Notes on Reinforcement Learning and Deep Reinforcement Learning

compiled by D.Gueorguiev, 3/20/25

Table of Contents

Introductory Notes

What is Reinforcement Learning: branch of machine learning concerned with making decisions and taking sequences of actions based on some current state thereby maximizing some reward objective over time.

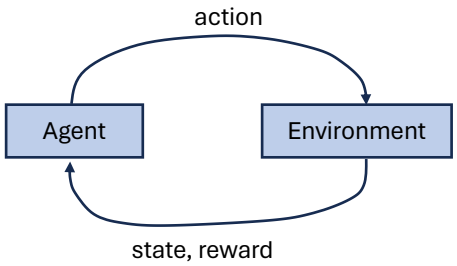
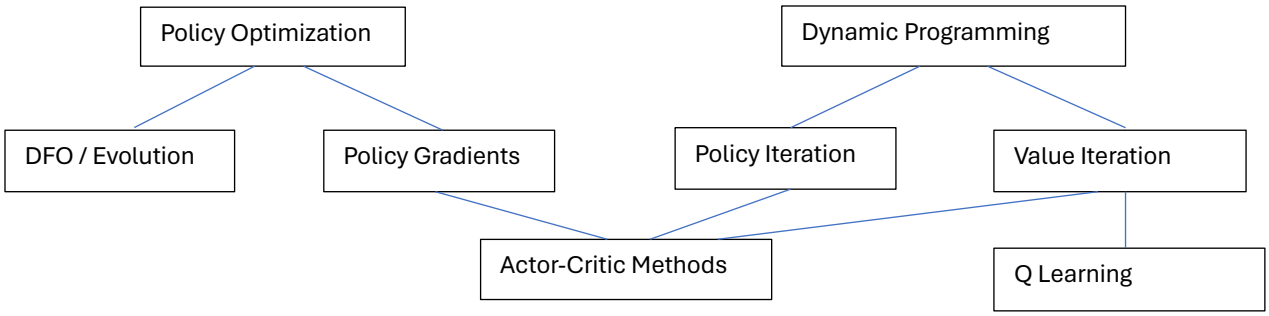


Figure 1: Feedback loop between the Agent and the Environment in RL

The Agent and the Environment interact with each other on discrete timesteps creating a feedback loop depicted in Figure 1. The Agent has a goal of maximizing the cumulative reward while interacting with the Environment.

Observations in RL:
Robotics: camera images, joint angles
Actions in RL:
Robotics: joint torques
Rewards in RL:
Robotics: stay balanced, navigate to target locations

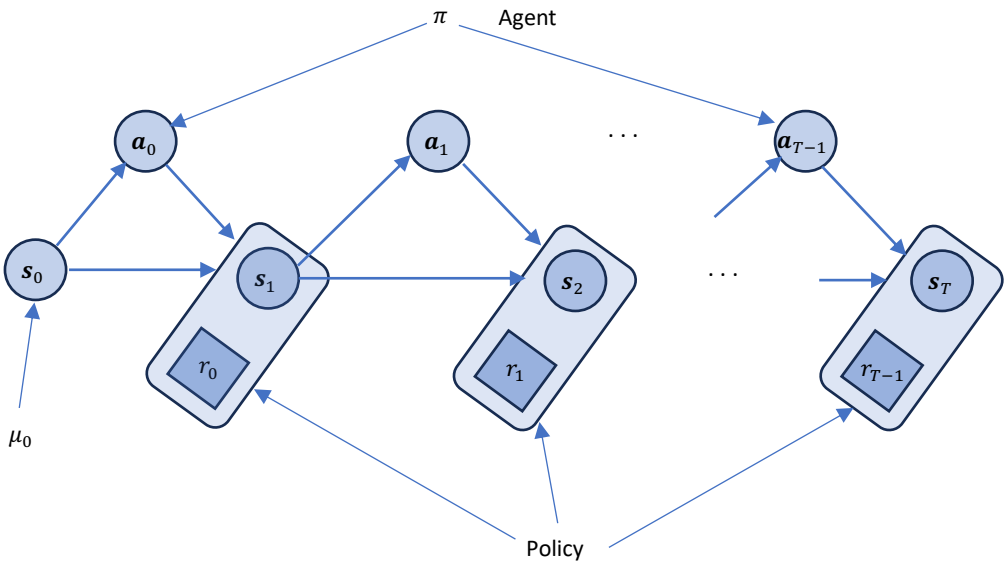
Approaches to RL



Two approaches to RL – the first approach is to optimize policy and the second one is dynamic programming. Policy is the function which takes the observations with the state of the system and outputs actions. The Policy Optimization approach looks at the RL problem as an optimization problem trying to optimize the expected reward , there are parameters in the policy, and we want to find such set of parameters which maximizes the expectation of the stochastic reward. Posing the problem as an optimization problem ignores all of the structure of the problem conveyed through the Bellman's equations. We are getting a noisy estimate of how good each parameter is and try to move toward that part of the parameter space where we are getting better performance – that is, higher expected reward. So, this is how the Derivative Free Optimization (DFO) methods and Evolutionary algorithms work – they work as a black box which takes a policy parameter vector and outputs a noisy performance number. These methods are very simple to implement, and they work surprisingly well. The other approach for Policy Optimization is by using Policy Gradient methods trying to measure the gradient of the performance with respect to the parameter vector of the policy. These second type of Policy Optimization methods scale better with respect to the number of parameters. Dynamic Programming / Approximate Dynamic Programming is a very different approach to solving RL problems. In certain cases, we can solve control problems exactly. What if we have slightly different parameter settings, will these algorithms still work? It turns out we can modify these algorithms in certain ways which can keep them valid. Policy Iteration and Value Iteration are the two algorithms which will exactly solve the MDP formulation of the RL problem and finding the optimal policy, but they only work if you have discrete state space and action space. If these spaces are finite sets, we can solve exactly the RL problem. In many real world problems, we need to do approximate versions of these algorithms. There is a dedicated field developing approximate dynamic programming algorithms for those real world problems which cannot be solved exactly. Q-Learning is one quite popular and successful method in this category. We can do Q Learning with function approximation performed by Neural network. Lastly, there are Actor-Critic methods – they are policy gradient methods, but they also use value functions helping the policy gradient method. These methods can scale well and be used to solve large / hard problems.

<u>What is Deep RL?</u>
It is RL using nonlinear function approximators, which do not make a lot of assumptions about the form of the approximated function. At any given time, the algorithm is solving optimization problem with gradient descent.

Markov Decision Process (MDP)
MDP is defined by the triplet $(\mathcal{S}, \mathcal{A}, p)$ where
$\mathcal{S}$ is the state space
$\mathcal{A}$ is the action space
$p(r, \boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ is the transition probability distribution
p tells us the probability of the reward r (scalar) , the next state $\boldsymbol{s}'$ (vector) given the current state $\boldsymbol{s}$ (vector) and the action $\boldsymbol{a}$ (vector).
Extra objects can be defined depending on the problem setting:
μ - initial state distribution
γ – discount factor



In each episode, the initial state is sampled from μ, and the process proceeds until terminal state is reached.
In the episodic setting the agent experiences are broken up into a sequence of episodes. In each episode a reward and a new state are generated from the old state after action is chosen. And this process continuous for each episode sequentially until we reach a terminal state. The terminal state is a special state with 0 reward from which there is no continuation. We can have different termination semantics – termination can indicate a good outcome (example: taxi robot reaches its destination), termination is neither good nor bad and always occurs (waiter robot finishes its shift after fixed amount of time), termination indicates bad outcome (walking robot fails over).
We want to maximize the expected reward per episode.

We consider two kinds of policies:

deterministic policies $\boldsymbol{a} = \pi(\boldsymbol{s})$
stochastic policies $\boldsymbol{a} \sim \pi(\boldsymbol{a} \mid \boldsymbol{s})$ – in this case the policy defines a conditional probability distribution over actions.
The policy will be the optimized function in our optimization problem.

Parametrized policy π_θ
there is some parameter vector $\boldsymbol{\theta}$ which indexes over the policy.

So based on this discussion we can write:

$\boldsymbol{s}_0 \sim \mu(\boldsymbol{s}_0)$
$\boldsymbol{a}_0 \sim \pi(\boldsymbol{a}_0|\boldsymbol{s}_0)$
$\boldsymbol{s}_1, r_0 \sim p(\boldsymbol{s}_1, r_0|\boldsymbol{s}_0, \boldsymbol{a}_0)$
$\boldsymbol{a}_1 \sim \pi(\boldsymbol{a}_1|\boldsymbol{s}_1)$
$\boldsymbol{s}_2, r_1 \sim p(\boldsymbol{s}_2, r_1|\boldsymbol{s}_1, \boldsymbol{a}_1)$
…
$\boldsymbol{a}_{T-1} \sim \pi(\boldsymbol{a}_{T-1}|\boldsymbol{s}_{T-1})$
$\boldsymbol{s}_T, r_{T-1} \sim p(\boldsymbol{s}_T, r_{T-1}|\boldsymbol{s}_{T-1}, \boldsymbol{a}_{T-1})$
$\boldsymbol{s}_T$ – terminal state (in.1)

Objective:
maximize $\eta(\pi)$, where
$\eta(\pi) = \mathbb{E}[r_0 + r_1 + \cdots + r_{T-1}|\pi]$ (in.2)

Parametrized policies

A family of policies indexed by parameter vector $\theta \in \mathbb{R}^d$
Deterministic policies: $\boldsymbol{a} = \pi(\boldsymbol{s}, \boldsymbol{\theta})$
Stochastic policies: $\pi(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{\theta})$
Obtaining an expression/function for the parametrized policy is analogous to applying a regressive model on input $\boldsymbol{s}$ and output $\boldsymbol{a}$ to obtain estimate for π. One way to obtain policy function is by constructing and training appropriate neural network for the purpose. This is the beginning of the Deep Reinforcement Learning methods.
With discrete action space the network outputs a vector of probabilities. With continuous action space the network could output mean and covariance matrix for example.

Derivative-Free Optimization Approach

Goal in RL: maximize the expected reward -

maximize $\mathbb{E}[R|\pi(\cdot, \boldsymbol{\theta})]$

The derivative-free optimization approach looks at the mapping $\boldsymbol{\theta} \rightarrow \blacksquare \rightarrow R$ as a black box. In order to maximize the expected return we are ignoring all other information except than the return R for the current episode.

Cross-Entropy Method as black-box Optimization method

The Cross-Entropy Method is an evolutionary algorithm, which at every point in time maintains a distribution of parameter vectors, some of which have higher fitness than others, so it tends to use those parameter vectors which have higher fitness than the rest ([18],[19],[20]). Cross-Entropy Method effectiveness has been compared against Approximate Dynamic Programming effectiveness developed by Dimitri Bertsekas *at al* (see for example [16] and [17]).
An important detail is that in the example considered in [17] there are 40 features on the Tetris screen, and we need to learn those features. It turns out that for problems with such size the Cross-Entropy method is especially suitable and renders comparable performance to the Approximate Dynamic Programming formulation. In general, for problems with such size the derivative-free optimization algorithms are very suitable tool.

Covariance Matrix Adaptation algorithm is modified Cross-Entropy Method by added heuristics which improves its performance in certain problems (see [22] and [23]).

Cross-Entropy Method algorithm

Initialize $\boldsymbol{\mu} \in \mathbb{R}^d, \boldsymbol{\sigma} \in \mathbb{R}^d$
iterations $1,2, \ldots$
Collect n samples of $\theta_i \sim \mathcal{N}\left(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma})\right)$
Perform a noisy evaluation $R_i \sim \theta_i$
Select the top p% of samples (e.g. $p = 20$), which we will call the ***elite set***.
Fit a Gaussian distribution, with diagonal covariance, to the elite set obtaining a new $\boldsymbol{\mu}, \boldsymbol{\sigma}$
Return the final $\boldsymbol{\mu}$

//TODO: finish this section (Lecture 1 of John Schulman)

Policy Gradient Methods

Again, Goal in RL: maximize the expected reward -

maximize $\mathbb{E}[R|\pi(\cdot, \boldsymbol{\theta})]$

Intuition:

collect a set of trajectories and
1) Make good trajectories more probable and make the bad trajectories less probable
2) Make the good actions more probable (Actor-Critic method (A2C), generalized advantage estimator (GAE))
3) Push actions toward good actions (Deterministic Policy Gradient Algorithms (DPG), Stochastic Value Gradient (SVG)) (see [45], [46])

//TODO: finish this section (mostly Lecture 2 of John Schulman)

Intro to Policy Optimization

We consider the case of stochastic parametrized policy π_θ. We want to maximize the expected return $J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}[R(\tau)]$. For the purposes of this derivation, we will take $R(\tau)$ to give the finite horizon undiscounted return, but the derivation for the infinite horizon discounted return setting is almost identical.
We would like to optimize the policy by gradient ascent as

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k} \qquad \text{(ipo.1)}$$

The gradient of policy performance, $\nabla_\theta J(\pi_\theta)$, is called policy gradient, and algorithms that optimize the policy this way are called *policy gradient algorithms*. Examples of such algorithms include Vanilla Policy Gradient and TRPO. PPO is often referred to as a policy gradient algorithm, though this is slightly inaccurate as PPO uses value functions to obtain better policy approximation.

To actually use this algorithm, we need an expression for the policy gradient which we can numerically compute. This involves two steps : 1) deriving the analytical gradient of policy performance, which turns out to have the form of an expected value, which can be computed with data from a finite number of agent-environment interaction steps.

We begin with few definitions and statements used in the derivation of policy gradient

Definition *Probability of a Trajectory*
The probability of a trajectory $\tau = (s_0, a_0, \ldots, s_{T+1})$ given that actions come from π_θ is

$$P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \qquad \text{(ipo.2)}$$

Definition *The Log-Derivative Trick*
$$\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\nabla_\theta \log P(\tau|\theta) \qquad \text{(ipo.3)}$$

Definition *Log-Probability of a Trajectory*
The log-prob of a trajectory is just

$$\log P(\tau|\theta) = \log \rho_0(s_0) + \sum_{t=0}^{T}(\log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)) \qquad \text{(ipo.4)}$$

Statement: *The gradients of environment functions are zero*. This is because the environment has no dependence on θ, so gradients of $\rho_0(s_0), P(s_{t+1}|s_t, a_t)$, and $R(\tau)$ are zero.

Then the expression of the Grad-Log-Prob of a trajectory is given with

$$\begin{aligned}\nabla_\theta \log P(\tau|\theta) &= \cancel{\nabla_\theta \log \rho_0(s_0)} + \sum_{t=0}^{T}(\cancel{\nabla_\theta \log P(s_{t+1}|s_t, a_t)} + \nabla_\theta \log \pi_\theta(a_t|s_t)) \\ &= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t). \qquad \text{(ipo.5)}\end{aligned}$$

Putting it all together, we derive the following:

Derivation for Basic Policy Gradient

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau\sim\pi_\theta}{\mathrm{E}}[R(\tau)] \\ &= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) \\ &= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) \\ &= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)\, R(\tau) \quad \text{(via the Log derivative trick)} \\ &= \underset{\tau\sim\pi_\theta}{\mathrm{E}}[\nabla_\theta \log P(\tau|\theta)\, R(\tau)] \quad \text{(rewritten as expectation)}\end{aligned}$$

$$\therefore \nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}[\textstyle\sum_{t=0}^{T} \nabla_\theta \log P(\tau|\theta)\, R(\tau)] \quad \text{(expression for grad-log-prob)} \qquad \text{(ipo.6)}$$

This is an expectation, which means that we can estimate it with a sample mean. If we collect a set of trajectories $\mathcal{D} = \{\tau_i\}_{i=1..N}$ where each trajectory is obtained by letting the agent act in the environment using the policy π_θ, the policy gradient can be estimated with

$$\hat{g} = \frac{1}{|\mathcal{D}|} \textstyle\sum_{\tau\in\mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, R(\tau) \quad , \qquad \text{(ipo.7)}$$

where $|\mathcal{D}|$ is the number of trajectories in $\mathcal{D}$ (here, N).

This last expression is the simplest version of the computable expression we desired. Assuming that we have represented our policy in a way which allows us to calculate $\nabla_\theta \log \pi_\theta(a_t|s_t)$, and if we are able to run the policy in the environment to collect the trajectory dataset, we can compute the policy gradient and take an update step.

Lemma *EGLP Lemma*
Expected Grad-Log-Prob is zero.
Suppose that P_θ is a parametrized probability distribution over a random variable, x. Then:

$$\underset{x\sim P_\theta}{\mathrm{E}}[\nabla_\theta \log P_\theta(x)] = 0 \qquad \text{(ipo.8)}$$

Proof:
Since $P_\theta(x)$ is a distribution, we have:

$$\int_x P_\theta(x) = 1$$

Applying gradient on both sides gives us:

$$\nabla_\theta \int_x P_\theta(x) = \nabla_\theta 1 = 0$$

Use the log derivative trick to get:

$$\begin{aligned}0 &= \nabla_\theta \int_x P_\theta(x) \\ &= \int_x \nabla_\theta P_\theta(x) \\ &= \int_x P_\theta(x)\nabla_\theta P_\theta(x) \\ \therefore 0 &= \underset{x\sim P_\theta}{\mathrm{E}}[\nabla_\theta \log P_\theta(x)]\end{aligned}$$

Let us examine the most recent expression for the policy gradient:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}[\textstyle\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, R(\tau)] \qquad \text{(ipo.9)}$$

Taking a step with this gradient pushes up the log-probabilities of each action in proportion to $R(\tau)$ which of course is the sum of all rewards obtained along the trajectory τ. However, such proportional push for *all* possible actions does not make much sense. Instead, agents should really only reinforce actions on the basis of their *consequences*. Rewards obtained before taking an action have no bearing on how good the action was: only rewards that come *after*. We can express this intuition with the expression for the policy gradient:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\textstyle\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})\right] \qquad \text{(ipo.10)}$$

In this form, actions are only reinforced based on rewards obtained after they are taken.
We'll call this form the "reward-to-go gradient", because the sum of rewards after a point in a trajectory,

$\hat{R}_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$ (ipo.11)

is called the reward-to-go from that point, and this policy gradient expression depends on the reward-to-go from state-action pairs.

A key problem with policy gradients is how many sample trajectories are needed to get a low-variance estimate for them. The formula we started with-

$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} [\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, R(\tau)]$ (ipo.12)

includes terms for reinforcing actions proportional to past rewards all of which had zero mean but non-zero variance: as a result, they would just add noise to sample estimates of the policy gradient. By removing them we reduce the number of sample trajectories needed.

Baselines in Policy Gradients

An immediate consequence of the EGLP lemma is that for any function b which only depends on state,

$\underset{a_t \sim \pi_\theta}{\mathrm{E}} [\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, b(s_t)] = 0$ (bpg.1)

This allows us to add or subtract any number of terms like this from our expression for the policy gradient, without changing it in expectation:

$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)\right)\right]$ (bpg.2)

Any function b used in this way is called a *baseline*.

The most common choice of baseline is the on-policy value function $V^\pi(s_t)$. Recall that this is the average return an agent gets if it starts in state s_t and then acts according to policy π for the rest of its life.

Empirically, the choice $b(s_t) = V^\pi(s_t)$ has the desirable effect of reducing variance in the sample estimate for the policy gradient. This results in faster and more stable policy learning. Conceptually, it encodes the intuition that if an agent gets what it expected, it should "feel" neutral about it.

Note:

In reality, $V^\pi(s_t)$ cannot be computed exactly, so it has to be approximated. This is usually done with a neural network, $V_\phi(s_t)$, which is updated concurrently with the policy so that the value network always approximates the value function of the most recent policy.

The simplest method for learning $V_\phi(s_t)$, used in most implementations of policy optimization algorithms (including VPG, TRPO, PPO and A2C) is to minimize a mean-squared-error objective

$\phi_k = \underset{\phi}{\operatorname{argmin}} \underset{s_t, \hat{R}_t \sim \pi_k}{\mathrm{E}} \left[\left(V_\phi(s_t) - \hat{R}_t\right)^2\right]$ (bpg.3)

where π_k is the policy at epoch k. This is done with one or more steps of gradient descent, starting from the previous value parameters ϕ_{k-1}.

Other Forms of the Policy Gradient

What we have seen so far is that the policy gradient has the general form

$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} [\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, \Phi_t]$ (ofpg.1)

where Φ_t could be any of

$\Phi_t = R(\tau),$ (ofpg.2)

or

$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$ (ofpg.3)

or

$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$ (ofpg.4)

All of this forms of Φ_t lead to the same expected value for the policy gradient, despite having different variances. There are two more good choices of weights Φ_t which we will discuss

1) ***On-Policy Action-Value Function***
$\Phi_t = Q^{\pi_\theta}(s_t, a_t)$ is a good choice for policy gradient weights.

Proof:
We need to prove that
$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, Q^{\pi_\theta}(s_t, a_t)]$. We will prove it for finite-horizon undiscounted return setting. We will sketch a proof for the infinite horizon discounted return.
We rewrite the expression for the policy gradient, using an expression for the reward-to-go as:

$\hat{R}_t = \sum_{t'=t}^{T} R(s_t', a_t', s_{t'+1})$ (ofpg.5)

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\hat{R}_t\right] \\ &= \sum_{t=0}^{T} \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{R}_t\right]\end{aligned}$$ (ofpg.6)

Define $\tau_{:t} = (s_0, a_0, \ldots, s_t, a_t)$ as the trajectory up to time t, and $\tau_{t:}$ as the remainder of the trajectory after that. By the law of iterated expectations, we can break up the preceding expression into:

$$\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{T} \underset{\tau_{:t}\sim\pi_\theta}{\mathrm{E}}\left[\underset{\tau_{t:}\sim\pi_\theta}{\mathrm{E}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{R}_t|\, \tau_{:t}\right]\right]$$ (ofpg.7)

The grad-log-prob is constant with respect to the inner expectation because it depends on s_t and a_t, which the inner expectation conditions on as fixed in $\tau_{:t}$. Thus, the grad-log-prob can be pulled out of the inner expectation expression:

$$\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{T} \underset{\tau_{:t}\sim\pi_\theta}{\mathrm{E}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \underset{\tau_{t:}\sim\pi_\theta}{\mathrm{E}}\left[\hat{R}_t|\tau_{:t}\right]\right]$$ (ofpg.8)

In MDP, the future only depends on the most recent state and action. As a result, the inner expectation, which expects over the future, conditioned on the entirety of the past (everything up to time t) – is equal to the same expectation if it only conditioned on the last timestep (s_t, a_t):

$$\underset{\tau_{t:}\sim\pi_\theta}{\mathrm{E}}\left[\hat{R}_t|\tau_{:t}\right] = \underset{\tau_{t:}\sim\pi_\theta}{\mathrm{E}}\left[\hat{R}_t|s_t, a_t\right]$$ (ofpg.9)

which is the definition of $Q^{\pi_\theta}(s_t, a_t)$: expected return, starting from state s_t and action a_t, when acting on-policy for the rest of the trajectory. ■
2) **The Advantage Function**

The Advantage of an action is given with

$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$ (ofpg.10)

describes how much better or worse it is than other actions on average relative to the current policy.

This choice $\Phi_t = A^{\pi_\theta}(s_t, a_t)$ is a good choice for policy gradient weights.

Proof: The argument is equivalent to the argument that $\Phi_t = Q^{\pi_\theta}(s_t, a_t)$ and then use the value function as a baseline which leads to an unbiased estimate.

Vanilla Policy Gradient

Background
The key idea underlying policy gradients is to push up the probabilities of actions that lead to higher return and push down the probabilities of actions that lead to lower return, until we arrive at optimal policy.

VPG is on-policy algorithm and can be used as a model with environments with either discrete or continuous action spaces.

As before, let $J(\pi_\theta)$ denote the expected finite-horizon undiscounted return of the policy. The gradient of $J(\pi_\theta)$ is

$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} [\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)]$ (vpg.1)

where τ is a trajectory and A^{π_θ} is the advantage function for the current policy.

The policy gradient algorithm works by updating the policy parameters via stochastic gradient descent on policy performance:

$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_{\theta_k})$ (vpg.2)

Policy gradient implementations typically compute advantage function estimates based on the infinite-horizon discounted return, despite otherwise using the finite-horizon undiscounted policy gradient formula.

VPG trains a stochastic policy in an on-policy way. This means that it explores by sampling actions according to the latest version of its stochastic policy. The amount of randomness in action selection depends on both initial conditions and the training procedure.

//TODO: finish the VPG algorithm

Generalized Advantage Estimator

As specified in [35] let us consider an undiscounted formulation of the policy optimization problem. The initial state s_0 is sampled from distribution ρ_0. A trajectory $(s_0, a_0, s_1, a_1, \dots)$ is generated by sampling actions according to policy $a_t \sim \pi(a_t \mid s_t)$ and sampling the states according to the dynamics $s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$, until a terminal absorbing state is reached. A reward $r_t = r(s_t, a_t, s_{t+1})$ is received at each timestep. The goal is to maximize the expected total reward $\sum_{t=0}^{\infty} r_t$, which is assumed to be finite for all policies. Notice that discount is not introduced yet in the algorithm. Discount will be introduced at a later step as an algorithm parameter that adjusts the bias-variance tradeoff. In [35] Schulman asserts that we can absorb the discount factor in the reward function by making it time dependent.

Recall that policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $\nabla_\theta \mathbb{E}[\sum_{t=0}^{\infty} \mathrm{r}_t]$. There are several different related expressions for the policy gradient, which have the form:

$g = \mathbb{E}[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)]$ (gae.1)

where Ψ_t may be one of the following:

1) total reward of the trajectory: $\sum_{t=0}^{\infty} r_t$

2) reward following action a_t: $\sum_{t'=t}^{\infty} r_{t'}$

3) baselined version of the reward following action a_t: $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$

4) state-action value function: $Q^\pi(s_t, a_t)$

5) advantage function: $A^\pi(s_t, a_t)$

6) TD residual: $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$

The latter formulas use the definitions

$$V^\pi(s_t) = \mathop{\mathbb{E}}_{s_{t+1:\infty}, a_{t:\infty}}[\sum_{l=0}^{\infty} r_{t+l}] \qquad Q^\pi(s_t, a_t) = \mathop{\mathbb{E}}_{s_{t+1:\infty}, a_{t+1:\infty}}[\sum_{l=0}^{\infty} r_{t+l}] \qquad \text{(gae.2)}$$

$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$ (gae.3) Advantage function

Here the subscript of $\mathbb{E}$ enumerates the variables being integrated over, where states and actions are sampled sequentially from the dynamics function $P(s_{t+1} \mid s_t, a_t)$ and policy $\pi(a_t \mid s_t)$, respectively. The colon notation $a : b$ refers to the inclusive range $[a, a+1, \ldots, b]$.

We will introduce a parameter γ that allows us to reduce variance by downweighing rewards corresponding to delayed effects, at the cost of introducing bias. This parameter corresponds to the discount factor used in discounted formulations of MDPs, but it can be treated as a variance reduction parameter in an undiscounted problem. For details on this technique to incorporate the discount factor see [37].
The discounted value functions are given by:

$$V^{\pi,\gamma}(s_t) = \mathop{\mathbb{E}}_{s_{t+1:\infty}, a_{t:\infty}}[\sum_{l=0}^{\infty} \gamma^l r_{t+l}] \qquad Q^{\pi,\gamma}(s_t, a_t) = \mathop{\mathbb{E}}_{s_{t+1:\infty}, a_{t+1:\infty}}[\sum_{l=0}^{\infty} \gamma^l r_{t+l}] \qquad \text{(gae.4)}$$

$A^{\pi,\gamma}(s_t, a_t) = Q^{\pi,\gamma}(s_t, a_t) - V^{\pi,\gamma}(s_t)$ (gae.5) Discounted advantage function

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

$$g^\gamma = \mathop{\mathbb{E}}_{s_{0:\infty}, a_{0:\infty}}[\sum_{t=0}^{\infty} A^{\pi,\gamma}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)] \qquad \text{(gae.6)}$$

Question: how to obtain a reasonably biased estimator for $A^{\pi,\gamma}$ which will give us a noisy estimate for the discounted policy gradient in (gae.6)?

γ-just estimator

We will introduce the notion of a γ-just estimator of the advantage function. The γ-just estimator is an estimator which does not introduce bias when we use it in place of the unknown quantity $A^{\pi,\gamma}$ in the expression for g^γ.
Note that we have already introduced bias by using $A^{\pi,\gamma}$ in place of A^π; here we are concerned with obtaining an unbiased estimate of g^γ, which is a biased estimate of the policy gradient of the undiscounted MDP.

Consider an advantage estimator $\hat{A}_t(s_{0:\infty}, a_{0:\infty})$ which in general is a function of the entire trajectory.

Definition: The estimator $\hat{A}_t$ is γ-just if

$$\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\hat{A}_t(s_{0:\infty},a_{0:\infty})\nabla_\theta \log \pi_\theta(a_t|s_t)\right] = \mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)] \qquad \text{(gae.7)}$$

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

$$\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\sum_{t=0}^{\infty}\hat{A}_t(s_{0:\infty},a_{0:\infty})\nabla_\theta \log \pi_\theta(a_t|s_t)\right] = g^\gamma \quad \text{(gae.8)}$$

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

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

Proof:

First, we split the expectation in the approximate expression for g^γ into two terms involving Q and b:

$$\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\,\hat{A}_t(s_{0:\infty},a_{0:\infty})\right] = \mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(Q_t(s_{t:\infty},a_{t:\infty}) - b_t(s_{0:t},a_{0:t-1})\right)\right]$$

$$= \mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\,Q_t(s_{t:\infty},a_{t:\infty})] - \mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\,b_t(s_{0:t},a_{0:t-1})] \quad \text{(gae.9)}$$

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

$$\begin{aligned}
&\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\,Q_t(s_{t:\infty},a_{t:\infty})] = \\
&= \mathop{\mathbb{E}}_{s_{0:t},a_{0:t}}\left[\mathop{\mathbb{E}}_{s_{t+1:\infty},a_{t+1:\infty}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\,Q_t(s_{t:\infty},a_{t:\infty})]\right] \\
&= \mathop{\mathbb{E}}_{s_{0:t},a_{0:t}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\mathop{\mathbb{E}}_{s_{t+1:\infty},a_{t+1:\infty}}[Q_t(s_{t:\infty},a_{t:\infty})]\right] \\
&= \mathop{\mathbb{E}}_{s_{0:t},a_{0:t}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\,A^\pi(s_t,a_t)] \qquad \text{(gae.10)}
\end{aligned}$$

Next,

$$\begin{aligned}
&\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\,b_t(s_{0:t},a_{0:t-1})] \\
&= \mathop{\mathbb{E}}_{s_{0:t},a_{0:t-1}}\left[\mathop{\mathbb{E}}_{s_{t+1:\infty},a_{t:\infty}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\,b_t(s_{0:t},a_{0:t-1})]\right] \\
&= \mathop{\mathbb{E}}_{s_{0:t},a_{0:t-1}}\left[\mathop{\mathbb{E}}_{s_{t+1:\infty},a_{t:\infty}}[\nabla_\theta \log \pi_\theta(a_t|s_t)]b_t(s_{0:t},a_{0:t-1})\right] \\
&= \mathop{\mathbb{E}}_{s_{0:t},a_{0:t-1}}[0\cdot b_t(s_{0:t},a_{0:t-1})] = 0 \quad \text{(gae.11)}
\end{aligned}$$

The last identity is true due to the EGLP Lemma proven earlier.

Thus we have shown that when $\mathop{\mathbb{E}}_{s_{t+1:\infty},a_{t+1:\infty}|s_t,a_t}[Q_t(s_{t:\infty},a_{t:\infty})] = Q^{\pi,\gamma}(s_t,a_t)$ holds then

$$\begin{aligned}
&\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\,\hat{A}_t(s_{0:\infty},a_{0:\infty})\right] = \mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(Q_t(s_{t:\infty},a_{t:\infty}) - b_t(s_{0:t},a_{0:t-1})\right)\right] = \\
&\mathop{\mathbb{E}}_{s_{0:t},a_{0:t}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\,Q^\pi(s_t,a_t)] = \mathop{\mathbb{E}}_{s_{0:t},a_{0:t}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(Q^{\pi,\gamma}(s_t,a_t) - V^{\pi,\gamma}(s_t)\right)\right] = \\
&\mathop{\mathbb{E}}_{s_{0:t},a_{0:t}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(A^{\pi,\gamma}(s_t,a_t)\right)\right] \qquad \text{(gae.12)}
\end{aligned}$$

Proposition: The following expressions are γ-just advantage estimators for $\hat{A}_t$:

$$\text{(a)}\ \sum_{l=0}^{\infty}\gamma^l r_{t+l}\ ,\ \text{(b)}\ A^{\pi,\gamma}(s_t,a_t)\ ,\ \text{(c)}\ Q^{\pi,\gamma}(s_t,a_t)\ ,\ \text{(d)}\ r_t + \gamma V^{\pi,\gamma}(s_{t+1}) - V^{\pi,\gamma}(s_t) \qquad \text{(gae.13)}$$

Proof:
(a) Let us start with $\sum_{l=0}^{\infty}\gamma^l r_{t+l}$ – according to the definition of γ-just advantage estimator we must show that

$$\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\hat{A}_t(s_{0:\infty},a_{0:\infty})\nabla_\theta \log\pi_\theta(a_t|s_t)\right] = \mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)\right] \quad \text{(gae.14)}$$

where $\hat{A}_t(s_{0:\infty},a_{0:\infty}) = \sum_{l=0}^{\infty}\gamma^l r_{t+l}$.

Thus, the LHS of (gae.14) becomes $\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\sum_{l=0}^{\infty}\gamma^l r_{t+l}\,\nabla_\theta \log\pi_\theta(a_t|s_t)\right] =$

$$\mathop{\mathbb{E}}_{s_{0:t},a_{0:t}}\left[\mathop{\mathbb{E}}_{s_{t+1:\infty},a_{t+1:\infty}}\left[\sum_{l=0}^{\infty}\gamma^l r_{t+l}\right]\nabla_\theta \log\pi_\theta(a_t|s_t)\right] = \mathop{\mathbb{E}}_{s_{0:t},a_{0:t}}\left[Q^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)\right] =$$

$$\mathop{\mathbb{E}}_{s_{0:t},a_{0:t}}\left[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)\right] = \mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)\right] \quad \text{(gae.15)}$$

The last expression in (gae.15) is obviously the RHS of (gae.14).
Notice that in order to obtain (gae.15) we have used the fact that $\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\nabla_\theta \log\pi_\theta(a_t|s_t)\,V^{\pi,\gamma}(s_t)\right] = 0$ which follows from (gae.11).

(b) $A^{\pi,\gamma}(s_t,a_t)$ is an γ-just advantage estimator
For $\hat{A}_t = A^{\pi,\gamma}(s_t,a_t)$ the γ-just requirement $\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\hat{A}_t(s_{0:\infty},a_{0:\infty})\nabla_\theta \log\pi_\theta(a_t|s_t)\right] =$
$\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)\right]$ is trivially satisfied.

(c) $Q^{\pi,\gamma}(s_t,a_t)$ is an γ-just advantage estimator
For $\hat{A}_t = Q^{\pi,\gamma}(s_t,a_t)$ we want to prove the identity $\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\hat{A}_t(s_{0:\infty},a_{0:\infty})\nabla_\theta \log\pi_\theta(a_t|s_t)\right] =$
$\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)\right]$

The LHS of the identity to be proven is:

$$\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[\hat{A}_t(s_{0:\infty},a_{0:\infty})\nabla_\theta \log\pi_\theta(a_t|s_t)\right] = \mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[Q^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)\right] \quad \text{(gae.16)}$$

The RHS of (gae.16) can be rewritten as:

//TODO: finish the section on Generalized Advantage Estimator

References

[1] Deep Reinforcement Learning: Lecture 1: Intro, Episodic Reinforcement Learning and Markov Decision Processes, John Schulman, OpenAI, Berkeley (MLSS Cadiz, 2016)
[2] Deep Reinforcement Learning: Lecture 2: More on Cross-Entropy Method and Value Functions, John Schulman, OpenAI, Berkeley (MLSS Cadiz, 2016)
[3] Deep Reinforcement Learning: Lecture 3: More on Episodic Reinforcement Learning and Policy Gradients, John Schulman, OpenAI, Berkeley (MLSS Cadiz, 2016)
[4] Deep Reinforcement Learning: Lecture 4: Performance of Policies, Policy Approximations, Parametrized Policies, Asynchronous Methods, John Schulman, OpenAI, Berkeley (MLSS Cadiz, 2016)
[5] Reinforcement Learning Course: Lecture 1: Intro to Reinforcement Learning, David Silver, DeepMind x UCL, 2015
[6] Reinforcement Learning Course: Lecture 2: Markov Decision Processes, David Silver, DeepMind x UCL, 2015
[7] Reinforcement Learning Course: Lecture 3: Planning by Dynamic Programming, David Silver, DeepMind x UCL, 2015
[8] Reinforcement Learning Course: Lecture 4: Model-Free Prediction, David Silver, DeepMind x UCL, 2015
[9] Reinforcement Learning Course: Lecture 5: Model-Free Control, David Silver, DeepMind x UCL, 2015
[10] Reinforcement Learning Course: Lecture 6: Value-Function Approximation, David Silver, DeepMind x UCL, 2015
[11] Reinforcement Learning Course: Lecture 7: Policy Gradient Methods, David Silver, DeepMind x UCL, 2015
[12] Reinforcement Learning Course: Lecture 8: Integrating Learning and Planning, David Silver, DeepMind x UCL, 2015
[13] Reinforcement Learning Course: Lecture 9: Exploration and Exploitation, David Silver, DeepMind x UCL, 2015
[14] Reinforcement Learning Course: Lecture 10: Classic Games, David Silver, DeepMind x UCL, 2015
[15] Learning Tetris Using the Noisy Cross-Entropy Method, Istvan Szita, Andras Loerincz, 2006
[16] Approximate Dynamic Programming - A Series of Lectures Given at CEA-Cadarache, France, Summer 2012, Dimitri Bertsekas
[17] Approximate Dynamic Programming Finally Performs Well In The Game of Tetris, Victor Gabillon et al, INRIA, 2013
[18] A Tutorial On The Cross-Entropy Method, Pieter-Tjerk de Boer et al, MIT, 2003
[19] The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning, RY Rubinstein, DP Kroese, 2004
[20] Application of the Cross-Entropy Method to the Buffer Allocation Problem in a Simulation-Based Environment, G. Allon et al, 2005
[21] Introduction to Rare Events Simulation, John F. Shortle, Pierre L'Eccuyer, Draft, 2011
[22] The CMA Evolution Strategy : A Tutorial, Nikolaus Hansen, Inria, 2023
[23] Optimizing Walking Controllers for Uncertain Inputs and Environments, Jack M. Wang et al, 2010
[24] Heavy Tails, Importance Sampling, and Cross-Entropy, Soren Asmussen, Dirk Kroese, Reuven Rubinstein, 2003
[25] Maximum Likelihood Theory for Incomplete Data from Exponential Family, Rolf Sundberg, U. of Stockholm, 1974
[26] Maximum Likelihood from Incomplete Data via the EM Algorithm, A.P. Dempster, 1977
[27] The EM Algorithm: An Old Folk Song Sung to Fast New Tune, XL Meng, 1997
[28] A Legacy of EM Algorithms, Kenneth Lange et al, 2023
[29] The MM Alternative to EM, TT Wu, Kenneth Lange, 2011
[30] Nonconvex Optimization via MM Algorithms: Convergence Theory, Kenneth Lange, 2021
[31] Reparametrization trick, Wikipedia
[32] The Log-derivative trick, Andy Jones' technical blog
[32] OpenAI: Spinning Up Part 1 Key Concepts in RL
[33] OpenAI: Spinning Up Part 2 Kinds of RL Algorithms
[34] OpenAI: Spinning Up Part 3 Intro to Policy Optimization
[35] High-Dimensional Continuous Control Using Generalized Advantage Estimation, John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, Pieter Abbeel, 2015
[36] PyTorch implementation of GAE: https://nn.labml.ai/rl/ppo/gae.html
[37] Approximate Gradient Methods in Policy-Space Optimization of Markov Reward Processes, Peter N. Marbach, John Tsitsiklis, 2003
[38] Notes on Generalized Advantage Estimator, Seita's Place blog, 2017

[39] Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning, Evan Greensmith, Peter L. Bartlett, Jonathan Baxter, JMLR, 2004
[40] Actor-Critic Algorithms, CS 294-112: Deep Reinforcement Learning, Sergey Levine, 2017
[41] Policy Gradient Methods for Reinforcement Learning with Function Approximation, Richard Sutton et al, AT&T, 1999
[42] Actor-Critic Algorithms, Vijay Konda, John Tsitsiklis, NIPS, 1999
[43] On Actor-Critic Algorithms, Vijay Konda, John Tsitsiklis, MIT, SIAM, 2003
[44] Off-Policy Actor-Critic, Thomas Degris, Martha White, Richard S. Sutton, 2013
[45] Deterministic Policy Gradient Algorithms, David Silver, DeepMind, 2014
[46] Learning Continuous Control Policies by Stochastic Value Gradients, Nicolas Heese et al, DeepMind, 2015

Appendix

Stochastic Gradient Descent

Gradient Bandit Algorithm and Stochastic Gradient Descent

//TODO: finish Gradient Bandit interpretation as SGD

//TODO: finish the appendix on SGD