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.

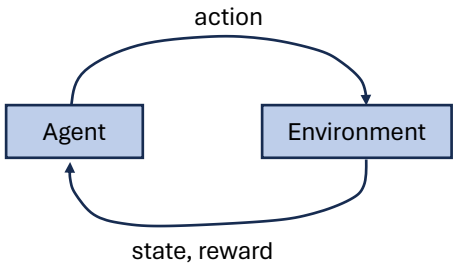


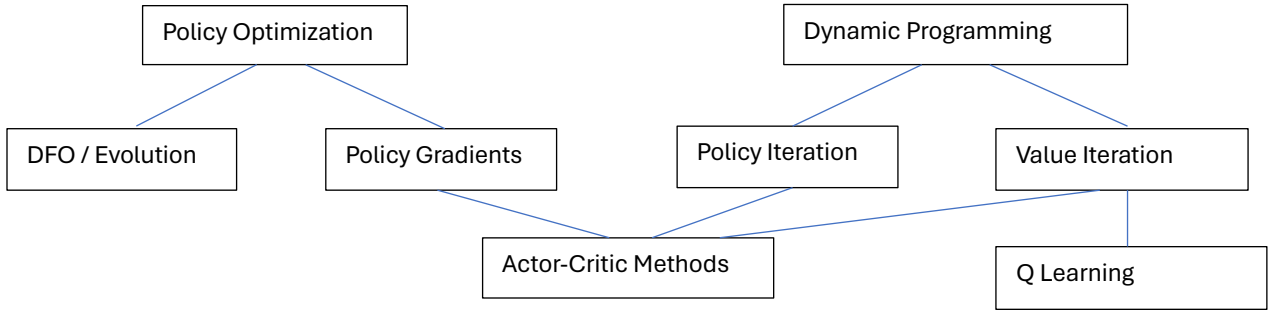
Figure 1: Feedback loop between the Agent and the Environment in RL

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

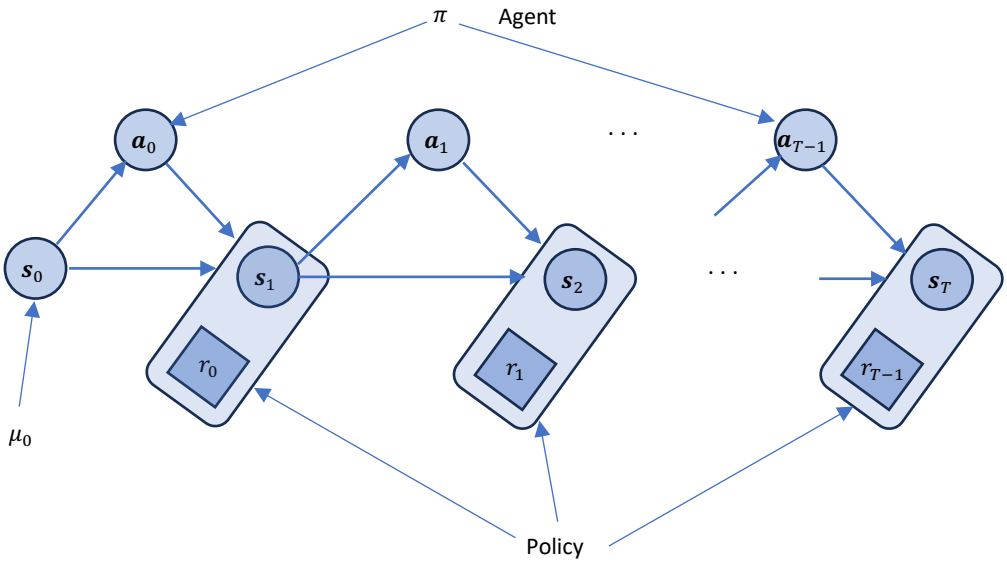


Two approaches to RL – the first approach is to optimize policy and the second one is dynamic programming. Policy is the function which takes the observations with the state of the system and outputs actions. The Policy Optimization approach looks at the RL problem as an optimization problem trying to 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.

What is Deep RL?
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]).

Sketch of an example Cross-Entropy method algorithm for optimization problems

Initialize $\boldsymbol{\mu} \in \mathbb{R}^d, \boldsymbol{\sigma} \in \mathbb{R}^d$
iterations 1,2, …
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 $\boldsymbol{\mu}$

Let us delve a bit deeper into the Cross Entropy method and some of its variations.

Cross Entropy Method for Rare Event Estimation with Examples

Example 1:
Consider the weighted graph on the Figure below with random weights with random weights $X_1, \dots, X_5$. Suppose the weights are independent of each other and are exponentially distributed with means $u_1, \dots, u_5$, respectively. Define $\boldsymbol{X} = (X_1, \dots, X_5)$ and $\boldsymbol{u} = (u_1, \dots, u_5)$. Define the PDF of $\boldsymbol{X}$ by $f(\cdot; \boldsymbol{u})$. Thus ,

$$f(x; \boldsymbol{u}) = \exp\left(-\sum_{j=1}^{5} \frac{x_j}{u_j}\right) \prod_{j=1}^{5} \frac{1}{u_j} \qquad \text{(cem.1)}$$

Let $S(\boldsymbol{X})$ be the total length of the shortest path from node A to node B. We wish to estimate from simulation

$$\ell = \mathbb{P}(S(\boldsymbol{X}) \geq \gamma) = \mathbb{E} I_{\{S(\boldsymbol{X}) \geq \gamma\}} \qquad \text{(cem.2)}$$

that is, the probability that the length of the shortest path $S(\boldsymbol{X})$ will exceed some fixed γ.

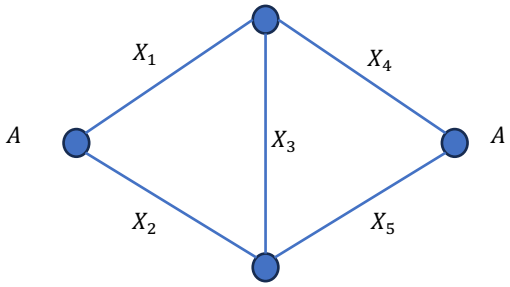


Figure: Shortest path from A to B

A straightforward way to estimate ℓ in (cem.2) is to use *Crude Monte Carlo* (CMC) simulation. That is, we draw a random sample $\boldsymbol{X}_1, \dots, \boldsymbol{X}_N$ from the distribution of X and use

$$\frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}) \geq \gamma\}} \qquad \text{(cem.3)}$$

as the unbiased estimator of ℓ. However, for a large γ the probability ℓ will be very small and CMC requires a very small and CMC requires a very large simulation effort, that is, N needs to be very large in order to estimate ℓ accurately – that is, to obtain a small relative error, for example of 1%. A better way to perform the simulation is to use *importance sampling* (IS). That is, let g be another probability density such that $g(\boldsymbol{x}) = 0 \Longrightarrow I_{\{S(\boldsymbol{X}) \geq \gamma\}} f(\boldsymbol{x}) = 0$. Using the density g we can represent ℓ as

$$\ell = \int I_{\{S(\boldsymbol{X}) \geq \gamma\}} \frac{f(\boldsymbol{x})}{g(\boldsymbol{x})} g(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_g I_{\{S(\boldsymbol{X}) \geq \gamma\}} \frac{f(\boldsymbol{x})}{g(\boldsymbol{x})} \qquad \text{(cem.4)}$$

Here the subscript g in the expectation on the last expression of (cem.4) indicates the importance sampling (IS) density.

An unbiased estimator of ℓ is

$$\hat{\ell} = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}) \geq \gamma\}} W(\boldsymbol{X}_i) \qquad \text{(cem.5)}$$

where $\hat{\ell}$ is called the importance sampling (IS) or the likelihood ratio (LR) estimator

$$W(\boldsymbol{x}) = f(\boldsymbol{x}) / g(\boldsymbol{x}) \qquad \text{(cem.6)}$$

is called the likelihood ratio (LR), and $\boldsymbol{X}_1, \dots, \boldsymbol{X}_N$ is a random sample from g, that is, $\boldsymbol{X}_1, \dots, \boldsymbol{X}_N$ are iid r.v.'s with density g. Obviously, the LR estimator in (cem.6) reduces to the CMC estimator (cem.3) when $g = f$.

If we restrict ourselves to g such that $\boldsymbol{X}_1, \dots, \boldsymbol{X}_N$ are independent and exponentially distributed with means $u_1, \dots, u_N$, then

$$W(\boldsymbol{x}; \boldsymbol{u}, \boldsymbol{v}) = \frac{f(\boldsymbol{x};\boldsymbol{u})}{f(\boldsymbol{x};\boldsymbol{v})} = \exp\left(-\sum_{j=1}^{N} x_j \left(\frac{1}{u_j} - \frac{1}{v_j}\right)\right) \prod_{j=1}^{N} \frac{v_j}{u_j} \qquad \text{(cem.7)}$$

In this case the "change of measure" is determined by the parameter vector $\boldsymbol{v} = (v_1, \dots, v_N)$. The problem is how to select $\boldsymbol{v}$ which gives the most accurate estimate of ℓ for a given simulation effort. One advantage of the CE method for rare event simulation is that it provides a fast way to determine/estimate the optimal parameters. An CE algorithm in this regard for rare event estimation is presented below:

CE Algorithm for rare event estimation

1) Define $\hat{\boldsymbol{v}}_0 := \boldsymbol{u}$. Set iteration counter $t := 1$

2) Generate a random sample $\boldsymbol{X}_1, \dots, \boldsymbol{X}_N$ according to the pdf $f(\cdot; \hat{v}_{t-1})$.

Calculate the performances $S(\boldsymbol{X}_i)$ for all i, and order them from smallest to biggest, $S_{(1)} \leq \cdots \leq S_{(N)}$. Let $\hat{\gamma}_t$ be the $(1-\rho)$ sample quantile of performances: $\hat{\gamma}_t := S_{(\lceil (1-\rho)N \rceil)}$, provided this is less than γ. Otherwise, put $\hat{\gamma}_t := \gamma$.

3) Use the same sample to calculate, for $j = 1, \dots, n$

$$\hat{v}_{t,j} = \frac{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \hat{\gamma}_t\}} W(\boldsymbol{X}_i; \boldsymbol{u}, \hat{\boldsymbol{v}}_{t-1}) X_{ij}}{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \hat{\gamma}_t\}} W(\boldsymbol{X}_i; \boldsymbol{u}, \hat{\boldsymbol{v}}_{t-1})} \qquad \text{(cem.8)}$$

4) If $\hat{\gamma}_t = \gamma$ then proceed to step 5); otherwise set $t := t + 1$ and reiterate from step 2)

5) Let T be the final iteration. Generate a sample $\boldsymbol{X}_1, \dots, \boldsymbol{X}_{N_1}$ according to the pdf $f(\cdot; \hat{\boldsymbol{v}}_T)$ and estimate ℓ via the IS estimate

$$\hat{\ell} = \frac{1}{N_1} \sum_{i=1}^{N_1} I_{\{S(\boldsymbol{X}_i) \geq \gamma\}} W(\boldsymbol{X}_i; \boldsymbol{u}, \hat{\boldsymbol{v}}_T) \qquad \text{(cem.9)}$$

Note that in steps 2-4 the optimal IS parameter $\hat{\boldsymbol{v}}$ is estimated. In the final step 5 this parameter is used to estimate the probability of interest. The parameters ρ (usually in $[0.01, 0.1]$) , N and N_1 need to be supplied in advance.

Example 2:

Consider a binary vector $\boldsymbol{y} = (y_1, \dots, y_n)$. Suppose that we do not know which components of $\boldsymbol{y}$ are 0s and which are 1s. However, we have an "oracle" which for each binary *input* vector $x = (x_1, \dots, x_n)$ returns the performance or response

$$S(\boldsymbol{x}) = n - \sum_{j=1}^{n} |x_j - y_j| \qquad \text{(cem.10)}$$

representing the number of matches between the elements of $\boldsymbol{x}$ and $\boldsymbol{y}$. The goal is to present a random search which reconstructs/decodes the unknown vector $\boldsymbol{y}$ by maximizing the function $S(\boldsymbol{x})$ on $[0,1]^n$ – the space of n-dimensional binary vectors.

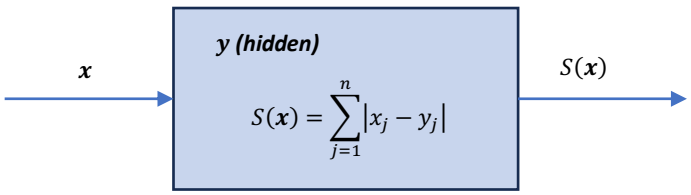


Figure: A black box for decoding vector $\boldsymbol{y}$

A naïve way to find y is to repeatedly generate binary vectors $X = (X_1, \dots, X_n)$ such that $X_i, i = 1..n$ are independent Bernoulli r.v's with probabilities $p_1, \dots, p_n$: $\boldsymbol{X} \sim \mathrm{Ber}(\boldsymbol{p})$ where $\boldsymbol{p} = (p_1, \dots, p_n)$. Note that for $p = y$ we have $S(\boldsymbol{X}) = n, \boldsymbol{X} = \boldsymbol{y}$ and the naïve search algorithm yields the optimal solution with probability 1.
The CE method for combinatorial optimization consists of casting the underlying problem into the rare event framework given with the earlier Algorithm. We create a sequence of parameter vectors $\widehat{\boldsymbol{p}}_0, \widehat{\boldsymbol{p}}_1, \dots$ and levels $\hat{\gamma}_1, \hat{\gamma}_2, \dots$ converges to the optimal performance (n here) and the sequence $\widehat{\boldsymbol{p}}_0, \widehat{\boldsymbol{p}}_1, \dots$ converges to the optimal parameter vector ($\boldsymbol{y}$ here). The CE procedure is described in the algorithm below similar to the earlier algorithm for Example 1.

CE Algorithm for rare event estimation solving Example 2

1) Start with some $\widehat{\boldsymbol{p}}_0$, say $\widehat{\boldsymbol{p}}_0 = \left(\frac{1}{2}, \frac{1}{2}, \dots, \frac{1}{2}\right)$. Let $t := 1$.

2) Draw a sample $\boldsymbol{X}_1, \dots, \boldsymbol{X}_N$ of Bernoulli vectors with probability vector $\widehat{\boldsymbol{p}}_{t-1}$. Calculate the performances $S(X_i)$ for all i, and order them from smallest to biggest, $S_{(1)} \leq \cdots \leq S_{(N)}$. Let $\hat{\gamma}_t$ be $(1-\rho)$ sample quantile of the performances $\hat{\gamma}_t = S_{(\lceil (1-\rho)N \rceil)}$.

3) Use the same sample to calculate $\widehat{\boldsymbol{p}}_t = \left(\hat{p}_{t,1}, \dots, \hat{p}_{t,n}\right)$ via

$$\hat{p}_{t,j} = \frac{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \hat{\gamma}_t\}} I_{\{X_{ij}=1\}}}{\sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \hat{\gamma}_t\}}} \qquad \text{(cem.11)}$$

$j = 1, .., n$, where $\boldsymbol{X}_i = (X_{i1}, \dots, X_{in})$, and increase t by 1.

4) if the *stopping criterion* is met, then `stop`; otherwise increment t go back to step 2).

Note: possible stopping criteria
i) $\hat{\gamma}_t$ variance drops below certain threshold for some predefined number of iterations
ii) stop when the probability vector $\widehat{\boldsymbol{p}}_t$ has become a binary array

Notice the similarity of the CE algorithms for both Examples 1 and 2. The method can be formulated more generally, and the generic algorithm will hold in both cases. Since this section of this document is supposed to give the reader only intuitive understanding of the method the reader is advised to look for details in [18] and [19].
//TODO: maybe be give the general formalism from the paper

CE Method for Solving Combinatorial Optimization Problems

Consider the following maximization problem: let $\mathcal{X}$ be a finite set of states and let S be a real-valued *performance function* on $\mathcal{X}$. We wish to find the maximum of S over $\mathcal{X}$, and the corresponding state(s) at which this maximum is attained. Let us denote the maximum by γ^*. Thus,

$$S(\boldsymbol{x}^*) = \gamma^* = \max_{\boldsymbol{x} \in \mathcal{X}} S(\boldsymbol{x}) \qquad \text{(cem.12)}$$

The very first thing we want to do in the CE method for this kind of problems is to associate an *estimation problem* with the original optimization problem given by (cem.12). For this purpose, we define a collection of indicator functions $\{I_{\{S(\boldsymbol{x}) \geq \gamma\}}\}$ on $\mathcal{X}$ for various *levels* (effectively thresholds) $\gamma \in \mathbb{R}$. Next, let $\{f(\cdot; \boldsymbol{v}), \boldsymbol{v} \in \mathcal{V}\}$ be a family of discrete pdfs on $\mathcal{X}$, parametrized by a real-valued parameter (vector) $\boldsymbol{v}$.

For a certain $\boldsymbol{u} \in \mathcal{V}$ we associate with (cem.12) the problem of estimating the number

//TODO: finish the CE method for combinatorial optimization problems

//TODO: Finish the section on Cross-Entropy Methods

The Monte Carlo Methods

Monte Carlo Methods are a family of learning methods for (i) estimation of the value functions and (ii) discovering optimal policies. With Monte-Carlo methods we do not need complete knowledge of the environment, this family of methods requires only experience – sample sequences of states, actions, and rewards from actual or simulated interaction with an environment. Learning from experience implies that no prior knowledge of the environment dynamics is required. Another use case of this family of methods is learning from simulated experience – in this scenario a model is needed but this model does not need to be accurate one generating the complete probability distributions of all possible transitions , required to formulate the dynamic programming equations which can be solved exactly.

Monte Carly family of methods solve the reinforcement learning problem based on averaging sample returns. To ensure that well-defined returns are available we will consider Monte Carlo family of methods only for episodic tasks where we do not need to deal with infinite quantities. We assume that experience is divided into episodes and that all episodes will eventually terminate no matter what actions are selected.
Only on the completion of an episode are value estimates and policies changed. Monte Carlo methods can thus be incremental in an episode-by-episode

//TODO: finish the section on Monte-Carlo Methods

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

Temporal-Difference Learning

//TODO: finish the section on Temporal Difference Learning (use Sutton's book chapter on TD and the original TD article)

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*
$\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)$ (ipo.3)

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

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

<u>Derivation for Basic Policy Gradient</u>

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau\sim\pi_\theta}{\mathrm{E}}[R(\tau)] \\ &= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) \\ &= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) \\ &= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)\, R(\tau) \quad \text{(via the Log derivative trick)} \\ &= \underset{\tau\sim\pi_\theta}{\mathrm{E}}[\nabla_\theta \log P(\tau|\theta)\, R(\tau)] \quad \text{(rewritten as expectation)} \\ \therefore \nabla_\theta J(\pi_\theta) &= \underset{\tau\sim\pi_\theta}{\mathrm{E}}[\textstyle\sum_{t=0}^{T} \nabla_\theta \log P(\tau|\theta)\, R(\tau)] \quad \text{(expression for grad-log-prob)} \qquad \text{(ipo.6)}\end{aligned}$$

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

$\hat{g} = \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, R(\tau)$, (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:
$\underset{x\sim P_\theta}{\mathrm{E}}[\nabla_\theta \log P_\theta(x)] = 0$ (ipo.8)
<u>Proof:</u>
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:
$0 = \nabla_\theta \int_x P_\theta(x)$

$$= \int_x \nabla_\theta P_\theta(x)$$
$$= \int_x P_\theta(x) \nabla_\theta P_\theta(x)$$
$$\therefore 0 = \mathop{\mathrm{E}}_{x \sim P_\theta} [\nabla_\theta \log P_\theta(x)]$$

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

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

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:

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

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,

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

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-

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

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.

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])

Score Function Gradient Estimator for Policies

Consider the expectation $\mathop{\mathbb{E}}_{x \sim p(x|\theta)} [f(x)]$. We want to compute the gradient with respect to θ.

$$\nabla_\theta \mathbb{E}_x[f(x)] = \nabla_\theta \int dx\, p(x \mid \theta)\, f(x) \qquad \text{(sfge.1)}$$

$$
\begin{aligned}
&= \int dx\, \nabla_\theta p(x \mid \theta)\, f(x) \\
&= \int dx\, p(x \mid \theta) \frac{\nabla_\theta p(x \mid \theta)}{p(x \mid \theta)} f(x) \\
&= \int dx\, p(x \mid \theta)\, \nabla_\theta p(x \mid \theta) f(x) \\
&= \mathop{\mathbb{E}}_{x}[f(x) \nabla_\theta p(x \mid \theta)] \qquad \text{(sfge.2)}
\end{aligned}
$$

In order to compute the gradient estimator all we need to do is to take samples of x.
The last expression results in an unbiased gradient estimator. By sampling $x_i \sim p(x \mid \theta)$ and computing $\hat{g}_i = f(x_i) \nabla_\theta p(x \mid \theta)$ we obtain the unbiased estimate of the gradient of the expectation of $f(x)$.
So we need to be able to compute and differentiate density $p(x \mid \theta)$ with respect to θ.

Importance Sampling provides an elegant alternative derivation for the gradient of the expectation of $f(x)$.
For details on Importance Sampling see [24].

Derivation via Importance Sampling

$$
\mathop{\mathbb{E}}_{x \sim \theta}[f(x)] = \mathop{\mathbb{E}}_{x \sim \theta_{\text{old}}}\left[\frac{p(x|\theta)}{p(x|\theta_{\text{old}})} f(x)\right]
$$

$$
\nabla_\theta \mathop{\mathbb{E}}_{x \sim \theta}[f(x)] = \mathop{\mathbb{E}}_{x \sim \theta_{\text{old}}}\left[\frac{\nabla_\theta p(x|\theta)}{p(x|\theta_{\text{old}})} f(x)\right]
$$

$$
\begin{aligned}
\nabla_\theta \mathop{\mathbb{E}}_{x \sim \theta}[f(x)]\Big|_{\theta=\theta_{\text{old}}} &= \mathop{\mathbb{E}}_{x \sim \theta_{\text{old}}}\left[\frac{\nabla_\theta p(x|\theta)|_{\theta=\theta_{\text{old}}}}{p(x|\theta_{\text{old}})} f(x)\right] \\
&= \mathop{\mathbb{E}}_{x \sim \theta_{\text{old}}}\left[\nabla_\theta \log p(x \mid \theta)|_{\theta=\theta_{\text{old}}} f(x)\right]
\end{aligned}
$$

Intuition of Score Function Gradient Estimator

$$
\hat{g}_i = f(x_i) \nabla_\theta \log p(x \mid \theta) \qquad \text{(sfge.3)}
$$

Let us assume that $f(x)$ measures the goodness / fitness of the sample x. Moving in the direction $\hat{g}_i$ pushes up the log prob of the sample, in proportion to how good it is.

Let us consider a whole trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1})$ and now x represents a whole trajectory. Then we can write

$$
\nabla_\theta \mathbb{E}_\tau[R(\tau)] = \mathbb{E}_\tau[\nabla_\theta \log p(\tau|\theta) R(\tau)] \qquad \text{(sfge.4)}
$$

Using the chain rule we write:

$$
p(\tau|\theta) = \mu(s_0) \prod_{t=0}^{T-1} [\pi(a_t|s_t, \theta) P(s_{t+1}, r_t|s_t, a_t)] \qquad \text{(sfge.5)}
$$

Then

$$
\log p(\tau|\theta) = \log \mu(s_0) + \sum_{t=0}^{T-1} [\log \pi(a_t|s_t, \theta) + \log P(s_{t+1}, r_t|s_t, a_t)] \qquad \text{(sfge.6)}
$$

Applying gradient on each side of the last equation leads to:

$$
\nabla_\theta \log p(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t, \theta) \qquad \text{(sfge.7)}
$$

Thus, we get:

$$
\nabla_\theta \mathbb{E}_\tau[R(\tau)] = \mathbb{E}_\tau[R(\tau) \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t, \theta)] \qquad \text{(sfge.8)}
$$

Interpretation : using good trajectories leading to high total reward can be interpreted as selecting good supervised examples in classification (for discrete action and state spaces) or regression (continuous action and/or state spaces). We do not know a priori the good examples (that is, the trajectories with high reward) – we must find those through trial and error via reinforcement learning.
Problem with the estimator given with the RHS of (sfge.8) is albeit it is not biased estimator it would results in really high variance.
Why? Because we have summation over all actions along the trajectory and each of them is a random variable with its own sizeable variance. The implication for an estimator with high variance is that we will need many samples before we start converging to a good solution. So the question is how do we reduce the variance of the policy gradient estimator?

So to recap here is our unbiased high variance estimator for the policy gradient:

$$\nabla_\theta \mathbb{E}_\tau[R(\tau)] = \mathbb{E}_\tau[(\sum_{t=0}^{T-1} r_t)(\nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t,\theta))] \quad \text{(sfge.9)}$$

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

Baselines in Policy Gradients

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

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

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:

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

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

$$\phi_k = \mathop{\text{argmin}}_{\phi} \mathop{\mathbb{E}}_{s_t,\hat{R}_t\sim\pi_k}\left[\left(V_\phi(s_t) - \hat{R}_t\right)^2\right] \qquad \text{(bpg.3)}$$

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

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

where Φ_t could be any of

$$\Phi_t = R(\tau), \qquad \text{(ofpg.2)}$$

or

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

or

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

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 $\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathbb{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, Q^{\pi_\theta}(s_t, a_t)\right]$. 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:

$$\hat{R}_t = \sum_{t'=t}^{T} R(s'_t, a'_t, s_{t'+1}) \qquad \text{(ofpg.5)}$$

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

Define $\tau_{:t} = (s_0, a_0, \dots, s_t, a_t)$ 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:

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

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:

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

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):

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

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

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

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

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

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:

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

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 $r_t = r(s_t, a_t)$ 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

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

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

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

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

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

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

Proposition: *Sufficient condition for an estimator $\hat{A}_t$ to be γ-just*
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.
Let us assume 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:

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

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

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

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

Next,

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

$$\underset{s_{0:t},a_{0:t}}{\mathbb{E}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\, Q^\pi(s_t,a_t)] = \underset{s_{0:t},a_{0:t}}{\mathbb{E}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(Q^{\pi,\gamma}(s_t,a_t) - V^{\pi,\gamma}(s_t)\right)\right] = \underset{s_{0:t},a_{0:t}}{\mathbb{E}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(A^{\pi,\gamma}(s_t,a_t)\right)\right] \quad \text{(gae.12)}$$

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

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

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

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 $\underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}[\sum_{l=0}^{\infty}\gamma^l r_{t+l}\,\nabla_\theta \log \pi_\theta(a_t|s_t)] =$

$$\underset{s_{0:t},a_{0:t}}{\mathbb{E}}\left[\underset{s_{t+1:\infty},a_{t+1:\infty}}{\mathbb{E}}[\sum_{l=0}^{\infty}\gamma^l r_{t+l}]\,\nabla_\theta \log \pi_\theta(a_t|s_t)\right] = \underset{s_{0:t},a_{0:t}}{\mathbb{E}}[Q^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)] =$$
$$\underset{s_{0:t},a_{0:t}}{\mathbb{E}}[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)] = \underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)] \quad \text{(gae.15)}$$

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 $\underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\, V^{\pi,\gamma}(s_t)] = 0$ 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 $\underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}\left[\hat{A}_t(s_{0:\infty},a_{0:\infty})\nabla_\theta \log \pi_\theta(a_t|s_t)\right] =$
$\underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)]$ 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 $\underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}[Q^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)] =$
$\underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)]$

The LHS of the identity to be proven is:

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

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

$$\underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)] = \underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}[Q^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)]$$

as we have shown earlier that $\underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\, V^{\pi,\gamma}(s_t)] = 0$ by (gae.11) which proves (c).

(d) $r_t + \gamma V^{\pi,\gamma}(s_{t+1}) - V^{\pi,\gamma}(s_t)$ is an γ-just advantage estimator

For $\hat{A}_t = r_t + \gamma V^{\pi,\gamma}(s_{t+1}) - V^{\pi,\gamma}(s_t)$ we want to prove the identity

$$\underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}\left[\left(r_t + \gamma V^{\pi,\gamma}(s_{t+1}) - V^{\pi,\gamma}(s_t)\right)\nabla_\theta \log \pi_\theta(a_t|s_t)\right] = \underset{s_{0:\infty},a_{0:\infty}}{\mathbb{E}}[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)] \quad \text{(gae.17)}$$

The LHS of (gae.17) can be written as:

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

The last relation is true because $\mathop{\mathbb{E}}_{s_{0:\infty},a_{0:\infty}}\left[Q^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)\right]=$

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

$$\mathop{\mathbb{E}}_{s_{0:t+1},a_{0:t}}\left[\left(r_t+\gamma\mathop{\mathbb{E}}_{s_{t+2:\infty},a_{t+1:\infty}}\left[\left(\sum_{l=0}^{\infty}\gamma^l r_{t+1+l}\right)\right]\right)\nabla_\theta \log \pi_\theta(a_t|s_t)\right]=\mathop{\mathbb{E}}_{s_{0:t+1},a_{0:t}}\left[\left(r_t+\gamma V^{\pi,\gamma}(s_{t+1})\right)\nabla_\theta \log \pi_\theta(a_t|s_t)\right]=$$

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

Advantage Function Estimation

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

$$\hat{g}=\frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\infty}\hat{A}_t^n\nabla_\theta \log \pi_\theta(a_t^n|s_t^n) \qquad \text{(gae.18)}$$

where n indexes over a batch of episodes.
Let V be an approximate value function.

//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
[47] Simple statistical gradient-following algorithms for connectionist reinforcement learning, Ronald J. Williams, Northeastern U, 1992
[48] Note on Q Functions in Reinforcement Learning, D. Gueorguiev, 2023

Appendix

Stochastic Gradient Descent

Gradient Bandit Algorithm and Stochastic Gradient Descent

//TODO: finish Gradient Bandit interpretation as SGD

//TODO: finish the appendix on SGD