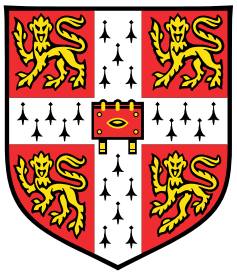
Improving Sample Efficiency for Gradient-based Policy Optimisation; with an Application to Structured Policy Functions



Sihui Wang

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning, Speech and Language Technology

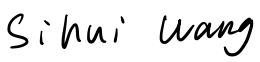
Hughes Hall 11 August 2017

Declaration

I, Sihui Wang of Hughes Hall, being a candidate for the M.Phil in Machine Learning, Speech and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

This dissertation contains a total word count of 11,009.

Sihui Wang
11 August 2017



Acknowledgements

I would like to express my earnest gratitude to my industrial supervisor, Dr Tom Gunter, for his insights and creativity in proposing this thought-provoking MPhil project which is real fun to work on. His unwavering support and passion have lead my way through the tough mental challenges of reinforcement learning, while his wisdom and charisma consistently serve as my source of inspiration. The discussions have always been fruitful and enlightening. I enjoy all the life lessons that he gave me as a supervisor, or rather as a friend beyond the pure academics.

I would also like to thank my academic supervisor, Professor Bill Byrne, for his care and support along the adventure of this thesis. The magical power of his soothing voice never fails to assure me in times of anxiety and self-doubt.

Abstract

In on-policy reinforcement learning, one popular class of approaches for inferring a strategy involves attempting to directly optimize the policy function, targeting improvement with respect to some long term notion of reward. However such methods are typically sample inefficient due to their on-policy nature—sufficient samples must be collected to compute all required estimates for each parameter update of the policy function. In this thesis, we will investigate existing work and introduce new methods to improve the sample efficiency for a state of the art policy gradient reinforcement learning method, Trust Region Policy Optimization (TRPO)[Schulman et al., 2015a].

One effective way to improve sample efficiency is to incorporate more prior knowledge into the policy function. We were motivated to consider incorporating hierarchical structure, as many real world planning problems are hierarchical in nature. However, if this exact nature of the hierarchical structure is a-priori unknown, then the variance of our gradient estimators will increase as compared to a naïve flat policy (although the performance of the final solution is expected to be lower bounded by that of the flat policy). Motivated by this, we come up with two variance reduction methods in Chapter 3. The first one uses an non-parametric model to give a baseline performance in terms of the estimated value of the state. And the second method uses proximity information in policy space to interpolate gradients from previous batches. Both methods are tested on a few tasks in OpenAI Gym[Brockman et al., 2016] and show promising results, namely achieving both faster convergence rate and better end performance. Finally we present a structured version of TRPO by incorporating a hierarchy, making use of the variance reduction technique presented in Chapter 3 to facilitate learning.

Table of contents

Chapter 1

Introduction

1.1 Motivation

Learning from interaction is a part of human nature that is deeply embedded in our genes. Be it a toddler, an adolescent or an adult, learning through experience to improve performance over time is an activity carried out subconsciously everyday. Kids learn to walk, run and ride bicycles by trial-and-error without being told how much force to exert or what angle the body should lean forward. They avoid the bad actions which lead to falling down while reinforcing the successful actions when rewards are experienced. Behaviorism defines learning as the acquisition of new behavior based on environmental conditions. Inspired by this kind of behaviorist psychology, the field of Reinforcement Learning (RL) emerges as a framework for solving sequential decision-making problems through interaction with the environment. [Sutton and Barto, 1998] provides an excellent introduction and overview on RL.

The applications of RL are ubiquitous, for example, robotics control, spoken dialogue systems, game playing and etc[Jurčíček et al., 2012; Kober and Peters, 2012; Mnih et al., 2013, 2015; Silver et al., 2016; Singh et al., 2000]. Among all the applications, there are two recent major breakthroughs. One being DeepMind's AlphaGo [Silver et al., 2016] beating a top ranked human Go master, which was expected to happen only after another decade. A second success story centers around the use of computer games as learning environments, namely the use of Deep RL to achieve human and above-human performances on a wide range of Atari games [Mnih et al., 2015], without any prior knowledge of the game itself. The successes above demonstrate RL is starting to show real promise as a powerful paradigm to solve complex sequential decision-making problems. Along with this recent renaissance of RL, there is a new toolkit called OpenAI Gym[Brockman et al., 2016]

which provides benchmarking problems to aid rapid prototyping and testing of reinforcement learning algorithms.

1.2 Contribution of the Thesis

Despite the latest progress within the RL community, most online RL algorithms are data inefficient when solving complex problems. One direction for improving sample efficiency is to incorporate prior knowledge into the policy function, we consider the addition of hierarchical structure: namely splitting one large complicated task into a gating mechanism over smaller sub-tasks. In doing so, we hope to find the optimal policy function in as few samples as possible (assuming we have strong knowledge about the form of the hierarchy a priori). Even in the cases where we do not have *strong* knowledge as to the exact nature of the hierarchy, the abstract prior implying that such a hierarchy exists should be sufficient to improve sample efficiency as compared to a naïve, very high capacity policy function. Motivated by this, we are interested in incorporating some form of hierarchical structure on top of a foundational, state of the art policy gradient RL method, Trust Region Policy Optimization (TRPO)[Schulman et al., 2015a].

In this thesis, we will be first give a detailed review of different reinforcement learning algorithms and the reasons behind why we decided to work on particular gradient-based policy optimization methods. Next, we explore several different ways to reduce the variance of gradient estimates in gradient-based policy optimization method which leads to better sample efficiency. We propose two new methods in achieving that. We will finally discuss how to incorporate a hierarchical structure into TRPO and combine it with the previous variance reduction method.

1.3 Thesis Outline

In Chapter 2, we will first give an overview on reinforcement learning as well as detailed reasoning behind why we chose to pursue an investigation of gradient-based policy optimization methods. In Chapter 3, we will present two new methods for variance reduction of gradient estimators by making use of previously sampled data. Though applied to TRPO specifically, they can be easily extended to other gradient-based policy optimization methods. In Chapter 4, we will extend TRPO to incorporate hierarchical structure, making use of the variance reduction methods introduced in Chapter 3. Finally we conclude the thesis by discussing potential future work.

Chapter 2

Reinforcement Learning Background

2.1 Introduction to Reinforcement Learning

2.1.1 Introduction

In reinforcement learning, a goal-directed agent tries to maximize its total long term (discounted) reward by interacting with an uncertain environment, aiming to improve its performance by learning from past experience. There are several important abstract entities in a RL setup: states, actions and rewards. The state, S, encodes our environment; it is an encapsulation of the current state of the world, as viewed by the agent. The action, A, is the subsequent decision made by the agent conditioning on the current state as well as other side information. After an action has been selected, a reward, R, is presented as an immediate feedback from our environment. Typically the action chosen will also induce an update to the state of the world, S. The reward received indicates how good the previous action is. The following figure is illustrative of the interaction between agent and the environment.

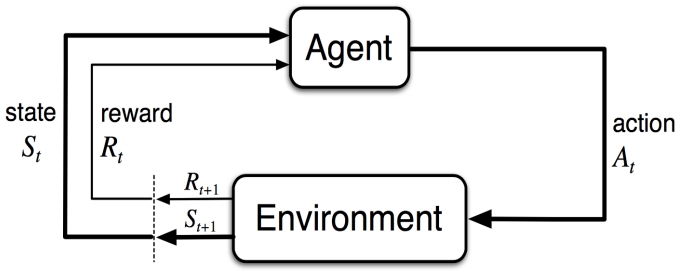


Fig. 2.1 [Sutton and Barto, 1998] At time t, the agent chooses action A_t based on the current state S_t, the environment gives a reward R_{t+1} and transfers the agent to the next state S_{t+1}, the process continues.

A standard assumption in the study of RL is that the state satisfies first order Markov property, namely the future does not depend on the past but only the present. This simply means the state at time t+1 only depends on the state and action at time t. The RL task is called a Markov decision process (MDP) if the Markov property is satisfied. This may not be true in real life, and relaxations of this are possible, however it greatly simplifies theoretical analysis, and has been demonstrated to be capable of solving non-Markovian task empirically. As described in [Sutton and Barto, 1998], some early studies managed to solve the pole-balancing task easily by using a coarse state signal (a rough discretization of state space, such as dividing the cart position into three regions: right, left and middle) which gives rise to a non-Markovian state. When considering an infinite horizon MDP, it is necessary to discount the reward by γ, $0 \leq \gamma \leq 1$, this is in accordance with the fact that future reward is worth less than the immediate reward, at the same time it ensures the expected return will always be finite.

2.1.2 Model-based vs Model-free Reinforcement Learning

There are two main techniques to tackle a RL problem, model based and model-free methods. [Dayan and Niv, 2008] provides an intuitive example distinguishing the two.

In model-based learning, a model of the environment is included to capture the environment's dynamics. More specifically we attempt to model the state transition probability, $P(S_{t+1} = s'|S_t = s, A_t = a)$ and the reward function $R(S_t = s, A_t = a)$. Planning is then carried out to choose appropriate actions according to the learned environment model. The advantage of model-based RL is its highly efficient use of data, hence less simulation is required to solve a problem. For example, PILCO[Deisenroth and Rasmussen, 2011] achieves great data efficiency in some continuous state-action tasks, solving Cart-Pole problem in only 17.5 seconds when interacting with the real physical system. Nevertheless, the effectiveness of model-based algorithms is highly dependent on and limited by their ability to sufficiently resemble the real environment. For tasks governed by the laws of physics, model-based methods usually work incredibly well. However for other tasks like dialogue systems, or playing Go, it is difficult to come up with a suitable model to encode the transition dynamics of the complex environment. In most cases, it is in fact unnecessary to understand the rules of the environment in order to solve a task. Hence the use of model-free RL methods is more commonly found in literature. Model-free algorithms learn a good behavior directly based on some trial-and-error experience. We make no assumptions about the dynamics of the environment; it is much easier to set up and more generalizable as c.f. model based RL, but

typically requires more interaction with the environment. We will focus solely on model-free RL in this thesis.

2.1.3 Value and Policy-based Optimization

There are two further abstraction choices made when formulating a RL setup: the policy and value functions. The policy π is defined to be a mapping from state to action if deterministic, $\pi : S \rightarrow A$, or a map onto a distribution over actions if stochastic. In the later case, each action, $a \in A$, has a probability $\pi(a|s)$ of being selected, for example, a Boltzmann softmax policy[Sutton and Barto, 1998].

The value of a state under policy π is the total (discounted) reward an agent can expect to accumulate over the future, starting from that state and following policy π, $V_\pi(s)$. This is defined as follows:

$$V_\pi(s) = \mathbb{E}_\pi\Big[\sum_{l=0}^{\infty} \gamma^l R_{t+l} | S_t = s\Big]$$

Similarly, the state-action value function under a policy π is the expected cumulative reward given that the agent took action, a, at state s:

$$Q_\pi(s,a) = \mathbb{E}_\pi\Big[\sum_{l=0}^{\infty} \gamma^l R_{t+l} | S_t = s, A_t = a\Big],$$

The rewards signal, R_t is given directly by the environment but those V and Q values can only be estimated and re-estimated from experience.

Value-based Optimization

There are three common ways of estimating these value functions: dynamic programming, Monte Carlo (MC) approximation and temporal-difference (TD) methods[Sutton, 1988]. We will not discuss dynamic programming as it requires an accurate representation of the environment which falls under model-based RL.

In comparison, MC methods require only an ability to generate uncorrelated samples from the system-environment setup: it uses the average of sample returns from the environment to approximate the expected values. The idea is with infinite number of samples (to each state or state-action pair), the empirical average will converge to the expected value by the law of large numbers. The update of parameters of value function can only be performed at the end

of an episode.

TD method on the other hand is able to update after every time step. The simplest TD(0) method uses the following update rule:

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha\Big[r(s_t, \pi(s_t)) + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)\Big] \tag{2.1}$$

where α is a constant step-size parameter and $r(s_t, \pi(s_t))$ is the reward obtained by following policy π at s_t.

In a model-free setting, we need to estimate the state-action value function, Q(s,a), in order to find better policy. There are typically two stages involved in a value-based optimization method. First, we estimate the value function under a particular policy π, this is known as the policy evaluation stage. Next, we improve the policy by making it greedy (or epsilon-greedy) with respect to the current value function. This is known as policy improvement stage. No explicit policy function model is needed in this case, it is generated indirectly from the value function. When the state and action spaces are small, it is possible to use tabular methods to store the values for each state-action pair as one entry. For example in a 10×10 grid-world task, we can easily explore all possible state-action pairs. However, if there are too many states or if we are dealing with continuous state and action spaces, it is impractical to store the state-action value in a look-up table. Instead, a parameterized function approximator such as a linear model or neural net, is used to estimate the corresponding values and their corresponding parameters are updated to better match the observed returns using gradient method.

Policy-based Optimization

Alternatively, it is possible to learn a policy directly without the use of value function. In some cases, the value function could be more difficult to estimate than learning a parametrized policy directly. For example, in the game of Tetris, it might be much easier to decide where to put a certain piece than estimate the expected infinite horizon future reward resulting from that placement; the policy space might have a more compact representation. It is also suggested in [Gabillon et al., 2013] that good policies are easier to represent and learn than the corresponding value functions in the game of Tetris. Moreover, since policies are determined indirectly via value function approximation, inappropriate models of value function can lead to inferior policies even for simple tasks[Weaver and Baxter, 1999]. In addition, for high-dimensional

or continuous action spaces, even if we manage to learn the action-state value function accurately, it is difficult to take the argmax over all actions and obtain the greedily optimal one; in such cases policy-based methods will clearly be more effective. One theoretical flaw regarding value-based parameterized function approximation is an inability to guarantee convergence apart from the linear parametrization case. Whereas if we use gradient-based policy optimization method, we are guaranteed to converge to at least a local optimum as long as the parameterized policy is continuously differentiable[Bertsekas and Tsitsiklis, 2000]. Furthermore, a value-based optimization method is oriented towards learning a deterministic policy which could fail in a stochastic environment. For example, any deterministic policy can be easily exploited by your opponent in the game of rock–paper–scissors. In certain circumstances, stochastic policies might be able to provide a better user experience: for example in dialogue systems, people might prefer to hear different responses with the same input/question. Even in the case where the true optimal policy for a task is indeed deterministic, having a stochastic policy helps enable better exploration of the state-action space, thereby allowing the agent to learn a richer set of behavior—which is why DDPG[Silver et al., 2014] (and much other prior art in the RL community) use a stochastic behavior policy. [Schulman and Abbeel, 2016] provides an excellent detailed review of a number of different policy optimization methods.

In such a policy-based optimization algorithm, we have a policy function, $\pi_\theta(a|s)$ parameterized by θ. This could be a neural network where the parameters being the weights and bias. This formulation reduces a complex RL setup to the following numerical optimization problem, where we would like to maximize the expected (discounted) cumulated reward with respect to θ.

$$\max_\theta \mathbb{E}[R|\pi_\theta]$$

To solve the above numerical optimization, we generally turn to one of the two main categories of policy-based optimization methods: gradient-free optimization method and gradient-based optimization method.

Gradient-free policy optimization method can be considered as a black-box optimization method. It perturbs (adds noise to) the policy parameters, subsequently selecting the parameters which result in better performance. Some of the most popular gradient-free methods include Cross-Entropy Method (CEM)[De Boer et al., 2005], Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)[Hansen, 2006], and Policy Learning by Weighting Exploration with the Returns (PoWER)[Kober and Peters, 2009]. Such black-box methods usually work remarkably well when the number of parameters is small. However those

evolutionary strategies ignore much of the useful structure within RL problems, they discard all the temporal information in the process and care only about the final performance. They are typically unable to scale well on high-dimensional space where there are large number of parameters, which is the case for neural network based policy functions.

Another family of policy-based optimization methods is Policy Gradient Reinforcement Learning (PGRL). The idea is to simply perform gradient ascend on the parameters in the direction which improves the expected future rewards. The advantage of using a gradient-based optimization, as opposed to gradient-free, is that it is less vulnerable to noise (poor quality reward information) as long as we move a small incremental step each time along the gradient direction. It is widely used for continuous action space RL problems and has received some recent attention as a result of its several breakthroughs in robotics locomotion, playing Atari games as well as playing Go[Schulman et al., 2017; Schulman et al., 2015a,b; Silver et al., 2016]. There have been attempts to incorporate parameter perturbation from gradient-free methods into gradient based optimization method, resulting in a more efficient learning than using each alone[Plappert et al., 2017].

2.2 Policy Gradient Reinforcement Learning

2.2.1 Vanilla Policy Gradient

Consider an infinite-horizon MDP discounted by γ. The initial state of an agent, s_0 is randomly sampled according to an initial state distribution $\rho_0(s)$. At time step t, the agent chooses its action a_t according to a policy $\pi_\theta(a_t|s_t)$ and receives a reward $r(s_t,a_t)$. As a result of executing this state-action pair, the agent is now brought to a new state s_{t+1} according to the (unseen) transition probability $P(s_{t+1}|s_t,a_t)$ and the process continues. If we let τ denotes a trajectory of state-action tuples, $\{(s_0,a_0),(s_1,a_1),...\}$, and denote the probability of obtaining the sequence τ under policy parameters θ by $P(\tau|\theta)$. The total reward received under the sequence τ is given by $R(\tau)$. The expected sum of rewards of a particular policy parameterized by θ is given by

$$\eta(\pi_\theta) = \mathbb{E}_\tau\Big[R(\tau)\Big]$$

The gradient of $\eta(\pi_\theta)$ w.r.t θ can be expressed as follows using the likelihood ratio trick.

$$\begin{aligned}
\nabla_\theta \eta(\pi_\theta) &= \nabla_\theta \int P(\tau|\theta) R(\tau) \mathrm{d}\tau \\
&= \int \frac{P(\tau|\theta)}{P(\tau|\theta)} \nabla_\theta P(\tau|\theta) R(\tau) \mathrm{d}\tau \\
&= \int P(\tau|\theta) \frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta)} R(\tau) \mathrm{d}\tau \\
&= \int P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) R(\tau) \mathrm{d}\tau \\
&= \mathbb{E}_\tau \Big[\nabla_\theta \log P(\tau|\theta) R(\tau) \Big]
\end{aligned} \tag{2.2}$$

The intuition behind such likelihood ratio gradient is that the agent should try to increase the probability of executing sequence τ which leads to positive reward and decrease the probability of τ with negative reward. Although a MDP assumes an infinite number of time steps, this is difficult to work with in practice. We will therefore restrict ourselves to episodic task and limit the maximum number of time steps in each episode to be T. The gradient of log-probability in obtaining sequence τ can be further decomposed into the following form:

$$\begin{aligned}
\nabla_\theta \log P(\tau|\theta) &= \nabla_\theta \log \Big[\underbrace{\rho_0(s_0)}_{s_0 \text{ distribution}} \times \prod_{t=0}^{T-1} \underbrace{P(s_{t+1}|s_t, a_t)}_{\text{dynamics model}} \times \underbrace{\pi_\theta(a_t|s_t)}_{\text{policy}} \Big] \\
&= \nabla_\theta \Big[\sum_{t=0}^{T-1} \log P(s_{t+1}|s_t, a_t) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) \Big] \\
&= \nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) \\
&= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)
\end{aligned} \tag{2.3}$$

It turns out we do not even need the model of the environment in order to perform such gradient estimation. In practice, we use Monte Carlo approximation to estimate the gradient by averaging over different time steps across different trajectories and this results in a well-known gradient-based algorithm called the REINFORCE algorithm [Williams, 1992]. This method is easy to understand and implement, however it tends to be unstable as small changes in the parameters tend to result in big unexpected changes in the policy space. It is difficult to choose an appropriate step size which would remain suitable throughout the whole learning process. The convergence will be extremely slow if the step size is too small, on the other hand the performance will be degraded severely by noise for a large step size.

Another drawback of vanilla gradient method is the score function might be numerically unstable, tending to infinity under certain conditions. For example, consider a Gaussian policy with $a \sim \mathcal{N}(\mu(s), \sigma^2)$, where $\mu(s) = \phi(s)^T\theta$. We have the score function expressed in the following form:

$$\nabla_\theta \log \pi_\theta(a|s) = \frac{(a-\mu(s))\phi(s)}{\sigma^2}$$

This implies as the noise of our policy tends to zero, the gradient estimator will tend to infinity, giving an unusable gradient estimator. Deterministic Policy Gradient Algorithms [Silver et al., 2014] have been suggested to tackle this limit problem, and they do so to good effect. They are however off-policy, relying on data collected using a random policy for exploration.

Vanilla policy gradient estimators are also not invariant to simple re-parametrizations. For example, if our policy has the following softmax form:

$$\pi_\theta(a|s) = \frac{\exp(\phi(s,a)^T\theta)}{\sum_{a'}\exp(\phi(s,a')^T\theta)}$$

then the score function, $\nabla_\theta \log \pi_\theta(a|s)$, has the following analytical form,

$$\nabla_\theta \log \pi_\theta(a|s) = \nabla_\theta[\phi(s,a)^T\theta - \sum_{a'}\phi(s,a')^T\theta] = \phi(s,a) - \sum_{a'}\phi(s,a')$$

If we re-parameterize our policy, for example, replace $\phi(s,a)$ with $\phi(s,a)+b$, although the softmax probability remains unchanged, the gradient estimator will be different. If, however, we use an alternative gradient direction, namely the natural gradient, the gradient estimator will be robust to such re-parameterization[Martens, 2014]. Such adaptations to REINFORCE algorithm bring us to the natural gradient algorithm[Kakade, 2002], and TRPO[Schulman et al., 2015a].

2.2.2 Trust Region Policy Optimization

The expected cumulative reward for an infinite time horizon can be rewritten in the following form:

$$\eta(\pi) = \mathbb{E}_{s_0,a_0,\ldots}\Big[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\Big]$$

We introduce an advantage function, defined by

$$A_\pi(s_t,a_t) = Q_\pi(s_t,a_t) - V_\pi(s_t)$$

we can think of $A_\pi(s_t, a_t)$ as the expected extra reward obtained by taking the best action in a certain state, given the agent is operating with policy π. We can then express the expected return of a policy π_θ in terms of advantage over $\pi_{\theta_{\text{old}}}$ (the proof can be found in [Kakade, 2002; Schulman et al., 2015a]). Intuitively, we can think of the difference between expected reward of two policies as the expected advantage of one policy over the other.

$$\eta(\pi_\theta) = \eta(\pi_{\theta_{\text{old}}}) + \mathbb{E}_{s_0, a_0, \ldots \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t A_{\pi_{\theta_{\text{old}}}}(s_t, a_t)\right] \tag{2.4}$$

Let $\rho_\pi(s)$ be the discounted state visitation frequencies,

$$\rho_\pi(s) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi)$$

where $s_0 \sim \rho_0$, $a_t \sim \pi(a_t|s_t)$ and $s_t \sim P(s_{t+1}|s_t, a_t)$. Then equation 2.4 can be expressed as sum over states and actions instead of time steps,

$$\begin{aligned}
\eta(\pi_\theta) &= \eta(\pi_{\theta_{\text{old}}}) + \sum_{t=0}^{\infty} \sum_s P(s_t = s|\pi_\theta) \sum_a \pi_\theta(a|s) \gamma^t A_{\pi_{\theta_{\text{old}}}}(s, a) \\
&= \eta(\pi_{\theta_{\text{old}}}) + \sum_s \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi_\theta) \sum_a \pi_\theta(a|s) A_{\pi_{\theta_{\text{old}}}}(s, a) \\
&= \eta(\pi_{\theta_{\text{old}}}) + \sum_s \rho_{\pi_\theta}(s) \sum_a \pi_\theta(a|s) A_{\pi_{\theta_{\text{old}}}}(s, a)
\end{aligned} \tag{2.5}$$

So the task now can be reduced to maximizing $\sum_s \rho_{\pi_\theta}(s) \sum_a \pi_\theta(a|s) A_{\pi_{\theta_{\text{old}}}}(s, a)$ with respect to θ, however this is difficult to optimize directly as the discounted visitation frequency, $\rho_{\pi_\theta}(s)$ depends on policy π_θ. Instead, we can introduce a local approximation in the following form:

$$L(\pi_\theta) = \eta(\pi_{\theta_{\text{old}}}) + \sum_s \rho_{\pi_{\theta_{\text{old}}}}(s) \sum_a \pi_\theta(a|s) A_{\pi_{\theta_{\text{old}}}}(s, a) \tag{2.6}$$

If our parameterized policy $\pi_\theta(a|s)$ is first order differentiable w.r.t θ, we can show that these two forms are the same up to first order, namely $\nabla_\theta L(\pi_\theta)|_{\theta=\theta_{old}} = \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_{old}}$. The proof is given as follows:

$$\nabla_\theta L(\pi_\theta)|_{\theta=\theta_{\text{old}}} = \sum_{s,a} \rho_{\pi_{\theta_{\text{old}}}}(s) A_{\pi_{\theta_{\text{old}}}}(s, a) \nabla_\theta \pi_\theta(a|s)|_{\theta=\theta_{\text{old}}} \tag{2.7}$$

$$
\begin{aligned}
\nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_{\text{old}}} &= \sum_{s,a} \rho_{\pi_{\theta_{\text{old}}}}(s) A_{\pi_{\theta_{\text{old}}}}(s,a) \nabla_\theta \pi_\theta(a|s)|_{\theta=\theta_{\text{old}}} \\
&\quad + \sum_{s,a} \Big(\nabla_\theta \rho_{\pi_\theta}(s)|_{\theta=\theta_{\text{old}}} \Big) \pi_{\theta_{\text{old}}}(a|s) A_{\pi_{\theta_{\text{old}}}}(s,a) \\
&= \nabla_\theta L(\pi_\theta)|_{\theta=\theta_{\text{old}}} + \sum_s \nabla_\theta \rho_{\pi_\theta}(s)|_{\theta=\theta_{\text{old}}} \Big[\sum_a \pi_{\theta_{\text{old}}}(a|s) \big(Q_{\pi_{\theta_{\text{old}}}}(s,a) - V_{\pi_{\theta_{\text{old}}}}(s) \big) \Big] \\
&= \nabla_\theta L(\pi_\theta)|_{\theta=\theta_{\text{old}}} \\
&\quad + \sum_s \nabla_\theta \rho_{\pi_\theta}(s)|_{\theta=\theta_{\text{old}}} \Big[\sum_a \pi_{\theta_{\text{old}}}(a|s) Q_{\pi_{\theta_{\text{old}}}}(s,a) - V_{\pi_{\theta_{\text{old}}}}(s) \sum_a \pi_{\theta_{\text{old}}}(a|s) \Big] \\
&= \nabla_\theta L(\pi_\theta)|_{\theta=\theta_{\text{old}}} + \sum_s \nabla_\theta \rho_{\pi_\theta}(s)|_{\theta=\theta_{\text{old}}} \Big[V_{\pi_{\theta_{\text{old}}}}(s) - V_{\pi_{\theta_{\text{old}}}}(s) \Big] \\
&= \nabla_\theta L(\pi_\theta)|_{\theta=\theta_{\text{old}}}
\end{aligned}
\tag{2.8}
$$

Hence $\eta(\pi_\theta)$ and $L(\pi_\theta)$ match each other to first order when evaluated at θ_{old}. And by definition we have $L(\pi_{\theta_{\text{old}}}) = \eta(\pi_{\theta_{\text{old}}})$. Therefore, it is easy to see if we make a small change in θ, an improvement in the local approximation, L, will also result in an improvement in η. Now the task reduces to maximizing the following surrogate objective function, replacing the advantage function with an action-state value function which only changes the objective function by a constant. Estimates in the below are then made through an importance sampling mechanism.

$$
\begin{aligned}
&\max_\theta \sum_s \rho_{\pi_{\theta_{\text{old}}}(s)} \sum_a \pi_\theta(a|s) A_{\pi_{\theta_{\text{old}}}}(s,a) \\
\equiv &\max_\theta \sum_s \rho_{\pi_{\theta_{\text{old}}}(s)} \Big(V_{\pi_{\theta old}}(s) + \sum_a \pi_\theta(a|s) A_{\pi_{\theta_{\text{old}}}}(s,a) \Big) \\
\equiv &\max_\theta \sum_s \rho_{\pi_{\theta_{\text{old}}}(s)} \Big(\sum_a \pi_\theta(a|s) V_{\pi_{\theta old}}(s) + \sum_a \pi_\theta(a|s) A_{\pi_{\theta_{\text{old}}}}(s,a) \Big) \\
\equiv &\max_\theta \sum_s \rho_{\pi_{\theta_{\text{old}}}(s)} \sum_a \pi_\theta(a|s) Q_{\pi_{\theta_{\text{old}}}}(s,a) \\
= &\max_\theta \mathbb{E}_{s \sim \rho_{\pi_{\theta_{\text{old}}}}, a \sim \pi_{\theta_{\text{old}}}} \Big[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} Q_{\pi_{\theta_{\text{old}}}}(s,a) \Big]
\end{aligned}
\tag{2.9}
$$

The above surrogate objective function suggests that if our $Q_{\pi_{\theta_{\text{old}}}}(s,a)$ is positive, the update will try to drive $\pi_\theta(a|s)$ to 1 and if the Q-value is negative, it tries to drive $\pi_\theta(a|s)$ to 0. However if we simply follow the Monte Carlo estimation of the gradient of above surrogate objective function, it can result in large changes in the value of θ especially when there are some outliers in our samples. In such case, our surrogate objective function will no longer be a good approximation of η. It would be useful if we can introduce a lower bound of $\eta(\pi_\theta) - L(\pi_\theta)$ in terms of the step size we are taking. Then we can make use of this explicit

lower bound to decide how big of a step to take. In fact, the following is proved in [Schulman et al., 2015a].

Theorem 1. *Let* $D_{KL}^{max}(\pi_{\theta_{old}}, \pi_\theta) = \max_s D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_\theta(\cdot|s))$*, then we have*

$$\eta(\pi_\theta) \geq L(\pi_\theta) - CD_{KL}^{max}(\pi_{\theta_{old}}, \pi_\theta)$$

where $C = \frac{2\varepsilon\gamma}{(1-\gamma)^2}$, $\varepsilon = \max_s \max_a |A_{\pi_{\theta_{old}}}(s,a)|$

Using the above equation, a policy which improves the right hand side is guaranteed to give us improvement on η. This is illustrated as follows.

$$\begin{aligned} \eta(\pi_\theta) &\geq L(\pi_\theta) - CD_{KL}^{max}(\pi_{\theta_{\text{old}}}, \pi_\theta) \\ &\geq L(\pi_{\theta_{\text{old}}}) - CD_{KL}^{max}(\pi_{\theta_{\text{old}}}, \pi_{\theta_{\text{old}}}) \\ &= \eta(\pi_{\theta_{\text{old}}}) \end{aligned} \tag{2.10}$$

Hence we can construct a minorization-maximization algorithm which is guaranteed to give monotonic improvement on η. The intuition behind this is to essentially limit the changes between consecutive policies, therefore a single update cannot make a drastic change in *policy space*, ensuring stability. Now our surrogate objective function has changed to $L(\pi_\theta) - CD_{KL}^{max}(\pi_{\theta_{\text{old}}}, \pi_\theta)$. However in practice, if we use the above suggested penalty coefficient C, this gives us very small step size and therefore lots of samples are required for policy to converge. Empirically, it is difficult to choose an appropriate penalty coefficient. Here is where the idea of using a trust region (borrowed from the optimization literature) comes in, allowing us to take larger steps in a robust way.

Trust region method is commonly used in mathematical optimization. The idea behind it is it's enough to have an approximate model near the current point within a (trust) region. The model is assumed to be accurate to within ε of the ground truth within that 'trust' region. If the approximation is adequate, we can expand the trust region, else we shrink it. A more detailed review on trust region can be found in [Yuan, 2000]. In the context of RL optimization, we use the $L(\pi_\theta)$ as an adequate model to approximate $L(\pi_\theta) - CD_{KL}^{max}(\pi_{\theta_{\text{old}}}, \pi_\theta)$ in the region where $D_{KL}^{max}(\pi_{\theta_{\text{old}}}, \pi_\theta) \leq \delta$, δ is a tunable parameter. The adequacy of the approximation is measured in terms of expected improvement in $L(\pi_\theta) - \mathscr{X}\left[D_{KL}^{max}(\pi_{\theta_{\text{old}}}, \pi_\theta) \leq \delta\right]$, where $\mathscr{X}$ is zero when the constrained inside is met and infinity when it is not. If there is no improvement in $L(\pi_\theta) - \mathscr{X}\left[D_{KL}^{max}(\pi_{\theta_{\text{old}}}, \pi_\theta) \leq \delta\right]$, we reduce the value of δ, namely shrinking the trust region. This gives us an effective way in choosing suitable step size in policy gradient optimization. The unconstrained optimization problem has now changed to

the following constrained optimization,

$$\max_{\theta} L(\pi_\theta) \qquad subject\ to\ D_{KL}^{max}(\pi_{\theta_{\text{old}}}, \pi_\theta) \leq \delta$$

This requires us to bound the KL divergence at every point in the state space which is impractical for large number or continuous state space. In practice, we made another approximation to the above trust region constrained optimization problem, namely replace the max KL by average KL. The average KL is then evaluated using Monte Carlo. Our optimization problem becomes maximizing Equation 2.9 with the average KL trust region constraint, namely

$$\max_{\theta} \mathbb{E}_{s\sim\rho_{\pi_{\theta_{\text{old}}}}, a\sim\pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} Q_{\pi_{\theta_{\text{old}}}}(s,a) \right] \tag{2.11}$$

$$\text{subject to } \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}} \left[D_{KL}(\pi_{\theta_{\text{old}}}(\cdot|s)||\pi_\theta(\cdot|s)) \right] \leq \delta$$

There are two sampling schemes, single path and vine path, for performing Monte Carlo estimation. Single path is typically used in policy gradient estimation for model-free setting where we sample one action from each state. Whereas for vine path, we sample different actions from a given state to form rollout trajectories. A graphical illustration of each is given below in Fig.2.2. The vine path method is able to give lower variance of Q-value at the cost of more simulations. Vine methods requires the system to be restored to a particular state which might not be practical on a real physical system. Hence we will stick with single path sampling method for the remainder of this thesis.

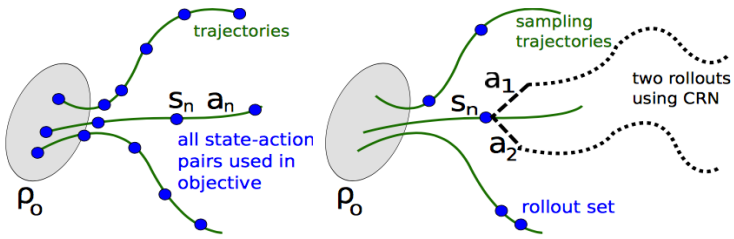


Fig. 2.2 Illustration of single path(left) and vine path(right)[Schulman et al., 2015a]

While solving the constraint optimization problem, we will make a few further approximations, namely a linear approximation to $L(\pi_\theta)$ and quadratic approximation to the average KL term. The approximations can be expressed as follows,

$$L(\pi_\theta) \approx L(\pi_{\theta_{\text{old}}}) + (\theta - \theta_{\text{old}})^T \nabla_\theta L(\pi_\theta)|_{\theta=\theta_{\text{old}}} \tag{2.12}$$

$$\begin{aligned}
\overline{D_{KL}}(\pi_{\theta_{\text{old}}}, \pi_\theta) &\approx \overline{D_{KL}}(\pi_{\theta_{\text{old}}}, \pi_{\theta_{\text{old}}}) + (\theta - \theta_{\text{old}})^T \nabla_\theta \overline{D_{KL}}(\pi_{\theta_{\text{old}}}, \pi_\theta)|_{\theta=\theta_{\text{old}}} \\
&\quad + \frac{1}{2}(\theta - \theta_{\text{old}})^T \nabla_\theta^2 \overline{D_{KL}}(\pi_{\theta_{\text{old}}}, \pi_\theta)|_{\theta=\theta_{\text{old}}}(\theta - \theta_{\text{old}}) \\
&= 0 + (\theta - \theta_{\text{old}})^T \nabla_\theta \sum_s \sum_a \pi_{\theta_{\text{old}}}(a|s) \log \frac{\pi_{\theta_{\text{old}}}(a|s)}{\pi_\theta(a|s)}|_{\theta=\theta_{\text{old}}} \\
&\quad + \frac{1}{2}(\theta - \theta_{\text{old}})^T \nabla_\theta^2 \overline{D_{KL}}(\pi_{\theta_{\text{old}}}, \pi_\theta)|_{\theta=\theta_{\text{old}}}(\theta - \theta_{\text{old}}) \\
&= -(\theta - \theta_{\text{old}})^T \sum_s \sum_a \pi_{\theta_{\text{old}}}(a|s) \nabla_\theta \log \pi_\theta(a|s)|_{\theta=\theta_{\text{old}}} \\
&\quad + \frac{1}{2}(\theta - \theta_{\text{old}})^T \nabla_\theta^2 \overline{D_{KL}}(\pi_{\theta_{\text{old}}}, \pi_\theta)|_{\theta=\theta_{\text{old}}}(\theta - \theta_{\text{old}}) \\
&= -(\theta - \theta_{\text{old}})^T \sum_s \sum_a \nabla_\theta \pi_\theta(a|s)|_{\theta=\theta_{\text{old}}} \\
&\quad + \frac{1}{2}(\theta - \theta_{\text{old}})^T \nabla_\theta^2 \overline{D_{KL}}(\pi_{\theta_{\text{old}}}, \pi_\theta)|_{\theta=\theta_{\text{old}}}(\theta - \theta_{\text{old}}) \\
&= -(\theta - \theta_{\text{old}})^T \sum_s \nabla_\theta \sum_a \pi_\theta(a|s)|_{\theta=\theta_{\text{old}}} \\
&\quad + \frac{1}{2}(\theta - \theta_{\text{old}})^T \nabla_\theta^2 \overline{D_{KL}}(\pi_{\theta_{\text{old}}}, \pi_\theta)|_{\theta=\theta_{\text{old}}}(\theta - \theta_{\text{old}}) \\
&= -(\theta - \theta_{\text{old}})^T \sum_s \nabla_\theta 1 + \frac{1}{2}(\theta - \theta_{\text{old}})^T \nabla_\theta^2 \overline{D_{KL}}(\pi_{\theta_{\text{old}}}, \pi_\theta)|_{\theta=\theta_{\text{old}}}(\theta - \theta_{\text{old}}) \\
&= \frac{1}{2}(\theta - \theta_{\text{old}})^T \nabla_\theta^2 \overline{D_{KL}}(\pi_{\theta_{\text{old}}}, \pi_\theta)|_{\theta=\theta_{\text{old}}}(\theta - \theta_{\text{old}})
\end{aligned} \tag{2.13}$$

If we use g to represent $\nabla_\theta L(\pi_\theta)|_{\theta=\theta_{\text{old}}}$ and A to represent $\nabla_\theta^2 \overline{D_{KL}}(\pi_{\theta_{\text{old}}}, \pi_\theta)|_{\theta=\theta_{\text{old}}}$, we are essentially trying to solve the following approximated optimization problem,

$$\max_\theta (\theta - \theta_{\text{old}})^T g, \text{ subject to } \frac{1}{2}(\theta - \theta_{\text{old}})^T A(\theta - \theta_{\text{old}}) \leq \delta$$

The local optimal solution of the above optimization problem must be a stationary point of the following Lagrangian form (assuming the KKT conditions hold), but not vice versa:

$$\mathscr{L}(\theta, \lambda) = (\theta - \theta_{\text{old}})^T g - \frac{\lambda}{2}\left[(\theta - \theta_{\text{old}})^T A(\theta - \theta_{\text{old}}) - \delta\right]$$

Differentiate $\mathscr{L}(\theta,\lambda)$ w.r.t θ,

$$\frac{\partial \mathscr{L}(\theta,\lambda)}{\partial \theta} = g - \lambda A(\theta - \theta_{\text{old}})$$

Setting $\frac{\partial \mathscr{L}(\theta,\lambda)}{\partial \theta} = 0$, we get

$$\theta - \theta_{\text{old}} = \frac{1}{\lambda} A^{-1} g$$

If we fix the value of λ throughout our learning process, this gives us an algorithm called natural policy gradient. It is shown that the natural gradient is moving to a greedily optimal action instead of just a better action which speeds up the convergence rate[Kakade, 2002]. Instead of fixing a single value of λ, TRPO does something smarter, namely it reduces the trust region (increase the value of λ) when expected improvement is not observed. We know the search direction is $A^{-1}g$, and we want to calculate the maximal allowable step size β such that $\theta_{\text{old}} + \beta s$ still satisfy the average KL constraint. This happens when $\frac{1}{2}(\theta - \theta_{\text{old}})^T A(\theta - \theta_{\text{old}}) = \delta$ using quadratic approximation to the KL term. We then have

$$\frac{1}{2}(\beta s)^T A(\beta s) = \delta \Rightarrow \beta = \sqrt{\frac{2\delta}{s^T A s}}$$

Hence we start with $\beta = \sqrt{\frac{2\delta}{s^T A s}}$ and perform a line search on $L(\pi_\theta) - \mathscr{X}\left[D_{KL}^{max}(\pi_{\theta_{\text{old}}}, \pi_\theta) \leq \delta\right]$, the value of β will be reduced until an improvement is observed. If no improvement is observed after reducing step size for a fixed number of times, the parameters will not be updated for this batch, this added line search guarantees that we are at least improving the surrogate objective function for collected samples. TRPO results in significantly better empirical performance and a more stable learning process compared to straightforward natural gradient policy optimization, though the changes are subtle.

The practical algorithm of TRPO is given below:

Algorithm 1 Practical Vanilla TRPO Algorithm

1: **for** `iteration=1, 2, ⋯` **do**
2: Collect samples using either single path or vine procedure
3: Compute raw discounted reward as the Monte Carlo estimate of Q-values.
4: Compute search direction $A^{-1}g$
5: Do line search to find an appropriate size of the trust region, namely the step size
6: **end for**

We implemented the above vanilla TRPO method from scratch using TensorFlow and had it tested on one continuous state and action task in OpenAI Gym called "Swimmer-v1". The objective of the task to make the swimmer swim forward as fast as possible. The state dimension is 8 and the action dimension is 2. The policy we used is diagonal Gaussian distribution with the mean determined by a one-layer neural net with tanh activation function. The standard deviation is just a parameter r which is independent of state, namely

$$a \sim \mathcal{N}(\text{NeuralNet}(s;\{\mathbf{W},\mathbf{b}\}),\exp(r))$$

The use of Gaussian distribution allows us to compute the analytical KL divergence between policy action distributions. The following performance results are obtained for "Swimmer-v1" task after averaging over 10 runs, we plot the mean along with its $\pm$ one standard deviation bounds.

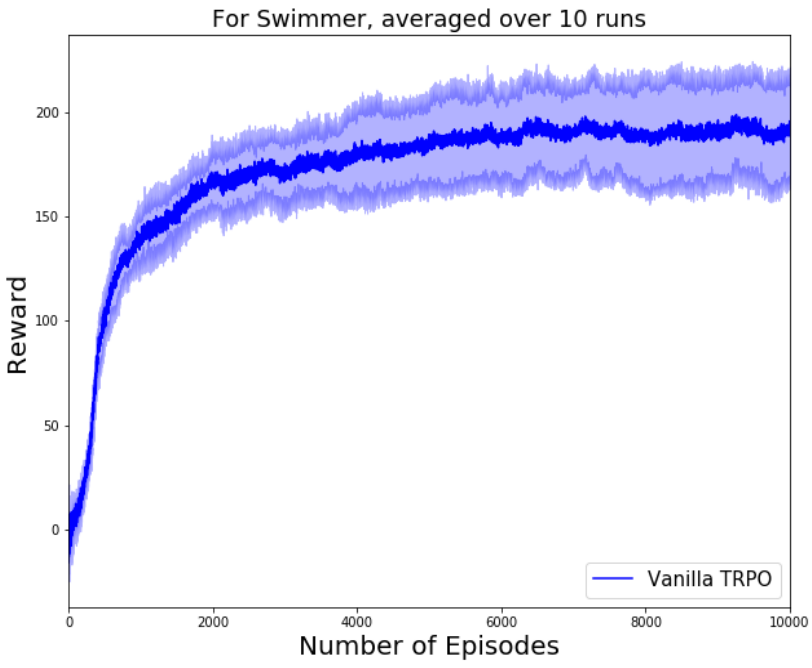


Fig. 2.3 The above learning is obtained with $\delta = 0.01$ and the parameters are updated after every 10000 time steps, the hidden nodes is set to be 10.

One of the run is recorded and uploaded to OpenAI Gym; the generated result is attached below:

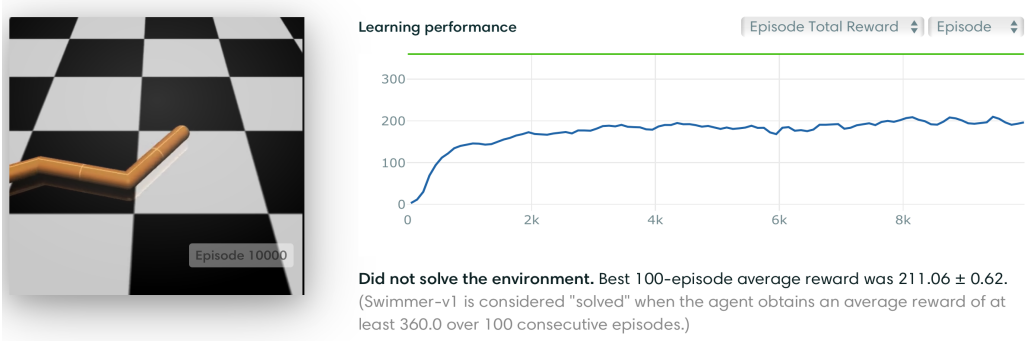


Fig. 2.4 https://gym.openai.com/evaluations/eval_orl9Wf5QHK9JpJKy4bIMA

It is interesting to note that the author of TRPO also uploaded a result by using TRPO-GAE[Schulman et al., 2015b] which is a later version of TRPO and obtain a less impressive performance on this task. This is an indication of how sensitive these algorithms are to initialization and hyperparameter setup.

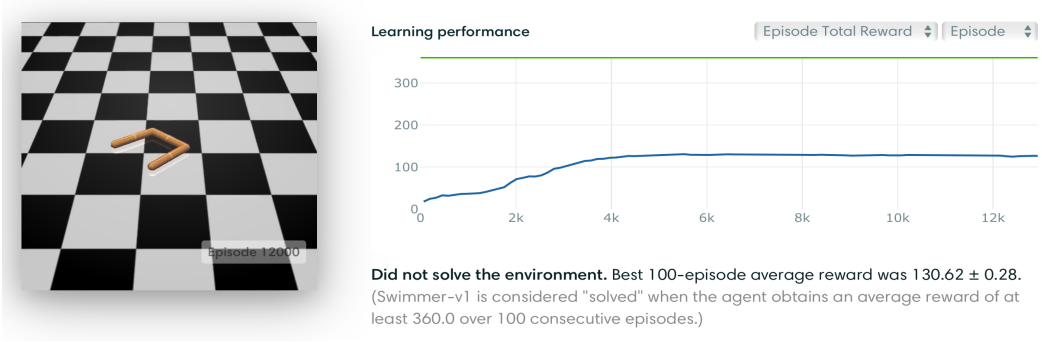


Fig. 2.5 https://gym.openai.com/evaluations/eval_UHEsnOhSw5cZd6uldLig

2.2.3 Proximal Policy Optimization

Proximal Policy Optimization(PPO) [Schulman et al., 2017] is a very recent policy gradient method, its prototype is a similar version of TRPO. It converts the trust region constrained optimization problem, Equation 2.11 into the following unconstrained optimization problem

$$\max_{\theta} \mathbb{E}_{s\sim\rho_{\pi_{\theta_{\text{old}}}},a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}Q_{\pi_{\theta_{\text{old}}}}(s,a) - \alpha D_{KL}(\pi_{\theta_{\text{old}}}(a|s)||\pi_\theta(a|s))\right] \tag{2.14}$$

It turns out it is difficult to fix a single value of α that performs well across different problems or even within a single problem as the collected samples change with changing policy. The raw version of PPO suggests use an adaptive α value, where we increase the value of α

if the KL measure is too big and vice versa. If we use first order approximation to L and quadratic approximation to the KL term as before, this basically becomes natural policy gradient with adaptive step size which is a rather trivial modification. However the main contribution of PPO comes from its later modification. It suggests maximizing the following clipped objective function, L^{CLIP}, instead of Equation 2.14.

$$\max_{\theta} \mathbb{E}_{s\sim\rho_{\pi_{\theta_{\text{old}}}},a\sim\pi_{\theta_{\text{old}}}}\Big[\min(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}Q_{\pi_{\theta_{\text{old}}}}(s,a), clip(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1-\varepsilon, 1+\varepsilon)Q_{\pi_{\theta_{\text{old}}}}(s,a))\Big] \tag{2.15}$$

where ε is a hyperparameter, usually 0.1 or 0.2.

The crux of obtaining stable performance in gradient-based policy optimization lies within ensuring small changes between consecutive policies. In TRPO or natural gradient policy, we use KL divergence to penalize big changes in polices. While in Equation 2.15, we impose a hard clip on the probability ratios between policies. This provides an alternative way to do a trust region update without having to compute the KL penalty. A minimum of the clipped and unclipped objective is taken as the final objective which gives a lower bound of the policy performance. From the following figure, we can see the objective function for a single state-action pair is bounded above by $(1+\varepsilon)Q$ and $(1-\varepsilon)Q$ when the Q value is positive and negative respectively. This implies we only use the clipped objective function when it is greater than these upper bounds, resulting in a smaller(more pessimistic) step size.

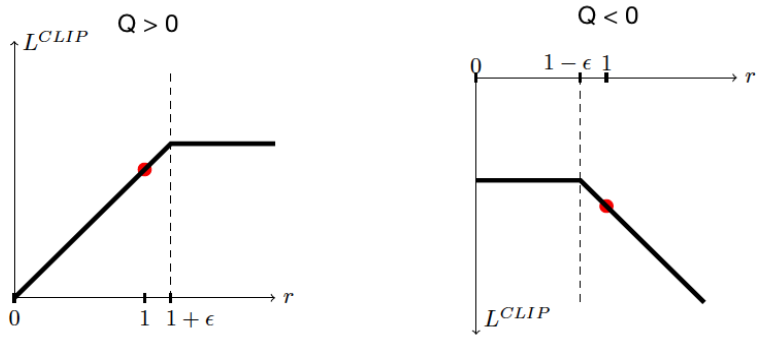


Fig. 2.6 The red circle indicates the starting point for the optimization, where r is the ratio between policies, i.e., $r = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$[Schulman et al., 2017].

PPO is clearly more robust than vanilla policy gradient algorithm as a result of the clipping. At the meantime, it is much computationally cheaper than TRPO or natural gradient policy as it only involves a first-order optimization task. Empirically, this method would seem to work surprising well, achieving better sample complexity on a collection of benchmark tasks.

Chapter 3

Variance Reduction Methods

3.1 Introduction

Apart from choosing an appropriate step size, another main challenge in gradient-based policy optimization is getting an accurate estimate of the gradient itself. Typically, the variance of our Monte Carlo gradient estimator tends to be high. Policy gradient techniques can require an unreasonably large number of samples in order to estimate the gradient accurately. In this chapter, we will introduce two new techniques for variance reduction.

Recall from Chapter 2, the gradient of our surrogate objective function for TRPO is

$$g = \nabla_\theta \mathbb{E}_{s\sim\rho_{\pi_{\text{old}}},a\sim\pi_{\theta_{\text{old}}}}\Big[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}Q_{\pi_{\theta_{\text{old}}}}(s,a)\Big] \tag{3.1}$$

If we let $Z = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}Q_{\pi_{\theta_{\text{old}}}}(s,a)$ and $\hat{Z}$ to represent its Monte Carlo approximation over n state-action pairs, the variance of the gradient estimator depends on the variance of $\hat{Z}$ as taking the gradient is a deterministic linear operation, which does not introduce any extra variance. The variance of $\hat{Z}$ is given below, we assume all our samples are independent and identically distribution for simplicity (though this is unlikely to be strictly true in practice). In the following, we denote 'var()' to mean the variance operator, and 'cov(.,.)' to represent the covariance between two variables.

$$\begin{aligned}\text{var}(\hat{Z}) &= \text{var}(\frac{1}{n}\sum_{i=1}^{n} Z_i) \\ &= \frac{1}{n}\text{var}(Z)\end{aligned} \tag{3.2}$$

We can see that if the number of samples per batch is not large enough, we will have a poorly performing, high variance gradient estimator (the estimate converges at a rate proportional to $\frac{1}{\sqrt{n}}$). In this chapter, we will be exploring different methods for reducing variance in these Monte Carlo estimates of policy gradient, without increasing the number of samples collected. These techniques can be equally applicable to any policy gradient methods, be it the REINFORCE algorithm, TRPO or PPO.

A problem with the gradient-based methods in previous chapter is that if the reward is always positive, the agent will try to increase the probability of every state-action pair in the sample set. Typically, a baseline is introduced to compare with the state-action value, if the reward for a state-action pair is higher than the baseline, we try to increase its probability and vice versa. The baseline can be any function, it is able to give us the same expectation of gradient estimator as long as the baseline does not depend on action. If we subtract a baseline b(s) from $Q_{\pi_{\theta_{\text{old}}}}(s,a)$ in Equation 3.1, the gradient becomes

$$\begin{aligned}
&\nabla_\theta \mathbb{E}_{s\sim\rho_{\pi_{\theta_{\text{old}}}},a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}(Q_{\pi_{\theta_{\text{old}}}}(s,a)-b(s))\right]\\
&= g-\nabla_\theta \mathbb{E}_{s\sim\rho_{\pi_{\text{old}}},a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}b(s)\right]\\
&= g-\nabla_\theta \sum_s \rho_{\pi_{\theta_{\text{old}}}}(s)\sum_a \pi_\theta(a|s)b(s)\\
&= g-\sum_s \rho_{\pi_{\theta_{\text{old}}}}(s)b(s)\nabla_\theta 1\\
&= g
\end{aligned} \tag{3.3}$$

We can see the value of the gradient estimator is unchanged after subtracting the baseline. Let $W=\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}b(s)$, the variance of Monte Carlo estimator of $Z-W$ is given as follows,

$$\begin{aligned}
\text{var}(\frac{1}{n}\sum_{i=1}^{n} Z_i-W_i) &= \frac{1}{n}\Big[\text{var}(Z)+\text{var}(W)-2\text{cov}(Z,W)\Big]\\
&= \text{var}(\hat{Z})-\frac{1}{n}(2cov(Z,W)-\text{var}(W))
\end{aligned} \tag{3.4}$$

If Z and W are strongly correlated and the variance of W is small enough, such that $2\text{cov}(Z,W)>\text{var}(W)$, then the variance of our estimator can be reduced. By introducing a action independent baseline function, we might be able to reduce variance without introducing any bias. [Greensmith et al., 2004] gives a much more detailed review on using

these 'control variates' to reduce variance for Monte Carlo gradient estimates within RL.

There are several orthogonal efforts to introduce a bias-variance trade-off into the gradient estimate, which have been shown to be successful[Gu et al., 2017; O'Donoghue et al., 2016; Schulman et al., 2015b; Wang et al., 2016]. Interpolated policy gradient[Gu et al., 2017] mixes some ratio of on-policy gradient with off-policy deterministic gradients, exploiting the benefits of both, and provides a unifying view of various different variance reduction methods in RL. In addition, the introduction of a discount factor can also serve the purpose of reducing variance in the gradient estimates. The higher the discount factor, the greater the reduction in variance (but typically at significant cost in increased bias).

3.2 Using K-Nearest Neighbors to Approximate the Value Function

One natural choice of baseline is the estimate of the state value function, $\hat{V}(s)$. The motivation behind this is that it is generally believed that the value function has lower variance than the Q function, and they are expected to be highly correlated. This ensures Equation 3.4 is dominated by 3.2. There has been various attempts to model the value function, typically the objective function is to minimize the distance between the predicted value function and the true value function. However such value function modeling usually requires a second high capacity model, with many more tunable parameters. In this section, we present a novel (in the detail) method to approximate the value function using a simple non-parametric model.

The value function under policy π, $V_\pi(s)$, is defined as follows:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_{a\sim\pi}[Q_\pi(s,a)] \\ &\approx \frac{1}{n}\sum_{i=1}^{n} r(s,a_i) \end{aligned} \tag{3.5}$$

The idea is that if we are able to sample a large number of actions from state s under policy π, $V_\pi(s)$ can be roughly approximated by averaging over all the raw rewards, $r(s,a_i)$ where $a_i \sim \pi(a|s)$. The problem is in practice such sampling can not be done easily in a model-free setting, as it requires the state to be restored between samples. Even if we are able to achieve that, it would also be very sample inefficient. However if we assume there is some degree of smoothness in the state-action space, we can use the k nearest neighbors (including itself)

of $Q_\pi(s,a)$ in the current batch (measured in terms of the standard Euclidean distance in the state space, which may be suboptimal) to approximate $V_\pi(s)$. If we let the Euclidean distance between the i^{th} neighbor to the point of interest be d_i. $V_\pi(s)$ is approximated using the following expression:

$$V_\pi(s) = \sum_{i=1}^{k} w_i \times r(s_i, a_i) \tag{3.6}$$

where $w_i = \frac{\exp(-\alpha d_i)}{\sum_{j=1}^{k} \exp(-\alpha d_j)}$, α is a tunable parameter, indicating the degree of penalty we want to impose on distance.

We compare the above method with vanilla TRPO on three different environments in OpenAI Gym. In all three environment, the state space is continuous and the action space is discrete. Let s represents the input state of dimension $n \times 1$ and let a represents the action space of dimension m. Then the distribution of our policy is given by

$$\pi(\cdot|s) = \text{softmax}(\theta^T s + b),$$

where θ is the weight of dimension $n \times m$, b is the bias of dimension $m \times 1$.

The following results are obtained after averaging over 50 runs under the same random seed for each method. The mean reward along with its one standard deviation bounds are displayed. We can see the above method is able to converge faster as well as find a better policy in all three cases than the vanilla TRPO without baseline. However a fairer comparison would involve a comparison of the performance of this method against other existing parametric methods for estimating the value function.

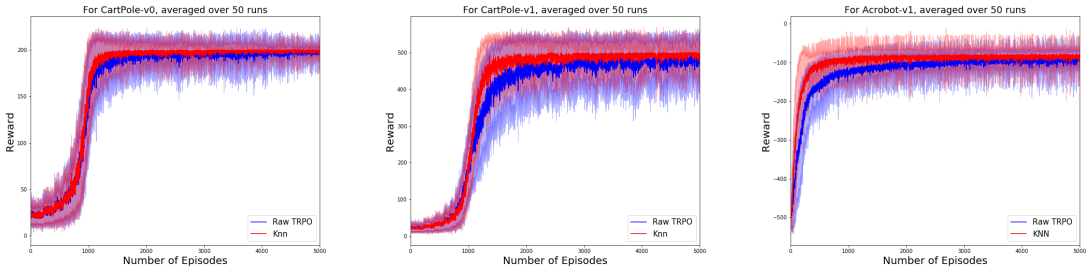


Fig. 3.1 The policy update is performed after every 5000 time steps, the number of nearest neighbor is chosen to be 50, $\alpha = 10$, $\delta = 0.01$.

3.3 Weighted Gradients with Policy Proximity Information

Even if we have an effective baseline, the main problem with the previous policy gradient methods is that they can only make use of the on-policy samples (if we want to avoid bias). This implies each batch of samples collected is only used once—to make estimates for a single update to the *current* policy. This is very sample inefficient. In this section, we will present a novel method, Weighted Gradients with Policy Proximity Information(WGPPI) to reduce variance using off-policy samples at the cost of introducing some bias.

The difficulty in calculating gradient in RL is that different state-action samples are generated with different policies. This restricts us to use previously collected samples under a different policy. However if the change in the policies is small enough, the previous samples must encode at least some information regarding the current policy. One way to measure the proximity between policies is via symmetric Kullback–Leibler (KL) divergence, though other distance measures such as (approximate) Jensen–Shannon or indeed (approximate) Wasserstein divergence can also be considered. Assume the current parameter is θ_n and the previous parameters from each batch are $\theta_1, \theta_2, ..., \theta_{n-1}$. Then the symmetric KL divergence between current policy and policy in j^{th} batch is calculated as follows:

$$\mathrm{KL}_{\mathrm{symm}}(\pi_{\theta_n}||\pi_{\theta_j}) = \frac{1}{2}\Big(\mathbb{E}_{s\sim\rho_{\theta_n}}[D_{\mathrm{KL}}(\pi_{\theta_n}(\cdot|s)||\pi_{\theta_j}(\cdot|s))] + \mathbb{E}_{s\sim\rho_{\theta_j}}[D_{\mathrm{KL}}(\pi_{\theta_j}(\cdot|s)||\pi_{\theta_n}(\cdot|s))]\Big) \tag{3.7}$$

It can then be approximated using samples from different batches, assuming there are N samples in each batch. The Monte Carlo estimator of the above symmetric KL divergence for discrete action space is given as follows.

$$\begin{aligned}\mathrm{KL}_{\mathrm{symm}}(\pi_{\theta_n}||\pi_{\theta_j}) \approx \frac{1}{2N}\Big(&\sum_{s_i\sim\rho_{\theta_n}}\sum_a p(a|s_i,\theta_n)\log\frac{p(a|s_i,\theta_n)}{p(a|s_i,\theta_j)} \\ &+ \sum_{s_i\sim\rho_{\theta_j}}\sum_a p(a|s_i,\theta_j)\log\frac{p(a|s_i,\theta_j)}{p(a|s_i,\theta_n)}\Big)\end{aligned} \tag{3.8}$$

We will use these distance measures to determine the associated weights that previous raw gradients should have on predicting the current one. If we use d_{nj} to denote $\mathrm{KL}_{\mathrm{symm}}(\pi_{\theta_n}||\pi_{\theta_j})$, then the corresponding weight w_j used to decide g_{WGPPI_n} is defined as follows:

$$w_j = \frac{\exp(-\beta d_{nj})}{\sum_{k=1}^{n}\exp(-\beta d_{nk})}$$

where β is a tunable parameter. It determines how much penalty we impose on the distance between policies. In the limiting case when $\beta \to \infty$, this is almost equivalent to the vanilla form of TRPO where we only use gradient estimate from current batch.

And the weighted gradient g_{WGPPI_n} is calculated as below:

$$g_{WGPPI_n} = \sum_i^n w_i g_i \tag{3.9}$$

where g_i is estimated in the same manner as in Chapter 2.

Now let us consider the variance of the new weighted gradient,

$$\begin{aligned}
var(\sum_i^n w_i g_i) &= \sum_i^n w_i^2 \; var(g_i) + \sum_i^n \sum_{j \neq i} w_i w_j \; cov(g_i, g_j) \\
&\leq \sum_i^n w_i^2 \; var(g_i) + \sum_i^n \sum_{j \neq i} w_i w_j \sqrt{var(g_i) var(g_j)} \\
&\leq \sum_i^n w_i^2 \; var(g_i) + \sum_i^n \sum_{j \neq i} w_i w_j \; \frac{var(g_i) + var(g_j)}{2} \\
&= \frac{1}{2} \sum_i^n \Big(2 w_i^2 \; var(g_i) + w_i \sum_{j \neq i} w_j (var(g_i) + var(g_j)) \Big) \\
&= \frac{1}{2} \sum_i^n w_i \Big(2 w_i \; var(g_i) + var(g_i) \sum_{j \neq i} w_j + \sum_{j \neq i} w_j \; var(g_j) \Big) \\
&= \frac{1}{2} \sum_i^n w_i \Big(2 w_i \; var(g_i) + var(g_i)(1 - w_i) + \sum_{j \neq i} w_j \; var(g_j) \Big) \\
&= \frac{1}{2} \sum_i^n w_i \Big(w_i \; var(g_i) + var(g_i) + \sum_{j \neq i} w_j \; var(g_j) \Big)
\end{aligned} \tag{3.10}$$

We make the assumption that the variances of all raw gradients are roughly the same, and equate this to X. Then we can show that:

$$\begin{aligned}
var(\sum_i^n w_i g_i) &\leq \frac{1}{2} \sum_i^n w_i \Big(w_i X + X + (1 - w_i) X \Big) \\
&= X
\end{aligned} \tag{3.11}$$

This shows if the variance of raw gradient estimators are roughly the same, the WGPPI method is guaranteed not to give more variance than before. And if the variance of gradient estimators is increasing, then we are guaranteed to get a lower variance. In practice, this is not

unreasonable as we have seen in Fig 2.3, the variance is actually typically increasing as the policy learning process proceeds, making this variance reduction method particularly suitable.

In the vanilla TRPO method, when the improvement is not observed after shrinking the step size for a few times, we do not perform any update, this implies our parameters will not change at all and the previous samples can indeed be used for the next update. These previous samples are discarded in the original TRPO implementation whereas the WGPPI scheme is able to effectively increase the batch size per update with no added bias, which leads to a more stable learning process. Even in the case when β is set to be a very large number, our weighted scheme still has an advantage over the original TRPO. This is because when the policy parameters starts to converge and the updates are small, In vanilla TRPO method, we might be susceptible to a sudden noise in the collected samples which gives us a wrong direction to move, whereas in the WGPPI method, the new gradient is able to take into account of all previous samples (and hence the single outlier does not adversely affect the gradient for that update much). In addition, since the optimization problem for learning the policy parameters is not convex, WGPPI method might give us the added advantage of escaping a local optimum even when the raw gradient estimate is relatively accurate. Such weighting scheme can be also be applied to the KNN method in the previous section or more generally in making use of previous off-policy samples to perform on-policy learning algorithm within the RL framework.

The practical algorithm for performing WGPPI is given below.

Algorithm 2 Practical WGPPI-TRPO Algorithm

1: Initialize an empty memory M to store parameters and raw gradients for different batches.
2: **for** `i=1, 2, ⋯` **do**
3: Store parameter values for current batch into M
4: Collect samples using single path rollout
5: Compute raw discounted reward as the Monte Carlo estimate of Q-values.
6: Compute raw gradient, g_i and append it to M
7: Calculate symmetric KL divergence and interpolated g_{WGPPI_i}
8: Compute search direction $A^{-1}g_{WGPPI_i}$
9: Do line search to find an appropriate size of the trust region, namely the step size
10: **end for**

In our implementation, we used a fixed memory size of 50 to store the raw historical gradients up until the current batch (although theoretically larger is better). Therefore when

the gradient from the 51^{st} batch comes in, the gradient from the first batch will be ejected from memory, in a first-in-first-out manner. Though a bigger memory size could be helpful if not limited by the computational resources. For the first gradient estimate, it will be unchanged (same as vanilla TRPO) as we have no history yet. These values are then re-weighted to form the new weighted gradient for the current batch using Equation 3.9.

Similarly we test the above algorithm using the following three tasks within OpenAI Gym, with the following parameters setting: $\beta = 500$, $\delta = 0.001$. The figures show the mean reward as well as its one standard deviation bounds. We have decided to use a very small sample size per update so that the variance of our gradient estimator is relatively large, which is the case for most tough RL problems in the wild. We observe a faster convergence rate over all three tasks.

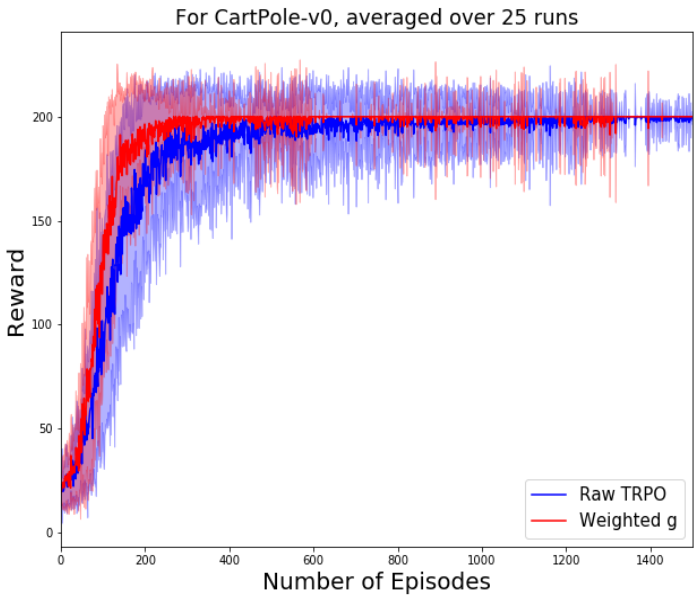


Fig. 3.2 CartPole-v0, the policy parameters are updated once the collected trajectory has more than 100 time steps

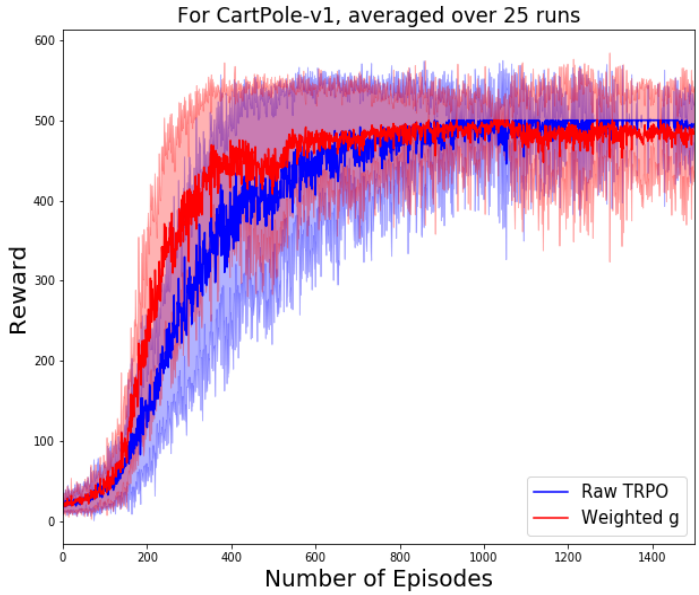


Fig. 3.3 CartPole-v1, the policy parameters are updated once the collected trajectory has more than 250 time steps

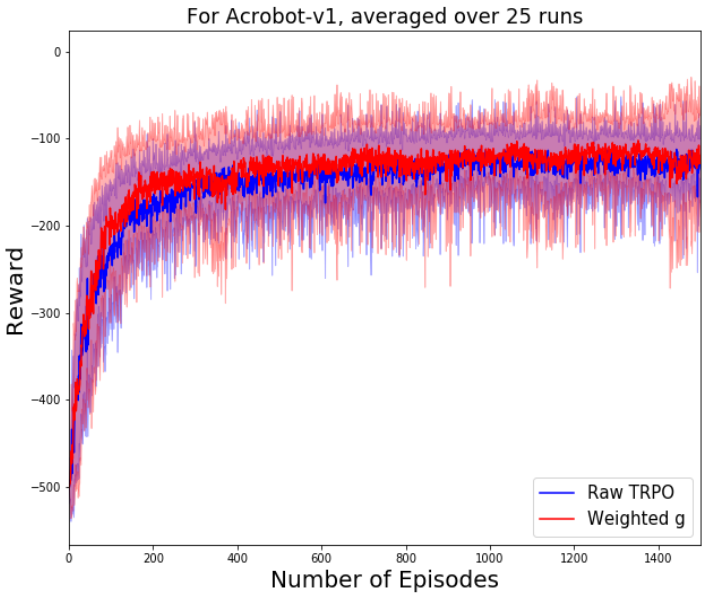


Fig. 3.4 Acrobot-v1: the policy parameters are updated once the collected trajectory has more than 200 time steps

Chapter 4

Structured Reinforcement Learning Policy Optimization

4.1 Introduction

In some RL problems, there might be multiple different optimal policies which can solve the same task successfully. In most of the cases, our predefined single model might not have enough modeling power to encode the space of all possible policies. For example, in the case of the puddle world task as shown below, the reward of the task is given by the negative length of the path and an extra penalty is imposed for passing through the puddles. It is difficult for a single policy approach to effectively solve this problem, as the policy would tend to be averaged over both paths. However as suggested in [Daniel et al., 2012], incorporating a hierarchical structure is able to effectively find both optimal solutions and in doing so gives rise to a faster convergence rate.

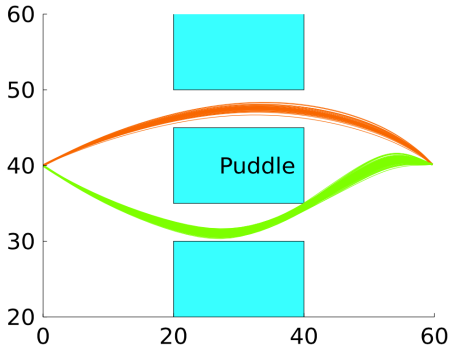


Fig. 4.1 HiREPS[Daniel et al., 2012] is able to find both paths for this task and converge much faster than REPS[Peters and Mülling, 2010]—which uses a non-hierarchical policy.

Hierarchical Relative Entropy Policy Search (HiREPS) makes use of something called the 'options framework'[Sutton et al., 1998, 1999]. Apart from the option framework, there are two other approaches to hierarchical RL, namely Hierarchies of Abstract Machines (HAMs) [Parr and Russell, 1998] and the MAXQ framework[Dietterich, 2000]. A detailed explanation of each of them can be found in [Barto and Mahadevan, 2003]. We will only be discussing the option framework in this thesis.

The general idea behind the option framework is to have a policy over different sub-policies, where each sub-policy is executed for a period of time until a termination criteria is triggered, and another sub-policy is selected. Mathematically, an option consists of the following three components: a policy $\pi : S \times A \mapsto [0,1]$, a termination condition $\beta : S \mapsto [0,1]$ and an input set $I \subseteq S$. An option $< \pi, \beta, I >$ is available in state s if and only if $s \in I$. We also need a policy over different options to determine which option to select after the old option terminates according to β. This option framework is able to exploit temporal abstraction, learning to regime switch through time. It has been successfully implemented in [Bacon et al., 2017; Daniel et al., 2012].

However, as pointed out in [Daniel et al., 2012], the modeling of temporal consistency β is difficult in and of itself. To reduce unnecessary complexity, we will limit the option to exist for one time-step. This means that the currently selected option only depends on the current state, but not on previously selected options, $\pi(o|s,o') = \pi(o|s)$. By doing this, we avoid the overhead incurred by estimating β, yet are still able to use different sub-policies at different time steps.

4.2 Hierarchical Trust Region Policy Optimization

We let o to represent an option and $\pi(a|o,s)$ to represent its corresponding policy over action space which we will call a sub-policy. We assume the input set I is defined over the whole state space for all options. $\pi(o|s)$ represents the policy over different sub-policies, which we will call it the gating network. Hence we can express our hierarchical policy $\pi(a|s)$ as follows:

$$\pi(a|s) = \sum_o \pi(o|s)\pi(a|o,s)$$

Assuming Algorithm 1 in Chapter 2 indeed gives monotonic improvement (though this might not in practice be the case given we have used a few approximations in our optimization), we can apply the algorithm to the sub-policies as well as the gating network to ensure they

give monotonic improvement separately. While updating our sub-policies, we fix the gating network, and while updating our gating network, we fix the sub-policy; intuitively this should give rise to a hierarchical policy with monotonic improvement. Alternatively both parameters can be updated together though this will be more complicated as it involves sum over all options. The former one leads us to the following vanilla, structured variant of the TRPO algorithm.

Algorithm 3 Vanilla Hierarchical TRPO

1: Initialize both gating network ϕ_{old} and sub-policy parameters θ_{old}
2: **for** $i = 1 : N$ **do**
3: Reset the initial state and sample $o \sim \pi_{\phi_{\text{old}}}(o|s)$, say we get o_k
4: **for** $j = 1 : M$ **do**
5: Follow $a \sim \pi_{\theta_{\text{old}}}(a|s, o_k)$
6: If done: reset environment, j++
7: **end for**
8: Use TRPO to update the parameters of a particular sub-policy, θ_k

$$\max_{\theta_k} \mathbb{E}_{s,a\sim\pi_{old}}\Big[\frac{\pi_\theta(a|o_k,s)}{\pi_{\theta_{\text{old}}}(a|o_k,s)} Q_{\pi_{old}}(s,a,o_k)\Big] \tag{4.1}$$

subject to $\overline{D_{\text{KL}}}(\pi_{\theta_{old}}(\cdot|o_k,s)||\pi_\theta(\cdot|o_k,s)) \leq \delta_{\text{sub-policy}}$

9: **end for**
10: Use TRPO to update the gating network parameters, ϕ

$$\max_{\phi} \mathbb{E}_{s,a,o\sim\pi_{old}}\Big[\frac{\pi_\phi(o|s)}{\pi_{\phi_{\text{old}}}(o|s)} Q_{\pi_{old}}(s,a,o)\Big] \tag{4.2}$$

subject to $\overline{D_{\text{KL}}}(\pi_{\phi_{old}}(\cdot|s)||\pi_\phi(\cdot|s)) \leq \delta_{option}$

11: Go back to step 2

In the hierarchical TRPO algorithm we only need to specify the number of options at the start of a learning process, no extra information is required. The limitation of the above algorithm is that it executes the same option throughout the entire episode. This limits the exploration for our option policy. We are essentially assuming that any state has a non-zero probability of being selected as a starting state and if we sample enough episodes, we should be exploring enough of the state spaces for different options. This is called 'the assumption of exploring starts' in [Sutton and Barto, 1998]. We can then stop the learning algorithm when parameters starts to converge. And use this set of learned parameters to perform the task allowing for the selection of different options within a single episode.

However a more effective and direct way to achieve the same end result, is to allow the agent to choose different options at different time steps during training. In such cases, we need to create a memory buffer to store the corresponding observations and actions, as well as the rewards received by different sub-policies. The update of parameters with regards to a certain sub-policy is carried out once the length of our buffer memory exceeds a threshold length. Then this memory buffer is cleared (after updating). Such an updating method has the added advantage of using less correlated samples in the Monte Carlo estimation stage as compared to vanilla hierarchical TRPO, where samples within the same episode may be highly correlated. A downside of this alternative process, however, is that some samples in the buffer memory might not be generated by the current policy. For example while we collect samples for a particular sub-policy, the gating network may be updated. This gives some bias to our estimation. In order to ensure the bias is small, we must update the sub-policy parameters frequently, however this gives rise to the high variance problem. This is where the variance reduction techniques introduced in Chapter 3 comes into play. If we combine the above method with WGPPI, we get the following algorithm:

Algorithm 4 WGPPI-Hierarchical TRPO (allow different options within a single episode)

Initialization of various parameters and memory buffer.
for $episode = 1 : N$ **do**
    Reset the initial state.
    while If not done **do**:
        Sample $o \sim \pi_\phi(o|s)$, then sample $a \sim \pi_\theta(a|s,o)$
    end while
    Store samples, actions, options and rewards into relevant buffer memory.
    if the memory buffers exceed a pre-defined length **then**
        Use WGPPI-TRPO to update parameters and empty corresponding memory buffer
    end if
end for

We apply the above WGPPI-Hierarchical TRPO along with the vanilla Hierarchical TRPO algorithm on CartPole-v0. This is not a suitable environment for testing our hierarchical algorithm (there is no hierarchy in the state-action space to exploit), however a hierarchical policy should still be capable of solving the problem satisfactorily (although convergence is expected to be slow). This setup is sufficient to test whether adding the variance reduction methods we presented in previous chapters are helpful or not. The experiments are averaged over 25 runs with the mean and one standard deviation bounds shown below. From the following graph, we can see the incorporation of WGPPI is able to give a slightly better performance than without (with no tuning of hyperparameters). Significantly more testing is

required to empirically evaluate the performance of this combination of methods.

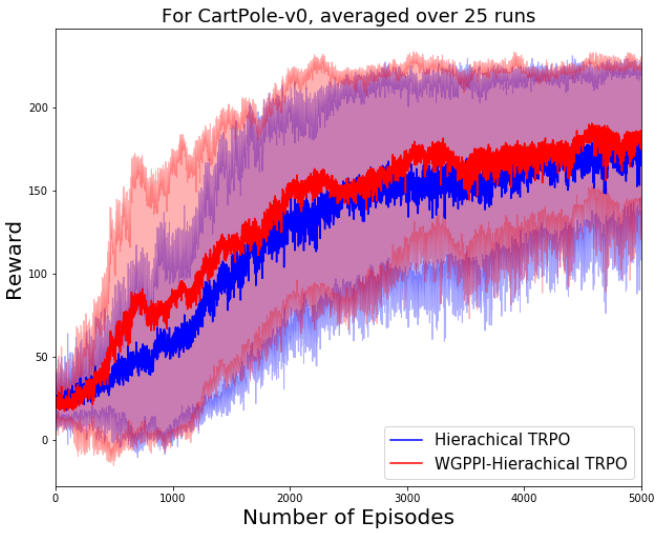


Fig. 4.2 The number of options is initialized to be 2, the sub-policy updates itself once the length of the buffer memory hits 100; option selection policy updates its parameters when the buffer memory hits 50 samples, δ for both sub-policy and option policy is 10^{-3}. The memory size for storing previous gradients is fixed to be 50 as before with the same alpha size of 50 (some tuning might be needed to achieve better results).

Chapter 5

Conclusion and Future Work

In summary, we have given a high level overview on RL and detailed reasoning for choosing model-free and policy-based learning methods over and above model-based and value-based approach. We explained several policy-gradient RL algorithms along with their common limitation: namely sample innefficiency as a result of only exploiting on-policy samples. We then present two alternative methods to tackling the high gradient estimator variance problem found in PGRL, and demonstrate improvement over three tasks in OpenAI Gym. WGPPI makes use of policy proximity information, and can be applied generally to on-policy algorithms to allow for the effective use of off-policy samples. Finally, we present the structured version of TRPO and show that by making use of WGPPI, we are able to effectively improve the performance—demonstrating the value of WGPPI even given a hierarchical policy.

Due to the time limit of this MPhil thesis, a number of potential experiments and other interesting related areas have not been explored. If more time was available, the following areas would be investigated:

- Due to time constrains and some setup issues within RLLAB[1], we are unable to run the Hierarchical TRPO codes on some truly hierarchically environment. From [Bacon et al., 2017], 'sequest' in Atari 2600 is considered as a suitable environment for the option framework, however, due to computational constraints, we are unable to find a suitable CNN architecture for both the vanilla TRPO as well as the Hierarchical one. The 'swimmer gather' from RLLAB would also be a good environment to test the algorithm on.
- The hierarchical TRPO algorithm can be further extended to determine the suitable number of options within a task. We only need to provide a maximum allowable

[1] https://rllab.readthedocs.io/en/latest/

number of options at the start of the training process, and an option will be abandoned if $\pi(o|s)$ gets too small. To prevent an option being thrown away prematurely, we need to ensure each option has been sampled a minimum number of times before deletion happens. Ideally a mechanism for generating new options is also present, which could be trivially introduced by modelling the overall policy as a dirichlet-process mixture of sub-policies.

- In Chapter 3, we only explored using symmetric KL divergence as a distance measure of policy proximity information, however it would be nice to compare a range of other distributional distance measures (or appropriate approximations), for example, Jensen–Shannon divergence, total variation distance, Wasserstein metric and etc.

- The parameter values chosen are not being tuned systematically, most of the values are chosen to be the same as the setup in the original TRPO paper. Grid search and Bayesian optimization could be used to find more suitable parameters under our structure.

- It would also be interesting to consider ensemble methods with several different policies, where we either use a voting scheme for discrete action space or some weighted interpolation for continuous action space. In addition to that, for continuous action space, we can come up with different sub-policies gradually, namely we train one policy first and when the parameter starts to converge, we add in another policy but giving it a smaller impact on deciding the action, this is basically the idea of residue modeling (or boosting) which works well as an approach to regression.

- While training for policies, we realized that there are sudden fluctuations in the performance even though the agent is able to give good performance most of the time. This might imply we have 'overfit' our policy with the limited amount of previously collected samples. One of the ways to avoid overfitting is to prevent extreme weights, we can simply add some weights regularizers to control their magnitudes. If our policy is modeled using a neural network, we can apply dropout as another means to prevent overfitting. A further possible approach is to slightly perturb our parameters through adding noises. If the policy does not overfit our sampled data, we should expect a negligible changes in terms of the performance (the performance should exhibit some degree of smoothness around the learned solution). This provides a more robust way to do policy optimization. Furthermore, another problem with on-policy optimization algorithms is that they are sensitive to initialization and might prematurely converge to a local optimum. By adding noise to the parameters, it can also lead to more

consistent exploration and a richer set of behaviors[Plappert et al., 2017]. Note that by adding noises to parameter space, we are able to perform state dependent exploration *instead* of state independent exploration if we just add random noises to the action space[Rückstieß et al., 2008].

- Instead of introducing various sub-policies to learn a complicated task, another alternative is to re-design the reward function. If our reward function is able to change according to different states, it is essentially equivalent to getting different polices. The success of such approach is noted in [Held et al., 2017], it allows the agent to discover a range of tasks itself and learn how to reach different parts of the maze in the Ant Maze environment.

References

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. 2017.

Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

Dimitri P Bertsekas and John N Tsitsiklis. Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, 10(3):627–642, 2000.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Christian Daniel, Gerhard Neumann, and Jan R Peters. Hierarchical relative entropy policy search. In *International Conference on Artificial Intelligence and Statistics*, pages 273–281, 2012.

Peter Dayan and Yael Niv. Reinforcement learning: the good, the bad and the ugly. *Current opinion in neurobiology*, 18(2):185–196, 2008.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

Marc Deisenroth and Carl 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)*, pages 465–472, 2011.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. 2000.

Victor Gabillon, Mohammad Ghavamzadeh, and Bruno Scherrer. Approximate dynamic programming finally performs well in the game of tetris. In *Advances in neural information processing systems*, pages 1754–1762, 2013.

Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5 (Nov):1471–1530, 2004.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, Bernhard Schölkopf, and Sergey Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. *arXiv preprint arXiv:1706.00387*, 2017.

Nikolaus Hansen. Tutorial: Covariance matrix adaptation (cma) evolution strategy. 2006.

David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. *arXiv preprint arXiv:1705.06366*, 2017.

Filip Jurčíček, Blaise Thomson, and Steve Young. Reinforcement learning for parameter estimation in statistical spoken dialogue systems. *Computer Speech & Language*, 26(3): 168–192, 2012.

Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538, 2002.

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

Jens Kober and Jan R Peters. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856, 2009.

James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Pgq: Combining policy gradient and q-learning. *arXiv preprint arXiv:1611.01626*, 2016.

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

Jan Peters and Katharina Mülling. Relative entropy policy search. 2010.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

Thomas Rückstieß, Martin Felder, and Jürgen Schmidhuber. State-dependent exploration for policy gradient methods. *Machine Learning and Knowledge Discovery in Databases*, pages 234–249, 2008.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *ArXiv e-prints*, July 2017.

John Schulman and Pieter Abbeel. Deep reinforcement learning through policy optimization. In *Proceedings of the Thirtieth Annual Conference on Neural Information Processing Systems (NIPS)*, 2016. URL https://media.nips.cc/Conferences/2016/Slides/6198-Slides.pdf.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 387–395, 2014.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Satinder P Singh, Michael J Kearns, Diane J Litman, and Marilyn A Walker. Reinforcement learning for spoken dialogue systems. In *Advances in Neural Information Processing Systems*, pages 956–962, 2000.

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

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. 1998.

Richard S Sutton, Doina Precup, and Satinder P Singh. Intra-option learning about temporally abstract actions. 1998.

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

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Lex Weaver and Jonathan Baxter. Reinforcement learning from state and temporal differences. Technical report, 1999.

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

Ya-xiang Yuan. A review of trust region algorithms for optimization. 2000.