University of Oxford

The Optimal Baseline for Policy Gradient Methods in Reinforcement Learning

by

Shivakumar Mahesh

St Peter's College

A dissertation submitted in partial fulfilment of the degree of Master of Science in Statistical Science.

Department of Statistics, 24–29 St Giles,
Oxford, OX1 3LB

September 2023

This is my own work (except where otherwise indicated)

Candidate: Shivakumar Mahesh

Signed: [illegible]

Date: 11 September 2023

Abstract

Policy gradient methods are a family of reinforcement learning algorithms that optimize parameterized policies with respect to the expected return by stochastic gradient ascent. These methods are prone to high variance, and so a common variance reduction technique is to use control variate subtraction, also known as the *baseline trick*. Deriving the optimal baseline for policy gradient methods involves minimizing the total variance of the policy gradient estimator. The variance and the optimal baseline in turn greatly depend on how the states are sampled. Existing approaches to deriving the optimal baseline assume that states must be sampled from the discounted state distribution, which is inconsistent with the optimal baseline when states are sampled directly through simulation of an agent's actions in an environment or *rollouts*. In this work, we derive the optimal baseline for policy gradient methods when states are sampled through rollouts and compare our optimal baseline to the ones stated in the literature.

Acknowledgements

I would like to thank my supervisor Jakob Foerster for agreeing to carry out this project and for his guidance and insight. I would also like to thank Jakub Grudzien Kuba, the author of the paper *Settling the variance of multi-agent policy gradients*, for his clarifications.

Contents

1 Introduction

1.1 Motivation

Machine Learning (ML) is considered to be one of the most beneficial technologies to be born out of statistics and computer science. Due to its widespread adoption and broad applicability, we expect menial and repetitive tasks to gradually be replaced by machine learning software and intelligent robots to eventually replace humans in roles that endanger their safety and possibly their lives. Machine Learning (Shalev-Shwartz & Ben-David, 2014) studies algorithms that improve at a particular goal by receiving, manipulating and interacting with data from an environment. The idea that we learn through interacting with our environment is probably the first to occur to us when we think about the nature of learning.

Reinforcement learning (Sutton and Barto, 2018, RL) stands out as the most natural machine learning paradigm as it is much more focused on goal-directed learning from interaction than other approaches to machine learning. RL is concerned with the design of intelligent agents that must learn which actions to take in an environment in order to maximize their long-term cumulative reward. RL aims to solve the problem of learning optimal behaviour through interaction with the environment and receiving reward signals, thus making it applicable to a plethora of real-world problems. RL has been applied to the control of autonomous vehicles and drones (Shalev-Shwartz et al., 2016), control of robotics (Kober et al., 2013), packet routing (You et al., 2019), whilst simultaneously marking important milestones in AI progress through the Games of Go (Silver et al., 2016, 2017), Starcraft (Vinyals et al., 2019), Poker (Brown & Sandholm, 2019) and recently, Diplomacy (Bakhtin et al., 2022).

We will focus on one of the most popular families of algorithms in RL, known as policy gradient methods (Sutton et al., 1999). Policy gradient methods use stochastic gradient ascent directly on the expected return (long-term cumulative reward) and hence are prone to high variance. An extensively studied and successfully applied method to reduce this variance is through control variate subtraction (Ross, 2002), also known as the "baseline trick" in the RL literature (Greensmith et al., 2004; Gu et al., 2017; Weaver & Tao, 2013). This work focuses on the optimal baseline, which has been derived in a number of different RL settings (Greensmith et al., 2004; Kuba et al., 2021; Wu et al., 2018) under the assumption that states are sampled through a discounted state distribution (we define this distribution in Section 2.2). However, this sampling strategy is rarely used in practice. Instead, when a computer simulation of an agent's environment exists, states can easily be sampled through simulation of an agent's actions in an environment. We refer to this as sampling states through *rollouts*. In this work, we derive the optimal baseline for policy gradient methods when states are sampled through rollouts and analyse the differences between our baseline and the ones stated in the literature.

1.2 Contributions

Against the background of the above, the main contributions of this work are the following:

- First, we show that the well-established optimal baseline (Greensmith et al., 2004; Kuba et al., 2021; Wu et al., 2018; Peters & Schaal, 2006, 2008; Rückstieß et al., 2008; Jie & Abbeel, 2010; Zhao et al., 2011) is no longer optimal when the popular sampling strategy of sampling directly from rollouts is used.
- Next, we derive the optimal state-dependent baseline in Theorem 2 that is consistent with sampling from rollouts. The derivations for the optimal action-dependent factorized baselines (Wu et al., 2018) in reinforcement learning and optimal baselines in multi-agent reinforcement learning (Kuba et al., 2021) for states sampled through rollouts are analogous to the one presented in Theorem 2.
- Finally, we present experiments which validate that our optimal baseline and the ones in the literature are indeed different.

This work is structured as follows: in Section 2.1 we introduce the preliminaries of reinforcement learning and formalize the RL problem as a Markov decision process. In Section 2.2 we introduce the basics of policy gradient methods and develop the definitions and lemmas used in later sections. In Section 2.3 we describe the vanilla policy gradient algorithm. Next, in Section 3.1 we prove the main theorem of this work which states the optimal baseline when states are sampled through rollouts. In Section 3.2 we analytically compare our optimal baseline to the ones stated in the literature. In Section 4 we supply experiments that validate that the optimal baseline in our work and the ones in the literature are non-trivially different. Finally, in Appendix A we provide Python code for all the experiments.

2 Reinforcement Learning

2.1 Preliminaries

We begin by formalizing the RL problem as a Markov decision process (Sutton and Barto, 2018, MDP). MDPs are a mathematically idealized form of the reinforcement learning problem for which precise theoretical statements can be made. An MDP is induced by an agent acting in an environment $\mathcal{E}$ and is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, r, P, \gamma, \rho^0 \rangle$ where,

- $\mathcal{S}$ denotes the *state space*, the set of all *states* the environment can exist in. The state space depends on the problem to be solved. For example, in the game of noughts and crosses, a straightforward representation of the state could be the position of each entry on the board captured by a 3×3 ternary matrix. The state space can be defined as the set of all 3×3 ternary matrices, that is $\{0, 1, 2\}^{3\times3}$ with 0 denoting a nought entry, 1 denoting a cross entry 2 denoting an unfilled entry. The same problem can be assigned a different state space simply by removing all the states in which the environment cannot exist in the game. For example, the 3×3 zero matrix is not a valid state that can be achieved with the above representation and such states can be removed. Therefore the state space is not unique for a given problem. Note that this is a discrete action space. For a different problem such as a pick-and-place robot, the states might be the latest readings of joint angles and velocities, hence the state space here is continuous.

- $\mathcal{A}$ denotes the *action space*, which is the set of actions that the agent can possibly take. In the noughts and crosses example, the action for the player selecting crosses is to pick any of the nine squares. A valid action space representation for the player is simply the set $\{1, \dots, 9\}$, which is a discrete action space. For the pick-and-place robot, it might be the voltages applied to each motor at each joint, a continuous action space.

- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the bounded reward function, which provides the instant reward for an agent taking an action in a given state. On one hand, the reward function for noughts and crosses might provide $+1$ for a win, -1 for a loss and 0 reward for all other state-action pairs. On the other hand, the pick-and-place robot might receive a -1 reward until the task is complete.

- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_0$ denotes the transition probability (or probability density) function, where $P(s'|s, a)$ is the probability of transitioning from state s to s' immediately given the agent takes action a in state s.

- $\gamma \in [0, 1)$ denotes the discount factor which determines the present value of future rewards.

- $\rho^0 : \mathcal{S} \to \mathbb{R}_0$ denotes the initial state distribution, where $d(s)$ is the probability (or probability density), the agent begins acting in the environment at state s.

The agent acts in the environment by following a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_0$, which is a set of probability distributions over actions for each state. The agent picks an action at state s by drawing samples from $\pi(\cdot|s)$.

The agent and environment interact at each of a sequence of discrete time steps $t \in \mathbb{N}_0$. At $t = 0$ the initial state is drawn from ρ^0. At each time step t, the agent receives a representation of the environment's state $s_t \in \mathcal{S}$, then the agent selects an action $a_t \sim \pi(\cdot|s_t)$, receives the reward $r(s_t, a_t)$, then moves to state $s_{t+1} \sim P(s|s_t, a_t)$. By construction neither r_t nor s_{t+1} depends on $((s_0, a_0), \dots (s_{t-1}, a_{t-1}))$ given (s_t, a_t), therefore they satisfy the *Markov property.* The agent–environment interaction in a Markov decision process, generates a record $\tau = \langle r_t, s_t, a_t \rangle_{t \in \mathbb{N}_0}$ called the *trajectory.*

We now define value functions, for general policies. For a fixed policy π and starting state $s_0 = s$, the *state-value function* $V_\pi : \mathcal{S} \to \mathbb{R}$ is defined as the discounted sum of future rewards

$$V_\pi(s) = \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s \right]$$

where the expectation is with respect to the randomness of the trajectory, that is, the randomness in state transitions and the stochasticity of π.

Similarly, the *action-value* function $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined as

$$Q_\pi(s, a) = \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a \right]$$

The *expected return* $\eta : \Pi \to \mathbb{R}$ for a fixed policy π and initial state distribution ρ^0 is defined as

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

The goal of the agent is to find a policy π that maximizes the *expected return*, i.e. the optimization problem the agent seeks to solve is:

$$\max_\pi \eta(\pi)$$

2.2 Policy Gradient Methods

In this section, we consider methods that learn a parameterized policy that can select actions without consulting a value function. We use the notation θ for the policy parameter vector and π_θ for our parameterized policy. We consider methods for learning the policy parameter based on the gradient of the scalar performance measure $\eta(\pi_\theta)$ with respect to the policy parameter θ. The policy gradient algorithm works by updating policy parameters via stochastic gradient ascent on policy performance:

$$\theta_{k+1} \leftarrow \theta_k + \widehat{\nabla\eta(\pi_{\theta_k})}$$

where $\widehat{\nabla\eta(\pi_{\theta_k})}$ is a stochastic gradient estimate whose expectation is the true gradient $\nabla\eta(\pi_{\theta_k})$. That is,

$$\mathbb{E}_\tau \widehat{\nabla\eta(\pi_{\theta_k})} = \nabla\eta(\pi_{\theta_k})$$

Theorem 1 allows us to construct such Monte Carlo estimates for the gradient of the expected return.

Before stating Theorem 1 we assume that the policy π_θ, together with the transition probability kernel P, induce a marginal distribution over states, for every time step $t \in \mathbb{N}_0$, denoted ρ_θ^t. Clearly ρ_θ^t depends on θ through π_θ for $t > 1$ but does not for $t = 0$, hence we use maintain the notation ρ^0 from above. We define an improper marginal distribution over states $\rho_\theta = \sum_{t=0}^{\infty} \gamma^t \rho_\theta^t$. We also use $\mathbf{P}_{s,s'}^n$ to denote the probability of transitioning from state s to state s' in n steps.

Theorem 1. *(Stochastic Policy Gradient (Sutton et al., 1999)). For discrete state space $\mathcal{S}$ and action space $\mathcal{A}$,*

$$\begin{aligned}\nabla_\theta\eta(\pi_\theta) &= \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \rho_\theta(s) Q_\theta(s,a) \nabla_\theta \pi_\theta(a|s) \\ &= \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\right]\end{aligned}$$

Proof. Consider the gradient of the value function for a particular state $s \in \mathcal{S}$,

$$\begin{aligned}\nabla_\theta V_\theta(s) &= \nabla_\theta \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q_\theta(s,a) \\ &= \sum_{a\in\mathcal{A}} \left[\nabla_\theta \pi_\theta(a|s) Q_\theta(s,a) + \pi_\theta(a|s) \nabla_\theta Q_\theta(s,a)\right] \\ &= \sum_{a\in\mathcal{A}} \left[\nabla_\theta \pi_\theta(a|s) Q_\theta(s,a) + \pi_\theta(a|s) \nabla_\theta \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) V_\theta(s')\right)\right] \\ &= \sum_{a\in\mathcal{A}} \left[\nabla_\theta \pi_\theta(a|s) Q_\theta(s,a) + \gamma \pi_\theta(a|s) \nabla_\theta \sum_{s'\in\mathcal{S}} P(s'|s,a) V_\theta(s')\right]\end{aligned}$$

$$= \sum_{a\in\mathcal{A}} \left[\sum_{x\in\mathcal{S}} \mathbf{P}^0_{s,x} \nabla_\theta \pi_\theta(a|s) Q_\theta(x,a) + \gamma \pi_\theta(a|s) \nabla_\theta \sum_{s'\in\mathcal{S}} P(s'|s,a) V_\theta(s') \right]$$
$$= \sum_{x\in\mathcal{S}} \sum_{a\in\mathcal{A}} \mathbf{P}^0_{s,x} \nabla_\theta \pi_\theta(a|s) Q_\theta(x,a) + \gamma \sum_{s'\in\mathcal{S}} \sum_{a\in\mathcal{A}} \pi_\theta(a|s) P(s'|s,a) \nabla_\theta V_\theta(s')$$
$$= \sum_{x\in\mathcal{S}} \sum_{a\in\mathcal{A}} \mathbf{P}^0_{s,x} \nabla_\theta \pi_\theta(a|s) Q_\theta(x,a) + \gamma \sum_{s'\in\mathcal{S}} \mathbf{P}^1_{s,s'} \nabla_\theta V_\theta(s')$$
$$= \sum_{s'\in\mathcal{S}} \left[\mathbf{P}^0_{s,s'} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q_\theta(s',a) + \gamma \mathbf{P}^1_{s,s'} \nabla_\theta V_\theta(s') \right]$$

Now, unravelling the recursion, in $\nabla_\theta V_\theta(s)$,

$$= \sum_{s'\in\mathcal{S}} \mathbf{P}^0_{s,s'} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q_\theta(s',a)$$
$$+ \sum_{s'\in\mathcal{S}} \gamma \mathbf{P}^1_{s,s'} \sum_{s''\in\mathcal{S}} \left[\mathbf{P}^0_{s',s''} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s'') Q_\theta(s'',a) + \gamma \mathbf{P}^1_{s',s''} \nabla_\theta V_\theta(s'')(s') \right]$$
$$= \sum_{s'\in\mathcal{S}} \mathbf{P}^0_{s,s'} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q_\theta(s',a) + \gamma \sum_{s'\in\mathcal{S}} \sum_{s''\in\mathcal{S}} \mathbf{P}^0_{s',s''} \mathbf{P}^1_{s,s'} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s'') Q_\theta(s'',a)$$
$$+ \gamma^2 \sum_{s'\in\mathcal{S}} \sum_{s''\in\mathcal{S}} \mathbf{P}^1_{s',s''} \mathbf{P}^1_{s,s'} \nabla_\theta V_\theta(s'')$$
$$= \sum_{s'\in\mathcal{S}} \mathbf{P}^0_{s,s'} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q_\theta(s',a) + \gamma \sum_{s''\in\mathcal{S}} \left(\sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s'') Q_\theta(s'',a) \right) \sum_{s'\in\mathcal{S}} \mathbf{P}^0_{s',s''} \mathbf{P}^1_{s,s'}$$
$$+ \gamma^2 \sum_{s'\in\mathcal{S}} \sum_{s''\in\mathcal{S}} \mathbf{P}^1_{s',s''} \mathbf{P}^1_{s,s'} \nabla_\theta V_\theta(s'')$$
$$= \sum_{s'\in\mathcal{S}} \mathbf{P}^0_{s,s'} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s') Q_\theta(s',a) + \gamma \sum_{s''\in\mathcal{S}} \mathbf{P}^1_{s,s''} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s'') Q_\theta(s'',a)$$
$$+ \gamma^2 \sum_{s''\in\mathcal{S}} \mathbf{P}^2_{s,s''} \nabla_\theta V_\theta(s'')$$
$$= \sum_{s'\in\mathcal{S}} \left(\mathbf{P}^0_{s,s'} + \gamma \mathbf{P}^1_{s,s''} \right) \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s') Q_\theta(s',a) + \gamma^2 \sum_{s''\in\mathcal{S}} \mathbf{P}^2_{s,s''} \nabla_\theta V_\theta(s'')$$

From here, the pattern is clear and the second summand converges to zero, hence,

$$= \sum_{s'\in\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t \mathbf{P}^n_{s,s'} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s') Q_\theta(s',a)$$

Given the initial state distribution ρ^0 and by the law of total probability $\rho^t_\theta(s') = \sum_{s\in\mathcal{S}} \rho^0(s) \mathbf{P}^t_{s,s'}$, we combine the above results to obtain the gradient of the expected return,

$$\begin{aligned}
\nabla_\theta \eta(\pi_\theta) &= \nabla_\theta \sum_{s\in\mathcal{S}} \rho^0(s) V_\theta(s) = \sum_{s\in\mathcal{S}} \rho^0(s) \nabla_\theta V_\theta(s) \\
&= \sum_{s\in\mathcal{S}} \rho^0(s) \left[\sum_{s'\in\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t \mathbf{P}^t_{s,s'} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s') Q_\theta(s',a) \right] \\
&= \sum_{s'\in\mathcal{S}} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s') Q_\theta(s',a) \left[\sum_{t=0}^{\infty} \gamma^t \rho^0(s) \mathbf{P}^t_{s,s'} \right] \\
&= \sum_{s'\in\mathcal{S}} \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s') Q_\theta(s',a) \left[\sum_{t=0}^{\infty} \gamma^t \rho^t_\theta(s') \right] \\
&= \sum_{s'\in\mathcal{S}} \rho_\theta(s') \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s') Q_\theta(s',a)
\end{aligned}$$

which proves the first equality in the theorem. The second equality namely,

$$\sum_{s'\in\mathcal{S}} \rho_\theta(s') \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s') Q_\theta(s',a) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta} \big[\nabla_\theta \log \pi_\theta(a|s) Q(s,a) \big]$$

is immediate considering, $\pi_\theta(a|s)\nabla_\theta \log \pi_\theta(a|s) = \nabla_\theta \pi_\theta(a|s)$. $\square$

To form Monte Carlo estimates of $\nabla_\theta \eta(\pi_\theta)$ from Theorem 1, we need to sample states from the discounted state distribution $\rho_\theta = \sum_{t=0}^{\infty} \gamma^t \rho^t_\theta(s)$. Since we assume access to sampling entire trajectories using computer simulations, it is straightforward to see how states can be sampled from the discounted state distribution using rejection sampling. We provide the rejection sampler in Algorithm 1.

Algorithm 1: Rejection sampler for $s \sim \rho_\theta$

```
Sample $s_0 \sim \rho^0$ and $a_0 \sim \pi_\theta$.
for $t = 1, 2, \dots$ do
    Sample $u \sim \mathcal{U}_{[0,1]}$.
    if $u < (1-\gamma)$ then
        return $s_t$
    else
        Sample $a_{t+1} \sim \pi(\cdot|s_t)$ and receive the sample $s_{t+1}$ from the
          environment $\mathcal{E}$.
    end
end
```

Although rejection sampling is easy, it is sample inefficient, i.e., it is computationally inefficient with regard to utilizing all the data sampled over the trajectories. Moreover, for $\gamma \approx 1$, (which is common practice for episodic tasks) the sampler only selects a single state even though approximately full trajectories were generated.

Therefore instead of directly using Theorem 1 to form estimators, we will turn our attention to an alternate expression for the gradient of the expected return presented in Lemma 1, which permits using Monte Carlo estimates for $\nabla_\theta \eta(\pi_\theta)$ formed through sampling entire trajectories, rather than sampling from the discounted state distribution.

Lemma 1. *For discrete state space $\mathcal{S}$ and action space $\mathcal{A}$,*

$$\nabla_\theta \eta(\pi_\theta) = \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) Q(s_t, a_t) \right]$$

where expectation is with respect to the randomness of the trajectory τ, that is, the randomness in state transitions and the stochasticity of π.

Proof. Using the first equality of Theorem 1,

$$\begin{aligned}
\nabla_\theta \eta(\pi_\theta) &= \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \rho_\theta(s) Q_\theta(s,a) \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) \\
&= \sum_{s \in \mathcal{S}} \sum_{t=0}^{\infty} \rho_\theta^t(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) \gamma^t Q_\theta(s,a) \nabla_\theta \log \pi_\theta(a|s) \\
&= \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \rho^0(s) \pi(a|s) \gamma^0 Q_\theta(s,a) \nabla_\theta \log \pi_\theta(a|s) \\
&+ \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} \left(\sum_{s \in \mathcal{S}} \rho^0(s) \sum_{a \in \mathcal{A}} \pi(a|s) P(s'|s,a) \right) \gamma^1 Q_\theta(s,a) \nabla_\theta \log \pi_\theta(a|s) \\
&+ \dots \\
&= \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) Q(s_t, a_t) \right]
\end{aligned}$$

where the third equality follows unravelling the infinite sum over $t \in \mathbb{N}_0$, and the last equality from the definition of expectation over trajectories.

□

The following Lemma applies control variate subtraction to Lemma 1, . This technique reduces the variance of the policy gradient expression without introducing bias by subtracting off a baseline $b(s_t)$ from $Q(s_t, a_t)$ at each time-step t. For our purposes, we will require a slightly more general Lemma than what is used in the literature, namely we require the baseline functions to vary over time.

Lemma 2. *(Time Varying Baselines). For a given set of baselines $b_1, b_2, \dots$ where each b_t depends only on s_t, the following holds*

$$\nabla_\theta \eta(\pi_\theta) = \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) \left(Q(s_t, a_t) - b_t(s_t) \right) \right]$$

where expectation is with respect to the randomness of the trajectory τ.

Proof. We show that the contribution from subtracting a baseline at each time step is zero.

$$\begin{aligned}
&\mathbb{E}_\tau\left[\sum_{t=0}^{\infty}\gamma^t\nabla_\theta\log\pi_\theta(a_t|s_t)b_t(s_t)\right]\\
&=\sum_{s\in\mathcal{S}}\sum_{t=0}^{\infty}\gamma^t\mathbb{P}(s_t=s|\pi,\rho^0)\sum_{a_t\in\mathcal{A}}\pi_\theta(a_t|s_t)\nabla_\theta\log\pi_\theta(a_t|s_t)b_t(s_t)\\
&=\sum_{s\in\mathcal{S}}\sum_{t=0}^{\infty}\gamma^t\mathbb{P}(s_t=s|\pi,\rho^0)\sum_{a_t\in\mathcal{A}}\nabla_\theta\log\pi_\theta(a_t|s_t)b_t(s_t)\\
&=\sum_{s\in\mathcal{S}}\sum_{t=0}^{\infty}\gamma^t\mathbb{P}(s_t=s|\pi,\rho^0)b_t(s_t)\sum_{a_t\in\mathcal{A}}\pi_\theta(a_t|s_t)\nabla_\theta\log\pi_\theta(a_t|s_t)\\
&=\sum_{s\in\mathcal{S}}\sum_{t=0}^{\infty}\gamma^t\mathbb{P}(s_t=s|\pi,\rho^0)b_t(s_t)\sum_{a_t\in\mathcal{A}}\nabla_\theta\pi_\theta(a_t|s_t)\\
&=\sum_{s\in\mathcal{S}}\sum_{t=0}^{\infty}\gamma^t\mathbb{P}(s_t=s|\pi,\rho^0)b_t(s_t)\nabla_\theta\sum_{a_t\in\mathcal{A}}\pi_\theta(a_t|s_t)\\
&=\sum_{s\in\mathcal{S}}\sum_{t=0}^{\infty}\gamma^t\mathbb{P}(s_t=s|\pi,\rho^0)b_t(s_t)\nabla_\theta 1=0
\end{aligned}$$

the result follows from Lemma 1. □

With the groundwork laid, we can introduce the following definition,

Definition 1. *(Policy Gradient Estimator) For a given set of baselines $b_1,\ldots,b_T$ and $\lambda\in(0,1)$ and we define the policy gradient estimator to be*

$$g^T(b,\lambda)=\sum_{t=0}^{T}\lambda^t\nabla_\theta\log\pi_\theta(a_t|s_t)\left(Q(s_t,a_t)-b_t(s_t)\right)$$

Note that $\lim_{T\to\infty}g^T(b,\gamma)$ is an unbiased estimator of the policy gradient as shown in Lemma 2. However the estimator $g^T(b,\lambda)$ is often used in practice with finite T and $\lambda=1$ even when $\gamma\neq1$ as shown in the next section.

2.3 Vanilla Policy Gradient Algorithm

The vanilla policy gradient (VPG) algorithm performs approximate stochastic gradient ascent (SGA) directly on the expected return $\nabla_\theta \eta(\pi_\theta)$ to optimize for the policy parameters θ. The approximation stems from the fact that the horizon can be infinite and $\lambda \neq \gamma$. It is worth noting that this is exact SGA whenever $\lambda = \gamma$ and the horizon T is known.

```
Algorithm 2: Vanilla Policy Gradient
  Input: number of episodes $N$, buffer size $B$, batch size $T$, a differentiable
         policy parameterization $\pi_\theta(a|s)$, oracle action value $Q_\pi(s,a)$,
         baseline $b_t(\cdot)$ and parameter $\lambda \in (0,1)$
1 for episode $k = 1, 2, \ldots, N$ do
2     Collect $B$ trajectories in $\mathcal{D}_k = \{\tau_1, \ldots, \tau_B\}$ by running policy $\pi_{\theta_k}$ in the
      environment
3     Estimate the policy gradient as
          $$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \lambda^t \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \left(Q_{\theta_k}(s_t, a_t) - b_t(s_t)\right)$$
4     Compute policy update
          $$\theta_{k+1} = \theta_k + \alpha \hat{g}_k$$
5 end
```

Note that in step 2, the trajectories $\{\tau_1, \ldots, \tau_B\}$ are collected i.i.d. Due to absence of correlation between the trajectories,

$$\begin{aligned} &Var\left(\frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j} \left(Q(s_t, a_t) - b_t(s_t)\right)\right) \\ &= \frac{1}{|\mathcal{D}_k|} Var\left(\sum_{t=0}^{T} \lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j} \left(Q(s_t, a_t) - b_t(s_t)\right)\right) \end{aligned}$$

so we will be interested in analysing the latter variance term in the following section.

3 Variance Reduction of the Policy Gradient Estimator

In this section, we define the variance of the policy gradient estimator and show that the set of baselines that minimize the variance can be derived in closed form. In this work, we study total variance, which is the sum of variances of all components $g_j^T(b,\lambda)$ of a vector estimator $g^T(b,\lambda)$.

Consider the variance of the component $g_j^T(b,\lambda)$

$$\begin{aligned}\mathrm{Var}(g_j^T(b,\lambda)) &= \mathrm{Var}\left(\sum_{t=0}^{T}\lambda^t\frac{\partial\log\pi_\theta(a_t|s_t)}{\partial\theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right)\right)\\ &= \sum_{t=0}^{T}\mathrm{Var}\left(\lambda^t\frac{\partial\log\pi_\theta(a_t|s_t)}{\partial\theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right)\right)\\ &\quad+2\sum_{t<t'}\mathrm{Cov}\left(\lambda^t\frac{\partial\log\pi_\theta(a_t|s_t)}{\partial\theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right),\right.\\ &\quad\left.\lambda^{t'}\frac{\partial\log\pi_\theta(a_{t'}|s_{t'})}{\partial\theta_j}\left(Q(s_{t'},a_{t'})-b_{t'}(s_{t'})\right)\right)\end{aligned}$$

The crucial difference between the existing approaches and ours is that we explicitly find the baselines that minimize $\sum_j\mathrm{Var}(g_j^T(b,\lambda))$ whereas Greensmith et al., 2004; Wu et al., 2018 minimize only

$$\sum_j\mathrm{Var}\left(\lambda^t\frac{\partial\log\pi_\theta(a_t|s_t)}{\partial\theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right)\right)$$

for each t due which is equivalent to minimizing

$$\sum_j\sum_{t=0}^{T}\mathrm{Var}\left(\lambda^t\frac{\partial\log\pi_\theta(a_t|s_t)}{\partial\theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right)\right)$$

as in Kuba et al., 2021, due to the assumption of uncorrelated states over time by sampling each state independently from the discounted state distribution. Note that both of these approaches do not consider the covariance terms in the objective, due to their sampling assumptions. We show that accounting for these terms leads to a markedly different optimal baseline.

3.1 Deriving the Optimal Baseline

In this section, we state and derive the optimal state-dependent baseline. Here by optimal baselines, we mean the set of baselines that globally minimize the total variance of the policy gradient estimator.

Theorem 2. *The optimal baseline for the policy gradient estimator $g^T(b,\lambda)$ is given by*

$$b_t^*(s_t)=\frac{\mathbb{E}_{s_{t+1:T},a_{t:T}}\left[\sum_{t'=t}^{T}\lambda^{t'-t}\langle\nabla_\theta\log\pi(a_t|s_t),\nabla_\theta\log\pi(a_{t'}|s_{t'})\rangle Q(s_{t'},a_{t'})\right]}{\mathbb{E}_{a_t\sim\pi}\langle\nabla_\theta\log\pi(a_t|s_t),\nabla_\theta\log\pi(a_t|s_t)\rangle}$$

where the expectation in the numerator is over the stochasticity of the future trajectory $(a_t, s_{t+1}, a_{t+1}, \dots, s_T, a_T)$ conditioned on s_t.

Proof. We are required to find the baselines that globally minimize the total variance of the policy gradient estimator $\sum_j \operatorname{Var}(g_j^T(b,\lambda))$. We do this by massaging $\sum_j \operatorname{Var}(g_j^T(b,\lambda))$ into an equivalent objective whose global minima are the same as that of $\sum_j \operatorname{Var}(g_j^T(b,\lambda))$. We will show that the equivalent objective is convex by proving that its hessian with respect to the baselines is positive semi-definite. Finally, we solve for a locally optimal solution, which will be globally optimal due to convexity.

First we decompose the variance of $g_j^T(b,\lambda)$, into a sum of variance terms and covariance terms

$$\begin{aligned}\operatorname{Var}(g_j^T(b,\lambda)) = &\sum_{t=0}^{T} \operatorname{Var}\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t) - b_t(s_t)\right)\right) \\ &+ 2\sum_{t<t'} \operatorname{Cov}\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t) - b_t(s_t)\right),\right. \\ &\left.\lambda^{t'} \frac{\partial \log \pi_\theta(a_{t'}|s_{t'})}{\partial \theta_j}\left(Q(s_{t'},a_{t'}) - b_{t'}(s_{t'})\right)\right)\end{aligned}$$

The t-th summand in the sum of variances can be written as,

$$\begin{aligned}&\operatorname{Var}\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t) - b_t(s_t)\right)\right) \\ &= \mathbb{E}\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t) - b_t(s_t)\right)\right)^2 \\ &- \left[\mathbb{E}\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t) - b_t(s_t)\right)\right)\right]^2\end{aligned}$$

Expanding the first summand,

$$\begin{aligned}&\mathbb{E}\left(\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\right)^2\left(Q(s_t,a_t)^2 + b_t(s_t)^2 - 2Q(s_t,a_t)b_t(s_t)\right)\right) \\ &= \mathbb{E}\left(\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\right)^2\left(b_t(s_t)^2 - 2Q(s_t,a_t)b_t(s_t)\right)\right) + \mathit{const}\end{aligned}$$

And for the second summand, since b_t is a baseline, its characterizing property renders

$$\begin{aligned}&\left[\mathbb{E}\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t) - b_t(s_t)\right)\right)\right] \\ &= \left[\mathbb{E}\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j} Q(s_t,a_t)\right)\right] = \mathit{const}\end{aligned}$$

So we may consider the contribution from the t-th variance term to be

$$\begin{aligned}
&\operatorname{Var}\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right)\right)\\
&=\mathbb{E}\left(\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\right)^2\left(b_t(s_t)^2-2Q(s_t,a_t)b_t(s_t)\right)\right)+const
\end{aligned}$$

Similarly for $t < t'$ the covariance term can be written as

$$\begin{aligned}
&\operatorname{Cov}\left(\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right),\right.\\
&\left.\lambda^{t'} \frac{\partial \log \pi_\theta(a_{t'}|s_{t'})}{\partial \theta_j}\left(Q(s_{t'},a_{t'})-b_{t'}(s_{t'})\right)\right)\\
&=\mathbb{E}\left[\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right)\right.\\
&\left.\cdot\lambda^{t'} \frac{\partial \log \pi_\theta(a_{t'}|s_{t'})}{\partial \theta_j}\left(Q(s_{t'},a_{t'})-b_{t'}(s_{t'})\right)\right]\\
&-\mathbb{E}\left[\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right)\right]\\
&\cdot\mathbb{E}\left[\lambda^{t'} \frac{\partial \log \pi_\theta(a_{t'}|s_{t'})}{\partial \theta_j}\left(Q(s_{t'},a_{t'})-b_{t'}(s_{t'})\right)\right]
\end{aligned}$$

For the first summand, we notice the key step for the proof,

$$\begin{aligned}
&\mathbb{E}\left[\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right)\right.\\
&\left.\cdot\lambda^{t'} \frac{\partial \log \pi_\theta(a_{t'}|s_{t'})}{\partial \theta_j}\left(Q(s_{t'},a_{t'})-b_{t'}(s_{t'})\right)\right]\\
&=\mathbb{E}\left[\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right)\right.\\
&\left.\cdot\lambda^{t'} \frac{\partial \log \pi_\theta(a_{t'}|s_{t'})}{\partial \theta_j}Q(s_{t'},a_{t'})\right]
\end{aligned}$$

which holds since for $t < t'$ the characteristic property of the baseline can be used for $b_{t'}$, rendering

$$\begin{aligned}
&\mathbb{E}\left[\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\left(Q(s_t,a_t)-b_t(s_t)\right)\right.\\
&\left.\cdot\lambda^{t'} \frac{\partial \log \pi_\theta(a_{t'}|s_{t'})}{\partial \theta_j}b_t(s_{t'})\right]=0
\end{aligned}$$

Therefore the contribution from the first summand of the covariance term is

$$-\mathbb{E}\left[\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j}\lambda^{t'} \frac{\partial \log \pi_\theta(a_{t'}|s_{t'})}{\partial \theta_j}b_t(s_t)Q(s_{t'},a_{t'})\right]+const$$

For the second summand, Using the characterizing property of baselines and noticing the relevant constants,

$$\begin{aligned}
&\mathbb{E}\left[\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j} \left(Q(s_t,a_t) - b_t(s_t)\right)\right] \\
&\cdot \mathbb{E}\left[\lambda^{t'} \frac{\partial \log \pi_\theta(a_{t'}|s_{t'})}{\partial \theta_j} \left(Q(s_{t'},a_{t'}) - b_{t'}(s_{t'})\right)\right] \\
&= \mathbb{E}\left[\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j} Q(s_t,a_t)\right] \\
&\cdot \mathbb{E}\left[\lambda^{t'} \frac{\partial \log \pi_\theta(a_{t'}|s_{t'})}{\partial \theta_j} Q(s_{t'},a_{t'})\right] = const
\end{aligned}$$

Therefore the contribution from twice the covariance term is

$$-2\mathbb{E}\left[\lambda^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta_j} \lambda^{t'} \frac{\partial \log \pi_\theta(a_{t'}|s_{t'})}{\partial \theta_j} b_t(s_t) Q(s_{t'},a_{t'})\right] + const$$

Therefore minimizing the total variance is equivalent to minimizing

$$\begin{aligned}
&V_T(b,\lambda) \triangleq \sum_{t=0}^{T} \mathbb{E}\Big[\lambda^{2t} \langle \nabla_\theta \log \pi(a_t|s_t), \nabla_\theta \log \pi(a_t|s_t)\rangle \\
&\cdot \big(b_t(s_t)^2 - 2Q(s_t,a_t)b_t(s_t)\big)\Big] \\
&- 2\sum_{t<t'} \mathbb{E}\left[\lambda^{t+t'} \langle \nabla_\theta \log \pi(a_t|s_t), \nabla_\theta \log \pi(a_{t'}|s_{t'})\rangle b_t(s_t) Q(s_{t'},a_{t'})\right]
\end{aligned}$$

We will consider this as a minimization problem with $T|\mathcal{S}|$ parameters, $b_t(s)$ for $t = 1, \dots, T$ and $s \in \mathcal{S}$.

Differentiating $V_T(b,\lambda)$ with respect to $b_t(s)$ (for a particular time-step t and state s),

$$\frac{\partial V_T(b,\lambda)}{\partial b_t(s)} = \mathbb{E}_{s_{0:t-1}a_{0:t-1}}\Bigg[2P(s|s_{t-1},a_{t-1})\mathbb{E}_{a_t\sim\pi}\Big(\lambda^{2t}(b_t(s)-Q(s,a_t))$$
$$\langle\nabla_\theta\log\pi(a_t|s),\nabla_\theta\log\pi(a_t|s)\rangle\Big)$$
$$-\sum_{t'>t}\mathbb{E}_{s_{t'},a_{t'}|s_t=s}\Big(\lambda^{t+t'}Q(s_{t'},a_{t'})\langle\nabla_\theta\log\pi(a_t|s_t),\nabla_\theta\log\pi(a_{t'}|s_{t'})\rangle\Big)\Bigg]$$
$$=\mathbb{E}_{s_{0:t-1}a_{0:t-1}}\Bigg[2P(s|s_{t-1},a_{t-1})\lambda^{2t}b_t(s)\mathbb{E}_{a_t\sim\pi}\Big(\langle\nabla_\theta\log\pi(a_t|s),$$
$$\nabla_\theta\log\pi(a_t|s)\rangle\Big)-\sum_{t'=t}^{T}\mathbb{E}_{s_{t'},a_{t'}|s_t=s}\Big(\lambda^{t+t'}Q(s_{t'},a_{t'})\langle\nabla_\theta\log\pi(a_t|s_t),$$
$$\nabla_\theta\log\pi(a_{t'}|s_{t'})\rangle\Big)\Bigg]$$

Note that $V_T(b,\lambda)$ is convex with respect to it's parameters $b_t(s)$ since

$$\frac{\partial^2 V_T(b,\lambda)}{\partial b_t(s)^2} = \mathbb{E}_{s_{0:t-1}a_{0:t-1}}\Bigg[2P(s|s_{t-1},a_{t-1})\mathbb{E}_{a_t\sim\pi}\Big(\lambda^{2t}\langle\nabla_\theta\log\pi(a_t|s_t),$$
$$\nabla_\theta\log\pi(a_t|s_t)\rangle\Big)\geq 0$$

and the cross partial derivatives for $t'\neq t$ or $s'\neq s$ satisfies,

$$\frac{\partial^2 V_T(b,\lambda)}{\partial b_t(s)\partial b_{t'}(s')}=0$$

which implies the hessian matrix $\mathbf{H}_V$ is a diagonal matrix with non-negative entries in the diagonal. Hence $\mathbf{H}_V$ is positive semi-definite which implies convexity of $V_T(b,\lambda)$ with respect to its parameters $b_t(s)$.
Now, we may derive a point of local minimum by setting $\frac{\partial V_T(b,\lambda)}{\partial b_t(s)}=0$. Global optimally follows from convexity.

$$b_t^*(s)=\frac{\sum_{t'=t}^{T}\lambda^{t'-t}\mathbb{E}_{s_{t'},a_{t'}|s_t=s}\Big[\langle\nabla_\theta\log\pi(a_t|s),\nabla_\theta\log\pi(a_{t'}|s_{t'})\rangle Q(s_{t'},a_{t'})\Big]}{\mathbb{E}_{a_t\sim\pi}\langle\nabla_\theta\log\pi(a_t|s),\nabla_\theta\log\pi(a_t|s)\rangle}$$
$$=\frac{\mathbb{E}_{s_{t+1:T},a_{t:T}}\Big[\sum_{t'=t}^{T}\lambda^{t'-t}\langle\nabla_\theta\log\pi(a_t|s),\nabla_\theta\log\pi(a_{t'}|s_{t'})\rangle Q(s_{t'},a_{t'})\Big]}{\mathbb{E}_{a_t\sim\pi}\langle\nabla_\theta\log\pi(a_t|s),\nabla_\theta\log\pi(a_t|s)\rangle}$$

for each $t=1,\dots,T$ and $s\in\mathcal{S}$. □

Now, we have laid the groundwork for comparing the different optimal baselines stated in the literature. We analyse and compare the baselines in the following section.

3.2 Comparison of Baselines

In the beginning of Section 3 we discussed the differences between the existing approaches and ours with regards to deriving the optimal baseline. We now state the different baselines and analyse the differences.
In this work, we have proposed,

$$b_t^*(s_t) = \frac{\mathbb{E}_{s_{t+1:T}, a_{t:T}} \left[\sum_{t'=t}^{T} \lambda^{t'-t} \langle \nabla_\theta \log \pi(a_t|s_t), \nabla_\theta \log \pi(a_{t'}|s_{t'}) \rangle Q(s_{t'}, a_{t'}) \right]}{\mathbb{E}_{a_t \sim \pi} \langle \nabla_\theta \log \pi(a_t|s_t), \nabla_\theta \log \pi(a_t|s_t) \rangle}$$

as the optimal baseline, while (Kuba et al., 2021) in Theorem 3 of their paper have proposed,

$$b_t^{\text{uncorr}}(s_t) = \frac{\mathbb{E}_{a_t \sim \pi} \left[\langle \nabla_\theta \log \pi(a_t|s_t), \nabla_\theta \log \pi(a_t|s_t) \rangle Q(s_{t'}, a_{t'}) \right]}{\mathbb{E}_{a_t \sim \pi} \langle \nabla_\theta \log \pi(a_t|s_t), \nabla_\theta \log \pi(a_t|s_t) \rangle}$$

It is worth mentioning that although (Kuba et al., 2021) is in the Multi-Agent Reinforcement Learning setting, the above result follows immediately considering a single agent and the same result for a single agent was derived by (Greensmith et al., 2004). Now, although (Wu et al., 2018) use the same assumptions as (Greensmith et al., 2004; Kuba et al., 2021) in deriving the optimal state-dependent baseline, they have stated a different baseline from $b_t^{\text{uncorr}}(s_t)$ in Equation (4) (Wu et al., 2018), namely,

$$\tilde{b}_t(s_t) = \frac{\mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi} \left[\langle \nabla_\theta \log \pi(a_t|s_t), \nabla_\theta \log \pi(a_t|s_t) \rangle Q(s_{t'}, a_{t'}) \right]}{\mathbb{E}_{a_t \sim \pi} \langle \nabla_\theta \log \pi(a_t|s_t), \nabla_\theta \log \pi(a_t|s_t) \rangle}$$

Unfortunately, their derivation is incorrect, refer to Equations (21), (22) and (23) of Appendix A (Wu et al., 2018).

They set the policy gradient estimator as $g := \nabla_\theta \log \pi_\theta(a_t|s_t) \left(Q(s_t, a_t) - b(s_t) \right)$ and derive the variance of the policy gradient estimator g as

$$\begin{aligned} \text{Var}(g) &= \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi} [(g - \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi}[g])^T (g - \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi}[g])] \\ &= \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi} \left[\nabla_\theta \log \pi_\theta(a_t|s_t)^T \nabla_\theta \log \pi_\theta(a_t|s_t) \right] b(s_t)^2 \\ &- 2\mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi} \left[\nabla_\theta \log \pi_\theta(a_t|s_t)^T \nabla_\theta \log \pi_\theta(a_t|s_t) Q(s_t, a_t) \right] b(s_t) \end{aligned}$$

which is incorrect. The correct sequence of derivations would instead render,

$$\begin{aligned} \text{Var}(g) &= \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi} [(g - \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi}[g])^T (g - \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi}[g])] \\ &= \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi} \left[\nabla_\theta \log \pi_\theta(a_t|s_t)^T \nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)^2 \right] \\ &- 2\mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi} \left[\nabla_\theta \log \pi_\theta(a_t|s_t)^T \nabla_\theta \log \pi_\theta(a_t|s_t) Q(s_t, a_t) b(s_t) \right] \\ &- 2\mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) (Q(s_t, a_t) - b(s_t)) \right] \nabla_\theta \eta(\pi_\theta) + \text{const} \end{aligned}$$

where the constant is with respect to $b(s_t)$. Of particular importance is the missing term $-2\mathbb{E}_{s_t\sim\rho_\theta, a_t\sim\pi}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)(Q(s_t,a_t)-b(s_t))\right]\nabla_\theta\eta(\pi_\theta)$. Noting that $b(s_t)$ is a function of the random variable $s_t\sim\rho_\theta$, and expectation is taken with respect to s_t, another inaccuracy is that $b(s_t)$ and $b(s_t)^2$ cannot be factorized out of the expectation unless it is a constant with respect to s_t, which is not the case. Henceforth we will compare our work to the baseline stated in both $b_t^{\text{uncorr}}(s_t)$ from (Greensmith et al., 2004; Kuba et al., 2021).

Consider the difference between the baselines $b_t^*(s_t)$ and $b_t^{\text{uncorr}}(s_t)$. Noting that $b_t^{\text{uncorr}}(s_t)$ is the simply the first summand of the $T-t+1$ summands of $b_t^*(s_t)$,

$$\begin{aligned}
b_t^*(s_t) - b_t^{\text{uncorr}}(s_t) &= \frac{\mathbb{E}_{s_{t+1:T},a_{t:T}}\left[\sum_{t'=t}^{T}\lambda^{t'-t}\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_{t'}|s_{t'})\rangle Q(s_{t'},a_{t'})\right]}{\mathbb{E}_{a_t\sim\pi}\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_t|s_t)\rangle} \\
&\quad - \frac{\mathbb{E}_{a_t\sim\pi}\left[\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_t|s_t)\rangle Q(s_{t'},a_{t'})\right]}{\mathbb{E}_{a_t\sim\pi}\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_t|s_t)\rangle} \\
&= \frac{\mathbb{E}_{a_t\sim\pi}\left[\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_t|s_t)\rangle Q(s_{t'},a_{t'})\right]}{\mathbb{E}_{a_t\sim\pi}\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_t|s_t)\rangle} \\
&\quad + \frac{\mathbb{E}_{s_{t+1:T},a_{t:T}}\left[\sum_{t'=t+1}^{T}\lambda^{t'-t}\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_{t'}|s_{t'})\rangle Q(s_{t'},a_{t'})\right]}{\mathbb{E}_{a_t\sim\pi}\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_t|s_t)\rangle} \\
&\quad - \frac{\mathbb{E}_{a_t\sim\pi}\left[\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_t|s_t)\rangle Q(s_{t'},a_{t'})\right]}{\mathbb{E}_{a_t\sim\pi}\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_t|s_t)\rangle} \\
&= \frac{\mathbb{E}_{s_{t+1:T},a_{t:T}}\left[\sum_{t'=t+1}^{T}\lambda^{t'-t}\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_{t'}|s_{t'})\rangle Q(s_{t'},a_{t'})\right]}{\mathbb{E}_{a_t\sim\pi}\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_t|s_t)\rangle}
\end{aligned}$$

Of particular importance are the inner products within the numerator's expectation, namely, $\langle\nabla_\theta \log\pi(a_t|s_t), \nabla_\theta \log\pi(a_{t'}|s_{t'})\rangle$ for $t' > t$. During training time, if all of these inner products are zero (or for practical purposes approximately zero), then there is no meaningful difference between the baselines established in the literature and ours. We show in the following section that this is not the case.

4 Experiments

In this Section, we verify that the optimal baseline introduced in this work and the one established in the literature are non-trivially different. Before describing the experiments themselves, we will introduce the environment and the different policies we will use.

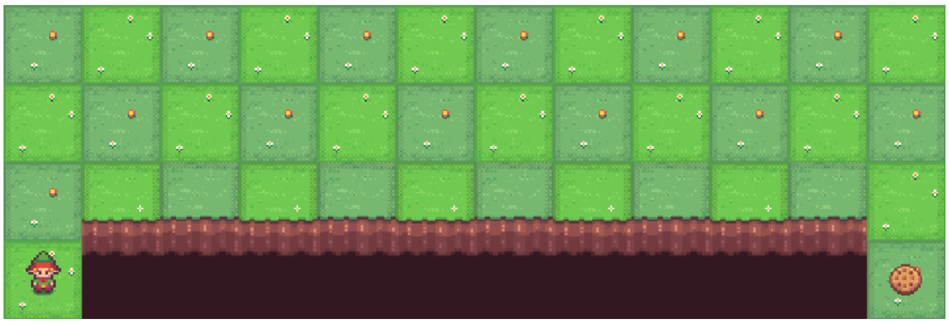

Figure 1: Cliff Walking environment from OpenAI Gym (Brockman et al., 2016).

We consider the Cliff Walking environment from OpenAI Gym (Brockman et al., 2016). This example was first considered in (Sutton & Barto, 2018). Consider the 4×12 grid world shown in Figure 1. It is a standard undiscounted ($\gamma = 1$), episodic task, with the starting state being the square on which the boy is positioned and the goal state where the cookie is positioned. There are four actions causing movement up, down, left and right. The reward is -1 on every transition, except those that transition into the lower rocky region, i.e. "the cliff". Stepping into this region incurs a reward of -100 and sends the agent immediately back to the starting state. It is not hard to see that the optimal policy takes the shortest path between the starting state and the cookie and achieves the optimal return of -13.

For our experiments, we will restrict the environment to a maximum of 100 time-steps. In each experiment our policy π_θ will be parameterized by a fully-connected two-layer linear neural network with a softmax activation function. The policies are trained on the task using the VPG algorithm introduced in Section 2.3. We will consider four different policies for the Cliff Walking environment, differentiated by the number of iterations they were trained for and the expected return $\eta(\pi_\theta)$ they achieve.

Policy	**Number of Training Iterations**	$\eta(\pi_\theta)$
Randomly Initialized	0	-100
Decent	100	-50
Approximately Optimal	150	-15
Optimal	200	-13

Table 1: Policies used in the experiments.

Our experiments consist of verifying whether there is a numerical difference between the optimal baseline in our work $b_t^*(s_t)$ and the one established in the literature $b_t^{\text{uncorr}}(s_t)$, during training time. Recall from Section 3.2,

$$b_t^*(s_t) - b_t^{\text{uncorr}}(s_t) = \frac{\mathbb{E}_{s_{t+1:T},a_{t:T}}\left[\sum_{t'=t+1}^{T} \lambda^{t'-t} \langle \nabla_\theta \log \pi(a_t|s_t), \nabla_\theta \log \pi(a_{t'}|s_{t'})\rangle Q(s_{t'}, a_{t'})\right]}{\mathbb{E}_{a_t\sim\pi}\langle \nabla_\theta \log \pi(a_t|s_t), \nabla_\theta \log \pi(a_t|s_t)\rangle}$$

For the experiments, we set $\lambda = \gamma = 1$.

4.1 Examining the Correlation between Gradients

We first verify that the inner products $\langle \nabla_\theta \log \pi(a_t|s_t), \nabla_\theta \log \pi(a_{t'}|s_{t'})\rangle$ for $t' > t$ are non-zero during training. In particular, we consider the empirical mean $\hat{\mathbb{E}}_\tau \langle \nabla_\theta \log \pi(a_0|s_0), \nabla_\theta \log \pi(a_t|s_t)\rangle$ where the mean is taken with respect to 10 sampled trajectories, for the four different policies. Note that the length of the trajectory is a random variable, therefore, only the trajectories τ that are of length at least t are taken in the empirical mean for $\hat{\mathbb{E}}_\tau \langle \nabla_\theta \log \pi(a_0|s_0), \nabla_\theta \log \pi(a_t|s_t)\rangle$.

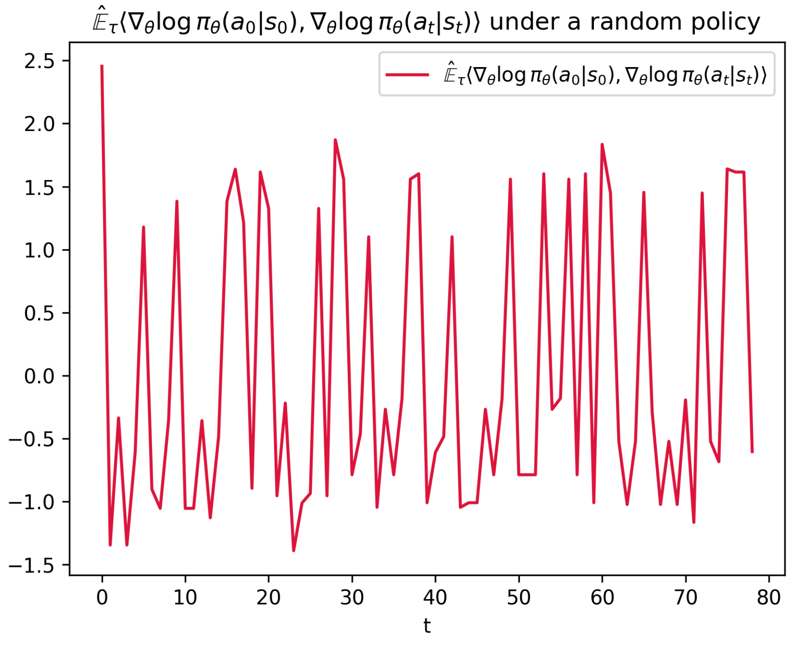

Figure 2: $\hat{\mathbb{E}}_\tau \langle \nabla_\theta \log \pi(a_0|s_0), \nabla_\theta \log \pi(a_t|s_t)\rangle$ under a random policy trained for 0 iterations.

From Figure 2 it is clear that under a random policy, the inner product terms are non-zero. Moreover, they fluctuate throughout the horizon, with mostly positive correlation.

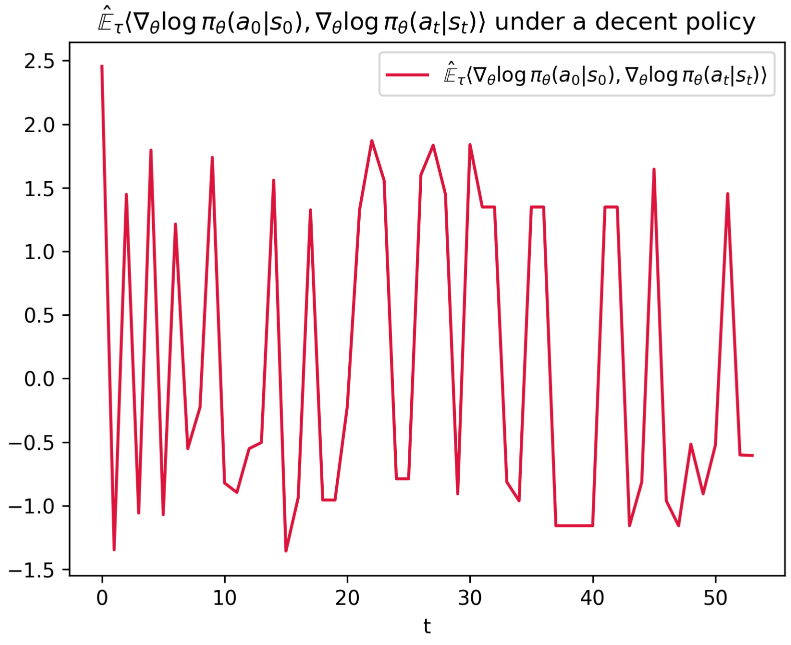

Figure 3: $\hat{\mathbb{E}}_\tau\langle\nabla_\theta \log \pi(a_0|s_0), \nabla_\theta \log \pi(a_t|s_t)\rangle$ under a policy trained for 100 iterations.

Figure 3 exhibits a similar trend to Figure 2, although the 100 training iterations seem to have slightly reduced the fluctuations of the inner products.

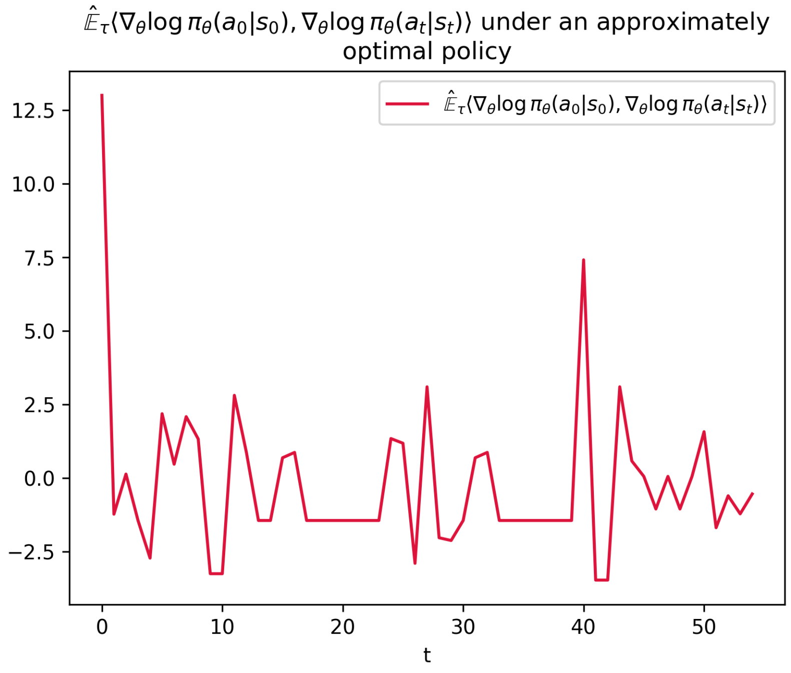

Figure 4: $\hat{\mathbb{E}}_\tau\langle\nabla_\theta \log \pi(a_0|s_0), \nabla_\theta \log \pi(a_t|s_t)\rangle$ under an approximately optimal policy trained for 150 iterations.

Figure 4 reveals a completely different trend, except for the initial spike at $t = 0$.

The 150 training iterations seem to have greatly reduced the fluctuations of the inner products, and they instead stay close to zero for $t > 1$ with occasional deviations.

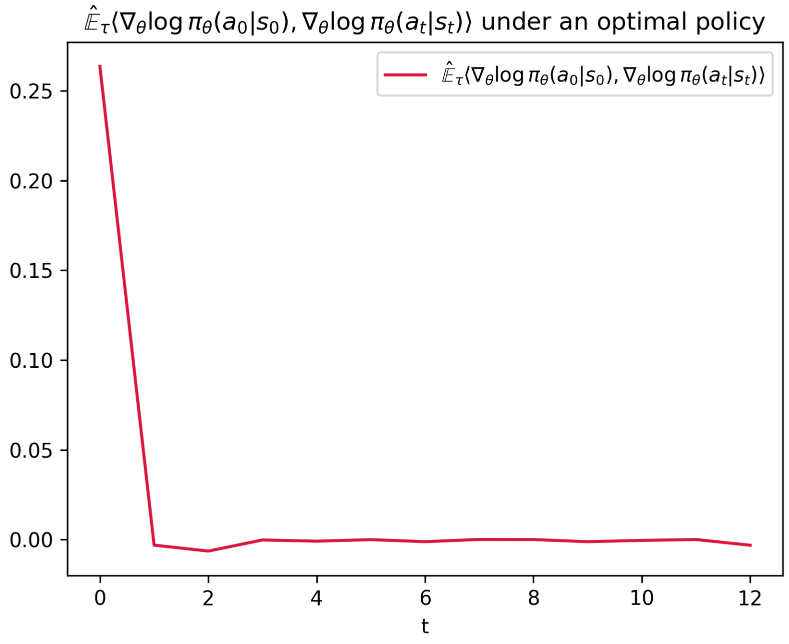

Figure 5: $\hat{\mathbb{E}}_\tau\langle\nabla_\theta \log \pi(a_0|s_0), \nabla_\theta \log \pi(a_t|s_t)\rangle$ under the optimal policy trained for 200 iterations.

Finally in Figure 5, we see a single non-zero value at $t = 0$, while the inner products have converged to zero for $t > 1$. Although the above figures are entirely dependent on the environment and policy parameterization, this verifies that the correlation between gradients is non-zero and hints that the baselines are indeed different under this setup.

4.2 Relative Difference between Baselines

In the final set of experiments, we verify that the baselines are indeed different under this setup. Consider the relative difference,

$$\frac{b_t^*(s_t) - b_t^{\text{uncorr}}(s_t)}{b_t^*(s_t)}.$$

We show that the approximate relative difference is large during training. In particular, we consider the relative difference

$$\frac{b_0^*(s_0) - b_0^{\text{uncorr}}(s_0)}{b_0^*(s_0)}$$

for the four different policies. To approximate the relative difference, we consider the quantities

$$\Delta_1 = \mathbb{E}_{s_{1:T},a_{1:T}}\left[\sum_{t'=1}^{T}\lambda^{t'-1}\langle\nabla_\theta\log\pi(a_0|s_0),\nabla_\theta\log\pi(a_{t'}|s_{t'})\rangle Q(s_{t'},a_{t'})\right]$$

and

$$\Delta_2 = \mathbb{E}_{a_0}\left[\langle\nabla_\theta\log\pi(a_0|s_0),\nabla_\theta\log\pi(a_0|s_0)\rangle Q(s_0,a_0)\right]$$

and approximate the relative difference using the equality,

$$\frac{b_0^*(s_0) - b_0^{\text{uncorr}}(s_0)}{b_0^*(s_0)} = \frac{\Delta_1}{\Delta_1+\Delta_2} = \frac{\hat{\Delta}_1}{\hat{\Delta}_1+\hat{\Delta}_2}$$

where $\hat{\Delta}_i$ is the empirical mean analogue of Δ_i for $i = 1, 2$. The relative differences are displayed in Table 2.

Policy	$\hat{\Delta}_1$	$\hat{\Delta}_2$	**Relative Difference**
Randomly Initialized	−48.65	−114.84	29.76%
Decent	−37.59	−108.80	25.68%
Approximately Optimal	−10.98	−85.56	11.3%
Optimal	0.12	−32.65	−0.36%

Table 2: Relative difference between the baselines $b_0^*(s_0)$ and $b_0^{\text{uncorr}}(s_0)$.

From Table 2, we note that there is a non-trivial relative difference in the baselines until the policy has converged to the optimal one, thereby verifying the phenomena from Section 4.1, that the correlation terms are crucial during training time.

5 Conclusions

In this work, we studied variance reduction of policy gradient methods, using the baseline subtraction technique. We showed that the optimal baseline varies with the sampling distribution for the states. In particular, we showed that the optimal baseline established in the literature, where states are sampled from the discounted state distribution was inconsistent with the sampling strategy used in practice, i.e., sampling states directly from rollouts. We then derived a novel optimal baseline consistent with states sampled through rollouts. We then compared our baseline analytically to the ones from (Greensmith et al., 2004; Kuba et al., 2021; Wu et al., 2018). Finally, we presented experiments which validated that our optimal baseline and the ones in the literature were indeed different.

References

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Shalev-Shwartz, S., Shammah, S., & Shashua, A. (2016). Safe, multi-agent, reinforcement learning for autonomous driving.

Kober, J., Bagnell, J., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*, 1238–1274. https://doi.org/10.1177/0278364913495721

You, X., Li, X., Xu, Y., Feng, H., & Zhao, J. (2019). Toward packet routing with fully-distributed multi-agent deep reinforcement learning. *2019 International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*, 1–8. https://doi.org/10.23919/WiOPT47501.2019.9144110

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489. https://doi.org/10.1038/nature16961

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, *550*, 354–. http://dx.doi.org/10.1038/nature24270

Vinyals, O., Babuschkin, I., Czarnecki, W., Mathieu, M., Dudzik, A., Chung, J., Choi, D., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J., Jaderberg, M., & Silver, D. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, *575*. https://doi.org/10.1038/s41586-019-1724-z

Brown, N., & Sandholm, T. (2019). Superhuman ai for multiplayer poker. *Science*, *365*(6456), 885–890. https://doi.org/10.1126/science.aay2400

Bakhtin, A., Brown, N., Dinan, E., Farina, G., Flaherty, C., Fried, D., Goff, A., Gray, J., Hu, H., Jacob, A. P., Komeili, M., Konath, K., Kwon, M., Lerer, A., Lewis, M., Miller, A. H., Mitts, S., Renduchintala, A., Roller, S., . . . Zijlstra, M. (2022). Human-level play in the game of <i>diplomacy</i> by combining language models with strategic reasoning. *Science*, *0*(0), eade9097. https://doi.org/10.1126/science.ade9097

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, & K. Müller (Eds.), *Advances in neural information processing systems* (Vol. 12). MIT Press. https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf

Ross, S. (2002). *Simulation*. Academic Press. https://books.google.co.uk/books?id=DApvQgAACAAJ

Greensmith, E., Bartlett, P. L., & Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *J. Mach. Learn. Res.*, *5*, 1471–1530.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., & Levine, S. (2017). Q-prop: Sample-efficient policy gradient with an off-policy critic.

Weaver, L., & Tao, N. (2013). The optimal reward baseline for gradient-based reinforcement learning.

Kuba, J. G., Wen, M., Yang, Y., Meng, L., Gu, S., Zhang, H., Mguni, D. H., & Wang, J. (2021). Settling the variance of multi-agent policy gradients. https://doi.org/10.48550/ARXIV.2108.08612

Wu, C., Rajeswaran, A., Duan, Y., Kumar, V., Bayen, A. M., Kakade, S., Mordatch, I., & Abbeel, P. (2018). Variance reduction for policy gradient with action-dependent factorized baselines. https://doi.org/10.48550/ARXIV.1803.07246

Peters, J., & Schaal, S. (2006). Policy gradient methods for robotics. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2219–2225.

Peters, J., & Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*, *21*(4), 682–697.

Rückstieß, T., Felder, M., & Schmidhuber, J. (2008). State-dependent exploration for policy gradient methods. In W. Daelemans, B. Goethals, & K. Morik (Eds.), *Machine learning and knowledge discovery in databases* (pp. 234–249). Springer Berlin Heidelberg.

Jie, T., & Abbeel, P. (2010). On a connection between importance sampling and the likelihood ratio policy gradient. *Advances in Neural Information Processing Systems*, *23*.

Zhao, T., Hachiya, H., Niu, G., & Sugiyama, M. (2011). Analysis and improvement of policy gradient estimation. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, & K. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 24). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2011/file/85d8ce590ad8981ca2c8286f79f59954-Paper.pdf

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym.

A Code

```
CODE FOR THE FIRST SET OF EXPERIMENTS

import torch
import gym
import numpy as np
from collections import namedtuple

def getGrad(w1,b1,w2,b2):
    return (torch.cat((torch.flatten(w1),torch.flatten(b1),
    torch.flatten(w2),torch.flatten(b2))))



def basisVector(i,size):
    a = np.zeros(size)
    a[i] = 1.0
    return a

class PolicyNet(torch.nn.Module):
    def __init__(self, input_size, output_size, hidden_layer_size=64):
        super(PolicyNet, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Linear(input_size, hidden_layer_size)
        self.fc2 = torch.nn.Linear(hidden_layer_size, output_size)
        self.softmax = torch.nn.Softmax(dim=0)

    def forward(self, x):
        x = torch.from_numpy(basisVector(x,self.input_size)).float()
        # this is for cliff-walking ONLY
        return self.softmax(self.fc2(self.fc1(x)))


    def get_action_and_logp(self, x):

        action_prob = self.forward(x)
        m = torch.distributions.Categorical(action_prob)
        action = m.sample()
        logp = m.log_prob(action)
        return action.item(), logp

    def act(self, x):
        action, _ = self.get_action_and_logp(x)
        return action
```

```
def gradInnerProductOverTime(env, num_traj=10, gamma=0.98,
            policy_learning_rate=0.01, policy_saved_path='xyz.pt'):

    input_size = 4*12 # This is for cliff-walking ONLY
    output_size = env.action_space.n
   Trajectory = namedtuple('Trajectory', 'states actions rewards dones logp')

    def collect_trajectory():
        state_list = []
        action_list = []
        reward_list = []
        dones_list = []
        logp_list = []
        state = env.reset()[0]
        done = False
        truncated = False
        steps = 0

        while (not (done or truncated)) and steps <= 100:
            action, logp = policy.get_action_and_logp(state)
            newstate, reward, done, truncated, info = env.step(action)

            state_list.append(state)
            action_list.append(action)
            reward_list.append(reward)
            dones_list.append(done)
            logp_list.append(logp)
            state = newstate
            steps += 1


        traj = Trajectory(states=state_list, actions=action_list,
        rewards=reward_list, logp=logp_list, dones=dones_list)
        return traj

    def calc_returns(rewards, weight):
        T = len(rewards)
        returns = [0]*T
        returns[T-1] = rewards[T-1]
        for i in range(2,T+1):
            returns[T - i] = returns[T-i + 1]*weight + rewards[T-i]

        return returns

    policy = PolicyNet(input_size, output_size)
    policy.load_state_dict(torch.load("initialCliffWalkController.pt"))


    mean_return_list = []

    traj_list = [collect_trajectory() for _ in range(num_traj)]

    innerProdGradList = []


    policy_loss = traj_list[0].logp[0]


    policy.zero_grad()
    policy_loss.backward()
    gradAtTimeZero = getGrad(policy.fc1.weight.grad,
    policy.fc1.bias.grad,
    policy.fc2.weight.grad, policy.fc2.bias.grad)
    innerProdGradList.append(torch.dot(gradAtTimeZero,
    gradAtTimeZero).item())
    # print((traj_list[0].states))


    for t in range(1,len(traj_list[0].actions)):

        policy_loss = traj_list[0].logp[t]
        # print("Traj lenght:", len(traj_list[0].actions))

        # policy_optimizer.zero_grad()
        policy.zero_grad()
        policy_loss.backward()

        currentGrad = getGrad(policy.fc1.weight.grad,
         policy.fc1.bias.grad,
        policy.fc2.weight.grad, policy.fc2.bias.grad)
     innerProdGradList.append(torch.dot(gradAtTimeZero,currentGrad).item())

    return policy, innerProdGradList


import matplotlib.pyplot as plt

env = gym.make("CliffWalking-v0")

agent, innerProdGradList = gradInnerProductOverTime(env, gamma=0.99, num_traj=1)
plt.plot(innerProdGradList, color = "crimson",
 label=r'$\hat{\mathbb{E}}_{\tau}{\langle\nabla_{\theta}\log\pi_{\theta}(a_0|s_0),\nab

plt.xlabel('t')

plt.title(r'$\hat{\mathbb{E}}_{\tau}{\langle\nabla_{\theta}\log\pi_{\theta}(a_0|s_0),\
+ " under a random policy ")

plt.legend()
plt.savefig('initial.png', format='png', dpi=300)
state = env.reset()
```

CODE FOR THE SECOND SET OF EXPERIMENTS

```
import torch
import gym
import numpy as np
from collections import namedtuple

def getGrad(w1,b1,w2,b2):
    return (torch.cat((torch.flatten(w1),torch.flatten(b1),
    torch.flatten(w2),torch.flatten(b2))))

def basisVector(i,size):
    a = np.zeros(size)
    a[i] = 1.0
    return a

class PolicyNet(torch.nn.Module):
    def __init__(self, input_size, output_size,
     hidden_layer_size=64):
        super(PolicyNet, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Linear(input_size, hidden_layer_size)
        self.fc2 = torch.nn.Linear(hidden_layer_size, output_size)
        self.softmax = torch.nn.Softmax(dim=0)

    def forward(self, x):
        x = torch.from_numpy(basisVector(x,
        self.input_size)).float() # this is for cliff-walking ONLY
        return self.softmax(self.fc2(self.fc1(x)))
      # return self.softmax(self.fc2(torch.nn.functional.relu(self.fc1(x))))

    def get_action_and_logp(self, x):
        # print(x)
        action_prob = self.forward(x)
        m = torch.distributions.Categorical(action_prob)
        action = m.sample()
        logp = m.log_prob(action)
        return action.item(), logp

    def act(self, x):
        action, _ = self.get_action_and_logp(x)
        return action

def gradInnerProductOverTime(env, num_traj=10, gamma=0.99,
           policy_learning_rate=0.01,
             policy_saved_path='xyz.pt'):

    input_size = 4*12 # This is for cliff-walking ONLY
    output_size = env.action_space.n
    Trajectory = namedtuple('Trajectory',
    'states actions rewards dones logp')

    def collect_trajectory():
        state_list = []
        action_list = []
        reward_list = []
        dones_list = []
        logp_list = []
        state = env.reset()[0]
        done = False
        truncated = False
        steps = 0

        while (not (done or truncated)) and steps <= 100:
            action, logp = policy.get_action_and_logp(state)
            newstate, reward, done, truncated,
            info = env.step(action)
            state_list.append(state)
            action_list.append(action)
            reward_list.append(reward)
            dones_list.append(done)
            logp_list.append(logp)
            state = newstate
            steps += 1


        traj = Trajectory(states=state_list,
     actions=action_list, rewards=reward_list, logp=logp_list, dones=dones_list)
        return traj


    policy = PolicyNet(input_size, output_size)
    policy.load_state_dict(torch.load("optimalCliffWalkController.pt"))




    traj_list = [collect_trajectory() for _ in range(num_traj)]
    innerProdGradListList = []
    for i in range(num_traj):
        innerProdGradList = []

        policy_loss = traj_list[i].logp[0]


        policy.zero_grad()
        policy_loss.backward()
        gradAtTimeZero = getGrad(policy.fc1.weight.grad,
      policy.fc1.bias.grad, policy.fc2.weight.grad, policy.fc2.bias.grad)
      innerProdGradList.append(torch.dot(gradAtTimeZero,gradAtTimeZero).item())


        for t in range(1,len(traj_list[i].actions)):

            policy_loss = traj_list[i].logp[t]


            policy.zero_grad()
            policy_loss.backward()

            currentGrad = getGrad(policy.fc1.weight.grad,
         policy.fc1.bias.grad, policy.fc2.weight.grad, policy.fc2.bias.grad)
        innerProdGradList.append(torch.dot(gradAtTimeZero,currentGrad).item())


        innerProdGradListList.append(innerProdGradList)


    return traj_list, innerProdGradListList


env = gym.make("CliffWalking-v0")
num_traj= 100
gamma = 0.99
traj_list, innerProdGradListList = gradInnerProductOverTime(env,
gamma=0.99, num_traj = 100)

from numpy import genfromtxt
q_values = genfromtxt('q_valuesCliffWalk.csv', delimiter=',')
# print(q_values)

averageDifference = 0
for i in range(0,num_traj):
    summation = 0
    for t in range(1,len(traj_list[i].actions)):
        summation += gamma**(t)*innerProdGradListList[i][t]*
```

```
                q_values[traj_list[i].states[t],traj_list[i].actions[t]]
    averageDifference += summation
averageDifference = averageDifference/num_traj

print("Average Difference in numerator: ", averageDifference)

averageFirstsummand = 0
for i in range(0,num_traj):
    summation = 0
    summation += gamma**(0)*innerProdGradListList[i][0]*
                q_values[traj_list[i].states[0],traj_list[i].actions[0]]
    averageFirstsummand += summation
averageFirstsummand = averageFirstsummand/num_traj

print("Average first summand in numerator: ", averageFirstsummand)

state = env.reset()

# RESULTS DELTA_1
# optimal policy numerator average difference : 0.12531747958312045
# approx optimal policy numerator average difference : -10.989155179901417
# decent policy numerator average difference : -37.59848502639185
# initial policy numerator average difference : -48.65707565793603


# RESULTS DELTA_2
# optimal policy average numerator  first summand : -32.6597399706321
# approx optimal policy average numerator  first summand : -85.56502592273189
# decent policy average numerator  first summand : -108.80923245366277
# initial policy average numerator  first summand : -114.84020572163305
```

CODE FOR THE Q-VALUES REQUIRED FOR THE SECOND SET OF EXPERIMENTS

```
import gym
import numpy as np
import pandas as pd


env = gym.make('CliffWalking-v0')
gamma = 0.99
learningRate= 0.1

numStates = 48
numActions = 4



# Initialize Q arbitrarily, in this case a table full of zeros
Q = np.zeros((numStates, numActions))

def epsilonGreedy(Q, state, epsilon):
    u = np.random.uniform(0,1)
    if u < 1 - epsilon:

        return np.argmax(Q[state])
    else:
        return env.action_space.sample()


# Loop for 50000 episodes
for i in range(50000):

    state = env.reset()[0]
    done = False

    # While episode is not done
    while not done:

        action = epsilonGreedy(Q, state, epsilon=0.1)

        nextState, reward, done, truncated, info = env.step(action)
        # Update Q

      tdError = (reward + gamma * np.max(Q[nextState])) - Q[state][action]
        Q[state][action] += learningRate * tdError
        # Update the state
```

```
          state = nextState

np.savetxt("QTableCliffWalk.csv", Q, delimiter=",")
```

CODE FOR THE VPG IMPLEMENTATION FOR CLIFF WALKING

```
import torch
import gym
import numpy as np
from collections import namedtuple

def basisVector(i,size):
    a = np.zeros(size)
    a[i] = 1.0
    return a

class PolicyNet(torch.nn.Module):
    def __init__(self, input_size, output_size, hidden_layer_size=64):
        super(PolicyNet, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Linear(input_size, hidden_layer_size)
        self.fc2 = torch.nn.Linear(hidden_layer_size, output_size)
        self.softmax = torch.nn.Softmax(dim=0)

    def forward(self, x):
        x = torch.from_numpy(basisVector(x,self.input_size)).float()
        return self.softmax(self.fc2(self.fc1(x)))


    def get_action_and_logp(self, x):

        action_prob = self.forward(x)
        m = torch.distributions.Categorical(action_prob)
        action = m.sample()
        logp = m.log_prob(action)
        return action.item(), logp

    def act(self, x):
        action, _ = self.get_action_and_logp(x)
        return action

def vpg(env, num_iter=200, num_traj=10, gamma=0.98,
        policy_learning_rate=0.01, policy_saved_path='noCriticCliffWalking.pt'):

    input_size = 4*12
    output_size = env.action_space.n
   Trajectory = namedtuple('Trajectory', 'states actions rewards dones logp')

    def collect_trajectory():
```

```
    state_list = []
    action_list = []
    reward_list = []
    dones_list = []
    logp_list = []
    state = env.reset()[0]
    done = False
    truncated = False
    steps = 0

    while (not (done or truncated)) and steps <= 200:
        action, logp = policy.get_action_and_logp(state)
        newstate, reward, done, truncated, info = env.step(action)

        state_list.append(state)
        action_list.append(action)
        reward_list.append(reward)
        dones_list.append(done)
        logp_list.append(logp)
        state = newstate
        steps += 1


    traj = Trajectory(states=state_list, actions=action_list,
    rewards=reward_list, logp=logp_list, dones=dones_list)
    return traj

def calc_returns(rewards, weight):
    T = len(rewards)
    returns = [0]*T
    returns[T-1] = rewards[T-1]
    for i in range(2,T+1):
        returns[T - i] = returns[T-i + 1]*weight + rewards[T-i]

    return returns

policy = PolicyNet(input_size, output_size)
policy_optimizer = torch.optim.Adam(policy.parameters(),
 lr=policy_learning_rate)

mean_return_list = []
for it in range(num_iter):
    traj_list = [collect_trajectory() for _ in range(num_traj)]
  returns = [calc_returns(traj.rewards, gamma) for traj in traj_list]
 episodic_returns = [calc_returns(traj.rewards, 1) for traj in traj_list]
```

```
          policy_loss_terms = [-1. *(gamma**j)* traj.logp[j]
       * torch.tensor([[returns[i][j]]]) for i, traj in enumerate(traj_list)
          for j in range(len(traj.actions))]

         policy_loss = 1. / num_traj * torch.cat(policy_loss_terms).sum()


          policy.zero_grad()
          policy_loss.backward()
          policy_optimizer.step()


          mean_return = 1. / num_traj * sum([traj_returns[0]
          for traj_returns in episodic_returns])
          mean_return_list.append(mean_return)

          if mean_return >= -20:
              print("THIS IS IT:", mean_return)
              torch.save(policy.state_dict(), policy_saved_path)
              quit()

          if it % 10 == 0:
            print('Iteration {}: Mean Return = {}'.format(it, mean_return))
              # print(policy.forward(traj_list[0].states[0]))
              torch.save(policy.state_dict(), policy_saved_path)

    return policy, mean_return_list




import matplotlib.pyplot as plt
env = gym.make("CliffWalking-v0")

agent, mean_return_list = vpg(env, num_iter=2000,  gamma=1, num_traj=1)
plt.plot(mean_return_list, color = "crimson")
plt.xlabel('Iteration')
plt.ylabel('Mean Return')
plt.title("No Critic Policy Gradient on cliffWalking")
plt.savefig('noCriticcliffWalking_returns.png', format='png', dpi=300)
state = env.reset()
```