Importance sampling relies on the observation that:

$$\mathbb{E}_{X\sim P}[f(X)] = \int_x P(x)f(x)dx = \int_x Q(x)\frac{P(x)}{Q(x)}f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{P(x_i)}{Q(x_i)}f(x_i) \qquad \text{where } x_i \sim Q \quad (14)$$

Unfortunately, this estimator can have high variance if the distributions P and Q are very different because the ratio $P(x)/Q(x)$ can vary wildly.

A common method for correcting this is to use *weighted* importance sampling [18], which uses the estimator:

$$\mathbb{E}_{X\sim P}[f(X)] \approx \sum_{i=1}^{N}\frac{P(x_i)}{Q(x_i)}f(x_i)\Bigg/\sum_{i=1}^{N}\frac{P(x_i)}{Q(x_i)} \qquad \text{where } x_i \sim Q \quad (15)$$

Though this estimator is biased, it has lower variance because the denominator cancels out variance in the numerator, and many authors have found the tradeoff to be worthwhile [18, 19]. For a concrete example of the difference between estimators in Equations (14) and (15), consider the case where $f(x) = C$, a constant function. In this case, the weighted estimator always yields C as its estimate, regardless of how much data is used, whereas there are no guarantees for the unweighted version.

For our purposes P is replaced by the P_{π_θ}, Q is replaced by the distribution P_π induced by an exploratory policy π, and $f(\tau) = \frac{\partial \log P_{\pi_\theta}(\tau)}{\partial\theta}R(\tau)$. Modifying the REINFORCE estimator (Equation (9)) appropriately, we obtain:

$$\begin{aligned}\frac{\partial J(\theta)}{\partial\theta} &\approx \sum_{i=1}^{N}\frac{P_{\pi_\theta}(\tau_i)}{P_\pi(\tau_i)}\left(\sum_{t=0}^{H-1}\frac{\partial\log\pi_\theta(a_t^{\tau_i}|s_t^{\tau_i})}{\partial\theta}\right)\left(\sum_{t=0}^{H-1}\gamma^t r(s_t^{\tau_i},a_t^{\tau_i})\right)\Bigg/\sum_{i=1}^{N}\frac{P_{\pi_\theta}(\tau_i)}{P_\pi(\tau_i)} \qquad \text{where } \tau_i\sim P_\pi \\ &= \sum_{i=1}^{N}\frac{\pi_\theta(a^{\tau_i}|s^{\tau_i})}{\pi(a^{\tau_i}|s^{\tau_i})}\left(\sum_{t=0}^{H-1}\frac{\partial\log\pi_\theta(a_t^{\tau_i}|s_t^{\tau_i})}{\partial\theta}\right)\left(\sum_{t=0}^{H-1}\gamma^t r(s_t^{\tau_i},a_t^{\tau_i})\right)\Bigg/\sum_{i=1}^{N}\frac{\pi_\theta(a^{\tau_i}|s^{\tau_i})}{\pi(a^{\tau_i}|s^{\tau_i})} \qquad \text{where } \tau_i\sim P_\pi\end{aligned}$$

where $\pi(a^\tau|s^\tau) = \prod_{t=0}^{H-1}\pi(a_t^\tau|s_t^\tau)$.

Using the weighted version of the estimator is especially important because Glynn [20] shows that $Var[P_{\pi_\theta}(\tau)/P_\pi(\tau)] > e^{\frac{H\varphi}{2}} - 1$, where φ is a measure of the difference between the distributions P_π and P_{π_θ} related to the KL divergence (more correctly, is it the KL-divergence of the steady-state edge-traversal distributions). Thus the variance of $P_{\pi_\theta}(\tau)/P_\pi(\tau)$ grows exponentially as P and P_{π_θ} diverge, which is likely to cause the variance in the gradient estimates to grow quickly as well.

With this expression for the gradient, many algorithms are now possible. For instance, we can collect an initial batch of trajectories using some exploratory policy. Then for any given policy, the importance-sampled gradient estimated using those samples is a deterministic function, which we can then optimize straightforwardly. An algorithm of this flavor was discussed in Peshkin and Shelton [19]. Alternatively, we could progressively build up a dataset of trajectories, gathering a small number of new trajectories using each new candidate policy. On the i-th iteration of this algorithm, the exploratory policy is a mixture of all past candidate policies, i.e. $\pi_i(a^\tau|s^\tau) = \frac{1}{i-1}\sum_{j=1}^{i-1}\pi_{\theta_j}(a^\tau|s^\tau)$. Tang and Abbeel [21] present an algorithm based on this idea. Degris et al. [22] present on Off-Policy Policy Gradient Theorem, which they use to derive the Off-Policy Actor-Critic algorithm appropriate for infinite-horizon settings. This algorithm can optimize a policy π_θ while following a fixed exploratory distribution π.

8 Deterministic Policy Gradient

All algorithms that we have encountered so far have assumed differentiable, stochastic policies, which map states to probability distributions over actions. However, deterministic policies, which simply map each state to an action, are also possible and are widely used throughout the general reinforcement learning literature. Formally, a deterministic policy μ is a function from states to actions: $\mu : S \to A$. As usual we will work with parameterized policy classes $\Pi = \{\mu_\theta|\theta \in \mathbb{R}^{|\theta|}\}$.

It is possible to define a policy gradient for deterministic policies. However, unlike stochastic policy gradients, the deterministic policy gradient requires that the action space be continuous. This is because we will need to calculate the gradient of $\mu_\theta(s)$ with respect to θ for some state s. Thus the mapping from parameter space to A defined by $\mu_\theta(s)$ for a fixed s must be a differentiable function, which requires that A be continuous. In environments where this requirement is satisfied, Silver et al. [23] provide a deterministic policy gradient theorem, which allows them to derive policy gradient algorithms that have many advantages over their stochastic counterparts. In order to circumvent the primary drawback of deterministic policies, namely their lack of exploration, the authors use techniques from the off-policy policy gradients, learning about and optimizing a deterministic policy while acting according to an stochastic exploratory policy.

Let $\mu_\theta : S \to A$ be a deterministic policy parameterized by a real-valued vector θ, $\mu_\theta(s)$ giving the action chosen by the policy in state s. The intuition for the deterministic policy gradient runs as follows. Recall our discussion of Generalized Policy Iteration algorithms for reinforcement learning, which alternate between policy evaluation and policy improvement steps. During policy evaluation $Q(s, a)$, the action-value function for the current policy, is estimated. During policy improvement, one approach is to alter the policy so as to choose the action in each state which has the largest action-value. However, if the action space is continuous, improving the policy in this way requires solving $\max_\theta Q(s, \mu_\theta(s))$ for each state s, a non-trivial global optimization. An alternative is to take only a single step in the direction of the gradient of Q with respect to θ, given by $\frac{\partial Q(s,\mu_\theta(s))}{\partial\theta} = \frac{\partial\mu_\theta(s)}{\partial\theta}\frac{\partial Q(s,\mu_\theta(s))}{\partial\mu_\theta(s)}$ using the chain rule. This is the core of the deterministic policy gradient; assuming Q is differentiable, we move θ in the direction of the gradient of Q with respect to θ. To account for multiple states, we simply average over states, weighting them by their long-term distribution. Thus, the Deterministic Policy Gradient Theorem (DPGT) [23] states:

$$\frac{\partial J(\theta)}{\partial\theta} = \int_S d_\mu(s)\frac{\partial\mu_\theta(s)}{\partial\theta}\frac{\partial Q(s,\mu_\theta(s))}{\partial\mu_\theta(s)}ds$$

Unlike the original PGT, this expression does not require integrating over actions, which means it can be estimated with fewer samples [23]. Interestingly, the authors also show that if we begin with a stochastic policy and progressively make it more deterministic by lowering its variance, then the limit of the gradient agrees with the DPGT. The authors also provide an notion of deterministic compatible function approximation, and show how to combine their algorithm with the Off-Policy Actor-Critic [22] to arrive at an algorithm with the advantages of both deterministic policies (fewer samples required for gradient estimates) and stochastic policies (sufficient exploration).

8.1 PEGASUS

The PEGASUS algorithm [24] is an alternative approach for maximizing $J(\theta)$ over deterministic policy classes in finite-horizon environments. The first step of PEGASUS is to transform the MDP $M = \langle S, A, T, r, d_0 \rangle$ into an equivalent MDP $M' = \langle S', A', T', r', d_0' \rangle$ in which all transitions are deterministic. Equivalent here means that every policy achieves the same average return in M' as it does in M; that is for every policy π, $J_M(\pi) = J_{M'}(\pi)$. Thus if we find a (locally) optimal policy for M', it is also a (locally) optimal policy for M. As we shall, the deterministic transitions can make it easier to optimize $J_{M'}$ than J_M.

PEGASUS derives M' from M in an interesting way. First, it assumes that the environment's relationship to the random number generator is known to the agent. More formally, we assume that $s_{t+1} = g(s_t, a_t, p)$, where g is a known deterministic function, and p is a vector of length $k < \infty$ with each element distributed according to $U(0, 1)$, representing the draws from the random number generator. The state space S' of M' is defined as $S' = S \times [0, 1]^\infty$. That is, S' is obtained by taking every state in S and augmenting it with every possible infinite-length vector with elements in $[0, 1]$. We set $A' = A$ and $r' = r$. T' is defined by the following procedure. Given some state $(s_t, p_t) \in S'$ and an action a_t chosen by the agent, the next state is determined by $(s_{t+1}, p_{t+1}) = (g(s_t, a_t, p_t^{0:k}), p_t^{k:\infty})$. Since g is a deterministic function, all transitions in M' are consequently deterministic. The only aspect of M' that remains random is the choice of initial state: the original portion has the same initial distribution as in M, and each element of the infinite vector is distributed $U(0, 1)$. The overall effect of the transformation is to "move" all randomness from the transitions into the selection of the initial state.

The point of all this is that it can be easier to optimize θ for M' than for M. Consider the following algorithm. For some positive integer N, sample N copies of the initial state of M', $(s_1, \ldots, s_N)$. Given an initial state and a deterministic policy, the transitions in M' are completely determined. Thus we can define the function τ' mapping from elements of S' and parameter values θ to trajectories. The sampled initial states now define a deterministic function from policy parameters θ to an estimate of the return of π_θ in M':

$$\widehat{J}_{M'}(\theta) = \frac{1}{N} \sum_{i=1}^{N} R(\tau'(s^i, \theta))$$

The weak law of large numbers dictates that $\widehat{J}_{M'}(\theta)$ is an unbiased estimate of $J_{M'}(\theta) = \int_{S'} d_0'(s) R(\tau(s, \theta)) ds$. We can maximize this deterministic function using any off-the-shelf optimization algorithm, such as gradient ascent with numerically computed derivatives. The main drawback of PEGASUS is the restriction that g be known. Many systems, in particular most physically realized systems, will not meet these requirements, and therefore its applicability is limited.

9 Finite Difference Methods

The problem of optimizing stochastic functions was first studied in the stochastic simulation community, beginning at least as far back as 1951 [25]. One product of these efforts is a class of algorithms known as finite-difference methods. These algorithms are similar to the popular finite-difference methods for numerically approximating gradients of deterministic functions, and are quite separate from the likelihood-ratio-based techniques we have seen so far. One advantage of most finite-difference algorithms is that they treat the stochastic function to be optimized as a black box. For example, they require only the return $R(\tau)$ from a sampled trajectory τ, and do not need knowledge of the states and actions traversed by the trajectory.

Similar to the likelihood-ratio-based techniques, finite-difference methods operate by following the gradient of the objective function. They differ in their methods for estimating the gradient. In general, in order to approximate $\frac{\partial J(\theta')}{\partial \theta}$, finite-difference methods generate some number of perturbations of θ', obtain an estimate of J at each perturbation by sampling and averaging, and fit a hyperplane to the data obtained in this way. This constitutes a local (because the perturbations are small) linear approximation of J [2],

which is exactly the definition of the gradient.

The original finite difference algorithm, known as the Kiefer-Wolfowitz (KW) method generates $2|\theta|$ perturbations for each gradient computation, one perturbation in both positive and negative directions along each coordinate axis (see Kiefer and Wolfowitz [26] for the 1-D case, Blum [27] for the N-D case). The gradient can then be estimated as:

$$\frac{\partial J(\theta)}{\partial \theta} \approx \frac{1}{2c}\left[\tilde{J}(\theta' + ce_1) - \tilde{J}(\theta' - ce_1),\ \ldots,\ \tilde{J}(\theta' + ce_{|\theta|}) - \tilde{J}(\theta' - ce_{|\theta|})\right]^T$$

where $\tilde{J}(\theta)$ is the empirically estimated expected reward when acting according to π_θ, c is a small positive constant, and e_i is the unit vector in the direction of the i-th coordinate axis.

KW can be inefficient because of the large number of sample trajectories required for each gradient computation (at least $2|\theta|$). Simultaneous Perturbation for Stochastic Approximation (SPSA) [28] is an algorithm which improves upon KW by generating just 2 perturbations and 2 samples trajectories per gradient evaluation. The perturbations are chosen randomly each iteration, with each dimension of the perturbation chosen from a symmetric Bernoulli distribution. This algorithm is named for the fact that it simultaneously changes all dimensions of θ to obtain perturbations, instead of one at a time as in KW. Under certain regularity conditions, SPSA can be rigorously shown to achieve better performance than KW for the same total number of sample trajectories.

9.1 Policy Gradient with Parameter-based Exploration

A relatively recent algorithm called Policy Gradient with Parameter-based Exploration (PGPE) [29] can be seen as both extending the SPSA algorithm and connecting finite-difference algorithms with likelihood-ratio methods. While not precisely a finite-difference algorithm, PGPE can be derived from SPSA in four simple steps.

The first step is to change our interpretation of SPSA's perturbation procedure. Suppose we have some candidate parameter value θ. Instead of regarding π_θ as our policy, and SPSA as perturbing that policy to find a gradient, we can regard the perturbation process as *part of* the policy, resulting in a new policy that is a *mixture* of each of the policies obtainable by perturbing θ . When we use this mixture policy to sample a trajectory, we first generate a perturbation $\theta + \Delta\theta$ and then follow $\pi_{\theta+\Delta\theta}$ for the entire trajectory. We can view the mixture policy as being parameterized by the value θ from which perturbations are generated. Our goal is now to find the optimal parameters for this mixture policy.

The second step is to change the way in which perturbations are generated, using a normal distribution centered on θ instead of a symmetric Bernoulli distribution as in SPSA. The third step is to add additional parameters to the parameterization of the mixture policy. In particular, we add parameters controlling the variance of the normal distributions that generate perturbations. This gives the algorithm control over exploration; the algorithm can increase the variance when it needs to explore more, and vice versa.

The final step is to change the way that the gradient is calculated. Now that we have switched to treating the perturbations as part of a mixture policy and included the variance of the perturbations as parameters, it is no longer clear that the finite-difference method would be computing the gradient of the objective with respect to the parameters of the mixture. Consequently, the authors of PGPE use a likelihood-ratio based method for calculating the gradient.

PGPE usually requires policy classes which have the property that the policies are deterministic other than the random selection of parameters at the beginning of the trajectory. This gives PGPE an important advantage over more standard likelihood-ratio based algorithms like REINFORCE. For REINFORCE, we typically optimize over stochastic policies, which is important in order to provide sufficient exploration. However, that stochasticity is also largely responsible for the variance in the gradient estimates, as random draws are taken on every time step. In contrast, PGPE policies make just a single random draw at the beginning of the trajectory. This can significantly reduce the variance in the gradient estimates while

retaining the ability to explore. This advantage is reflected in the experimental results in Sehnke et al. [29] where PGPE is shown outperforming both REINFORCE and episodic Natural Actor-Critic on two difficult example tasks.

10 Recent Advances

Many significant advances have been made in recent years based on the foundational work described above. In large part, these algorithms still operate by searching directly in parameter space for good policies, but in most cases have developed more sophisticated ways of improving the current policy than following a gradient.

The Relative Entropy Policy Search algorithm [30] improves upon natural gradient algorithms by being more explicit about the central idea behind natural gradient, which is to find the perturbation $\Delta\theta$ of the current parameter θ which gives the largest increase in reward while changing the distribution over trajectories the least. Instead of formulating the problem using gradient techniques, which are only valid locally, Peters et al. [30] formulate their problem as an explicit optimization problem, attempting to maximize reward while restricting the KL divergence between the distributions induced by the old policy and the new one to some fixed upper limit. This allows them to potentially take larger steps away from the current policy than natural gradient, which lets them utilize the off-policy sample re-use techniques discussed in Section 7. (Note that they bound the KL-divergence between the steady-state edge traversal distributions, which is what Glynn gives his lower bound on the variance in terms of).

Guided Policy Search [31] uses off-policy techniques to deal with the problem of exploration. They first use a well-studied reinforcement learning technique called Differential Dynamic Programming (DDP) to obtain a time-dependent policy (with a parameterization that cannot be chosen by the user). They then obtain sample trajectories from this policy, which they use to initialize an off-policy policy gradient algorithm that has many similarities to the one used in Tang and Abbeel [21], which we discussed in Section 7. These high-quality initial sample trajectories serve to guide the policy search algorithm to regions of high-reward. The authors provide empirical demonstrations showing that this guidance significantly improves learning speeds. The only drawback is that DDP requires a model of the environment.

Expectation-maximization (EM) [32] is a popular statistical inference algorithm, solving the problem of finding a parameter vector θ which maximizes the log-likelihood $\log P(X|\theta)$ of some observed data X. Several recent algorithms have turned to EM-based techniques to perform policy search. Such algorithms transform the reinforcement learning problem into an *inference problem* of maximizing the log-likelihood $\log P(\mathcal{O}|\theta)$ with respect to θ, where $\mathcal{O}$ is binary a "optimality" random variable that depends on the trajectory. Usually, we define $P(\mathcal{O} = 1|\tau) \propto e^{R(\tau)}$ so that optimality is much more likely for high-reward trajectories. We first assume we have observed $\mathcal{O} = 1$, and then using standard variational techniques, we decompose $\log P(\mathcal{O}|\theta)$ as:

$$\log P(\mathcal{O}|\theta) = \int q(\tau) \log \frac{P(\mathcal{O}, \tau|\theta)}{q(\tau)} d\tau - \int q(\tau) \frac{P(\tau|\mathcal{O}, \theta)}{q(\tau)} d\tau = \mathcal{L}(q, \theta) + KL(q||P(\cdot|\mathcal{O}, \theta))$$

for some variational distribution over trajectories q. We then alternate between minimizing the KL divergence term with respect to q and maximizing $\mathcal{L}$ with respect to θ, both of which cause $\mathcal{L}$, a lower bound on $\log P(\mathcal{O}|\theta)$, to increase [32]. Usually one or both of these steps will only be approximately optimized, and different algorithms implement different ways of performing each optimization, most of which are sampled-based [33, 34, 35].

11 Conclusion

Throughout this study we have looked at a wide range of policy gradient algorithms for finding high quality policies in high-dimensional, continuous environments. The REINFORCE algorithm showed us how to

approximate the gradient from samples in finite-horizon environments, and a related technique based on the Policy Gradient Theorem allowed us to derive a policy gradient actor-critic algorithm suitable for infinite-horizon problems. Next, we reformulated the notion of gradient to give us an alternate definition of policy gradient which always gives the same direction in policy space regardless of how the policy is parameterized, and saw how to derive algorithms based on this alternate gradient definition. We then summarized progress on off-policy policy gradient techniques, which use importance sampling to yield algorithms that make better use of each sample trajectory, and are thus more sample-efficient. The following section looked at approximating the policy gradient when optimizing over classes of deterministic policies, and showed that the gradient can be estimated using fewer samples in such cases because integrating over actions is not required. We then moved on to finite-difference policy gradient algorithms, which are inspired by a technique for numerically approximating gradients of deterministic functions. Finally, we briefly summarized state-of-the-art policy gradient techniques that have progressed beyond the earlier foundational work.

References

[1] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998. ISBN 0262193981.

[2] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *Intelligent Robots and Systems, 2006. (IROS). IEEE/RSJ International Conference on*. IEEE, 2006.

[3] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[4] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[5] Theodore J Perkins and Doina Precup. A convergent form of approximate policy iteration. In *Advances in Neural Information Processing Systems (NIPS)*, 2002.

[6] I H Witten. An adaptive optimal controller for discrete-time markov environments. *Information and Control*, 34:286–295, 1977.

[7] Richard S Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, 1984.

[8] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13, 1999.

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

[10] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13, 1999.

[11] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008.

[12] Sham Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems (NIPS)*, volume 14, 2001.

[13] J Andrew Bagnell and Jeff Schneider. Covariant policy search. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[14] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[15] Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191. American Mathematical Society, 2007.

[16] Shalabh Bhatnagar, Richard S Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

[17] Surya T Tokdar and Robert E Kass. Importance sampling: a review. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):54–60, 2010.

[18] A Rupam Mahmood, Hado P van Hasselt, and Richard S Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[19] Leonid Peshkin and Christian R Shelton. Learning from scarce experience. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, 2002.

[20] Peter W Glynn. Likelilood ratio gradient estimation: an overview. In *Proceedings of the 19th Winter Simulation Conference (WSC)*. ACM, 1987.

[21] Jie Tang and Pieter Abbeel. On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[22] Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

[23] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2014.

[24] Andrew Y Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*. Morgan Kaufmann Publishers Inc., 2000.

[25] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[26] Jack Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[27] Julius R Blum. Multidimensional stochastic approximation methods. *The Annals of Mathematical Statistics*, pages 737–744, 1954.

[28] James C Spall. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *Aerospace and Electronic Systems, IEEE Transactions on*, 34(3):817–823, 1998.

[29] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[30] Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In *Proceedings of 24th AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[31] Sergey Levine and Vladlen Koltun. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[32] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[33] Jens Kober and Jan Peters. Learning motor primitives for robotics. In *Robotics and Automation, 2009. (ICRA). IEEE International Conference on*. IEEE, 2009.

[34] Gerhard Neumann. Variational inference for policy search in changing situations. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011.

[35] Sergey Levine and Vladlen Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.