Deep Reinforcement Learning

Han Cheol Moon
School of Computer Science and Engineering
Nanyang Technological University
Singapore
hancheol001@edu.ntu.edu.sg

November 27, 2023

Contents

Chapter 1

Introduction

1.1 Introduction

Reinforcement learning (RL) is a science of decision making. RL is learning what to do so as to maximize a numerical reward signal. The learner is not told which actions to take, but instaed must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. There are some differences compared to other machine learning paradigms:

- RL uses training information that *evaluates* the actions taken rather than *instructs* by giving correct actions.
  - Evaluative feedback, not about best or worst.
  - There is no supervisor, only a *reward* signal
- Delayed reward: Feeedback is delayed, not instantaneous.
- Time really matters (sequential, non i.i.d. data).
- Agent's action affects the subsequent data it receives. e.g., a robot moves around a room and it receives a different front view.
- The environment is initially unknown. The agent interacts with the environment. The agent improves its policy
  - Trial-and-error search
- Planning: the environment is known.
- Exploitation: the model finds more information about the environment
- Exploration: the model exploits known information

1.1.1 Sequential Decision Making

- Goal: select actions to maximize total future reward

- Actions may have long term consequences (we have to plan ahead)
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward.

Fully observability: agent directly observes environment state. Fully observable environment is one in which the agent can always see the entire state of environment. In case of fully observable environments all relevant portions of the environment are observable.

$$O_t = S_t^a = S_t^w$$

So agent state is equal to environment state and information state. Formally, this is a **Markov decision process** (MDP).

Partially observability: agent indirectly observes environment. For example, a robot with camera vision isn't told its absolute location or a trading agent doesn't tell about a tread and only observes current prices. Now agent state is not equal to the environment state. Formally this is a partially observable Markov decision process (POMDP).

RL agents can be categorized as follows:

- Value based: policy is implicit
- Policy based
- Actor-Critic
- Model Free
- Model based

1.1.2 Exploration and Exploitation

- Reinforcement learning is like a trial-and-error learning
- The agent should discover a good policy from its experiences of the environment without losing too much reward along the way
- Exploration: finds more information about the environment
- Exploitation: exploits known information to maximize reward
  - Restaurant selection:
    - Exploitation: go to your favorite restaurant
    - Exploration: try a new
  - Oil drilling:
    - Exploitation: drill at the best location
    - Exploration: try a new location

1.2 Markov Chain

- Reachable: $i \to j$
- Communicate: $i \leftrightarrow j$
- Irreducible: $i \leftrightarrow j, \forall i, j$
- Absorbing state: If the only possible transition is to itself. This is also a terminal state.
- Transient state: A state s is called a transient state, if there is another state s', that is reachable from s, but not vice versa.
- Recurrent state: A state that is not transient.
- Periodic: A state is periodic if all of the paths leaving s come back after some multiple steps ($k > 1$).
  - Recurrent state is aperiodic if $k = 1$.
- Ergodicity if a Markov chain follows:
  - Irredicible
  - Recurrent
  - Aperiodic

1.3 Multi-Armed Bandits

Bandit problems are stateless. Each arm has a fixed distribution of rewards. It does not depend on which arms were pulled previously. The goal is to explore the reward distributions of all arms and then keep pulling the best one. We only have a single chance of selecting an action in each episode.

Markov decision processes are a temporal extension of bandit problems: pulling an arm influences the future rewards. Technically, there is a state that changes by pulling an arm. The reward distributions depend on that state.

You can view bandit problems as Markov decision processes where all states are terminal. In that case, all decision sequences have a length of 1 and subsequent pulls don't influence each other.

- When the lever of a slot machine is pulled it gives a random reward coming from a probability distribution specific to that machine.
- Although the machines look identical, their reward probability distributions are different.
- In each turn, gamblers need to decide whether to play the machine that has given the highest average reward so far, or to try another machine.

In our k-armed bandit problem, each of the k actions has an expected or mean reward given that that action is selected: let us call this the *value* of that action. We denote the action selected on time step t as A_t, and the corresponding reward as R_t. The value then of an arbitrary action a, denoted $q_*(a)$, is the expected reward given that a is selected:

$$q_*(a) = \mathbb{E}[R_t | A_t = a].$$

Since we do not know which action is the best, we have to estimate the value of actions a at time step t, $Q_t(a)$.

1.3.1 Action-value Methods

One natural way to estimate this is by averaging the rewards actually received: $Q_t(a)$ is sum of rewards when a taken prior t over number of times a taken prior to t:

$$Q_t(a) = \frac{\sum_{t=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{t=1}^{t-1} \mathbb{1}_{A_i=a}}$$

Then, the simplest action selection rule is to select one of the actions with the highest estimated value, which is a greedy action selection method represented as follows:

$$A_t = \underset{a}{\operatorname{argmax}}\, Q_t(a).$$

This approach always exploits current knowledge to maximize immediate reward; it spends no time at all sampling apparently inferior actions to see if they might really be better. A simple alternative is to behave greedily most of the time, but every once in a while, instead select randomly from among all the actions. This near greedy action selection rule is called ϵ-greedy method. In the limit as the number of steps increases, every action will be sampled an infinite number of times, which ensures all the $Q_t(a)$ converge to $q_*(a)$.

1.3.2 Incremental Implementation of The Action-value Methods

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)Q_n\right) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]
\end{aligned}
$$

- Q_n: Old estimate
- Q_{n+1}: New estimate
- R_n: New reward

This is an incremental formulas for updating averages with small, constant computation required to process each new reward. This update rule can be expressed in a general form:

$$
NewEstimate \leftarrow OldEstimate + StepSize \underbrace{\Big[Target - OldEstimate\Big]}_{error}.
$$

The target is presumed to indicate a desirable direction in which to move, though it may be noisy.

Chapter 2

Markov Decision Process

The general framework of MDPs (representing environments as MDPs) allows us to model virtually any complex sequential decision-making problem under uncertainty in a way that RL agents can interact with and learn to solve solely through experience.

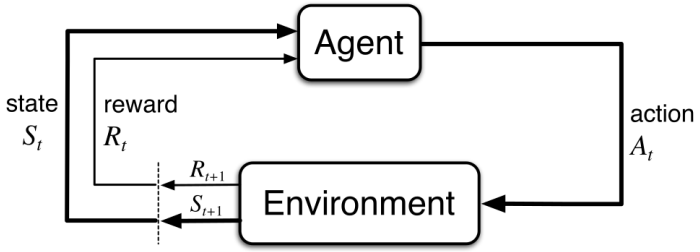


Figure 2.1: The agent-environment interaction in a Markov decision process.

Components of RL:

- An agent
- A policy
- A reward signal: what is good in an immediate sense.
- A value function: what is good in the long run.
- A model of the environment: This allows inferences to be made about how the environment will behave.

Definition 1 (Markov Property) *A state S_t is **Markov** if and only if*

$$P[S_{t+1}|S_t, A_t] = P[S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, ...]$$

- Actions: a mechanism to influence the environment
- State: specific configurations of the environment

Definition 2 (Transition Function)

$$p(s', r|s, a) = P(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)$$

- The way the environment changes as a response to actions is referred to as the state-transition probabilities, or more simply, the transition function, and is denoted by $T(s, a, s')$.
- $\sum_{r\in\mathcal{R}}\sum_{s'\in\mathcal{S}} p(s', r|s, a) = 1, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$
- $p(s'|s, a) = P(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a) = \sum_{r\in\mathcal{R}} p(s', r|s, a)$

Definition 3 (Reward Hypothesis) *All goals can be described by the maximization of expected cumulative reward.*

- That all of what we mean by goals and purposes can be well thought of as athe maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

Definition 4 (Reward Function)

$$\begin{aligned} r(s, a) &= \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] \\ &= \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a) \end{aligned}$$

- $R_t \in \mathcal{R} \subset \mathbb{R}$. Note that negative reward is still reward.
- The expected reward function is defined as a function that takes in a state-action pair.
- It is the expectation of reward at time step t, given the state-action pair in the previous time step.
- It can also be defined as a function that takes a full transition tuple s, a, s'.

$$\begin{aligned} r(s, a, s') &= \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s] \\ &= \sum_{r\in\mathcal{R}} \frac{p(s', r|s, a)}{p(s'|s, a)} \end{aligned}$$

Definition 5 (Discount Factor, γ)

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_t$$

- The sum of all rewards obtained during the course of an episode is referred to as the *return*, G_t.
- Episodic tasks: $G_t = R_{t+1} + R_{t+2} + \cdots + R_T$.
  - $G_t = R_{t+1} + \gamma G_{t+1}$
- Continuing tasks: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$.
  - $\gamma = 0$: myopic evaluation
  - $\gamma = 1$: far-sighted evaluation
  - Uncertainty about the future that may not be fully observed
  - Mathematically convenient to discount rewards.
  - Avoid infinite returns in cyclic Markov processes.

2.1 Policies and Value Functions

- Policies are universal plans, which provides all possible plans for all states.
  - Plans are not enough to fully describe an environment in stochastic environments.
    - What if an agent intends to move right, but ends up going left. Which action does the agent take?
  - Policies are the per-state action prescribtions.
  - Policy can be stochastic or deterministic.
  - A policy is a function that prescribes actions to take for a given non-terminal state.

2.1.1 The State-Value Functions

Almost all reinforcement learning algorithms involve estimating *value functions*-functions of states (or of state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state). The notion of "how good" here is defined in terms of future rewards that can be expected, or, to be precise, in terms of **expected return**. Of course, the rewards the agent can expect to receive in the futhre depend on what actions it will take. Accordingly, value functions are defined with respect to particular ways of action, called policies.

Formally, a policy is a mapping from states to probablities of selecting each possible action. It can be defined

$$\pi(a|s)$$

Definition 6 (The State-Value Function, V) *The state value function $v(s)$ for policy π of an Markov Reward Process is the expected return starting from state s*

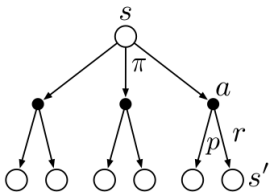
$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k}\Big| S_t = s\right], \quad \forall s \in \mathcal{S}$$

- The value of a state s is the expection over policy π.
- Reward: one-step signal that an agent gets/ Return: total discounted rewards/ Value function: expected return.
- If we are given a policy and the MDP, we should be able to calculate the expected return starting from every single state.

- *Bellman equation*

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v(s_{t+1})|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned}$$

- Starting from state s, the root node at the top, the agent could take any of some set of actions – three are shown in the diagram – based on its policy π. From each of these, the environment could respond with one of several next states, s' (two are shown in the figure), along with a reward, r, depending on its dynamics given by the function p. The Bellman equation averages over all the possibilities, weighting each by its probability of occuring. **It states that the value of the start state must equal (discounted) the value of the expected next state, plus the reward expected along the way**.



- The derivation details are given in Appendix A.1

2.1.2 The Action-Value Function

- Another critical question that we often need to ask is not merely about the value of a state but the value of taking action a at a state s.
- Which action is better under each policy?
- The action-value function, also known as Q-function or $Q^\pi(s, a)$, captures precisely this.
  - The expected return if the agent follows policy π after taking action a in state s.

Definition 7 (The Action-Value Function, Q) *The action-value function $q_\pi(s, a)$ for policy π is the expected return starting from state s, tacking action a under policy π*

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]$$

- The Bellman equation for action values is given by

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r | s, a)[r + \gamma v_\pi(s')] \end{aligned}$$

- Notice that we do not weight over actions because we are interested only in a specific action.
- The state-value function can be expressed by using the action-value function as follows:
- The derivation is given in Appendix A.1

The state-value function can be decomposed as follows:

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\
&= \sum_{g_t} g_t \cdot P[G_t|S_t = s] \\
&= \sum_{g_t} g_t \cdot P[G_t, S_t = s]/P[S_t = s] \\
&= \sum_{g_t} g_t \cdot \sum_a P[G_t, S_t = s, A_t = a]/P[S_t = s] \\
&= \sum_{g_t} g_t \cdot \sum_a \Big[P[G_t|S_t = s, A_t = a]P[S_t = s, A_t = a]\Big]/P[S_t = s] \\
&= \sum_{g_t} g_t \sum_a P[G_t, A_t = a|S_t = s]P[A_t = a|S_t = s] \\
&= \sum_a \sum_{g_t} g_t P[G_t|S_t = s, A_t = a]P[A_t = a|S_t = s] \\
&= \sum_a q_\pi(s, a)\pi(a|s)
\end{aligned}$$

Note that the expectation is parameerized π as written in $\mathbb{E}_\pi$. We can also prove it by the *Law of Total Expectation* [1],

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}[\mathbb{E}_\pi[G_t|S_t = s, A_t = a]] \\
&= \sum_a \mathbb{E}_\pi[G_t|S_t = s, A_t = a]P(A_t = a|S_t = s) \\
&= \sum_a q_\pi(s, a)\pi(a|s)
\end{aligned}$$

An intuitive explantation of this derivation is that the expectation depends on an action $a \sim \pi(a|s)$. We want to estimate the expected total return by the sampled action (this is because the total return is the function of a, implicitly). Then we need to introduce an action variable a and its probability in the expression as the second line of the equation.

2.1.3 The Action-Advantage Function

Definition 8 (The Action-Advantage Function, A)

$$a_\pi(s, a) = q_\pi(s, a) - v_\pi(s)$$

- The advantage function describes how much better it is to take action a instead of following policy π. In other words, the advantage of choosing action a over the default action.

2.2 Optimality

Solving a reinforcement learning task means, roughly, finding a policy that achieves a lot of reward over the long run. For finite MDPs, we can precisely define an optimal policy in the

[1] $\mathbb{E}[\mathbb{E}[X|Y]] = \sum_y \Big[\sum_x x \cdot\ p(X = x|Y)\Big]p(Y = y) = \mathbb{E}[X]$

following way. Value functions define a partial ordering over policies. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. In other words, $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$. There is always at least one policy that is better than or equal to all other policies. This is an *optimal policy.* The optimal state-value function, denoted v_* can be defined as

Definition 9 (Optimal State-Value Function) *The optimal state-value function $v_*(s)$ is the maximum value over all policies*

$$v_*(s) = \max_\pi v_\pi(s), \quad \forall s \in \mathcal{S}.$$

- The optimal state-value function can be obtained as follows:

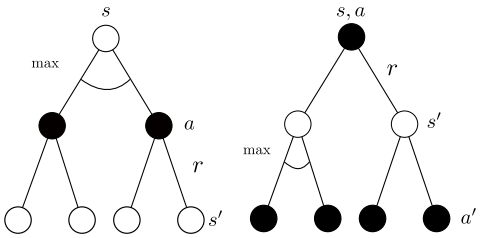
$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{r,s'} p(s', r|s, a) \Big[r + \gamma v_*(s') \Big] \end{aligned}$$

Definition 10 (Optimal Action-Value Function) *The optimal action-value function $q_*(s,a)$ is the maximum value over all policies*

$$q_*(s,a) = \max_\pi q_\pi(s,a), \quad \forall s \in \mathcal{S} \text{ and } a \in \mathcal{A}.$$

- The optimal action-value function can be obtained as follows:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma \max_a q_*(s', a')] \end{aligned}$$



- The optimal value function specifies the best possible performance in the MDP.
- The MDP is solved when we know the optimal value function

Theorem 1 (Optimal Policy Theorem)

$$\pi \geq \pi' \quad \textit{if} \quad v_\pi(s) \geq v_{\pi'}(s), \forall s$$

For any Markov Decision Process:

- *There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$*
- *All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$*
- *All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$*

An optimal policy can be found by maximizing over $q_*(s, a)$,

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \text{argmax}_{a \in \mathcal{A}}\, q_*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy
  - Q-learning: learns Q values first
  - Policy gradient: learns optimal policy without learning Q values

Chapter 3

Dynamic Programming

Let π and π' be any pair of deterministic policies such that, for all $s \in \mathcal{S}$,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s).$$

Then the policy π' must be as good as, or better than π. Equivalently, it must obtain the following inequality for all states $s \in \mathcal{S}$:

$$v'_\pi(s) \geq v_\pi(s).$$

Theorem 2 (Policy Improvement Theorem) *$q_\pi(s, \pi'(s)) \geq v_\pi(s)$, then $v'_\pi(s) \geq v_\pi(s)$.*

Proof.

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \,|\, S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) |\, S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) |\, S_t = s] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots |\, S_t = s] \\
&= \mathbb{E}_{\pi'}[G_t |\, S_t = s] \\
&= v_{\pi'}(s).
\end{aligned}$$

3.1 Policy Evaluation

- Prediction problem: refers to the problem of **evaluating policies** (sipmly, rating policies), of estimating value functions given a policy (learning to predict returns).

- Control problem: problem of **finding optimal policies**. Usually solved by the pattern of generalized policy iteration (GPI), where the competing processes of policy evaluation and policy improvement progressively move policies towards optimality.
- Policy evaluation: refers to algorithms that solve the prediction problem.
  - Iterative policy evaluation.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')].$$

  - Init $v_0(s)$ for all s arbitrarily and to 0 if s is terminal.
  - Bootstrapping: $v_1(s) \to v_2(s) \to \cdots \to v_N(s)$

3.2 Policy Improvement

- Policy improvement: algorithms that make new policies that improve on an original policy by making it greedier than the original with respect to the value function of that original policy. The following approach considers all possible actions at each state and selects the best according to $q_\pi(s, a)$ in a greedy way.

$$\begin{aligned}\pi'(s) &= \operatorname*{argmax}_a q_\pi(s, a) \\ &= \operatorname*{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')].\end{aligned}$$

- The greedy policy takes the action that looks best in the short term according to v_π.

3.3 Value Iteration

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')].$$

Chapter 4

Monte Carlo Methods

Our goal is to estimate value functions and find optimal policies. Monte-Carlo methods require only *experience*–sample sequences of states, actions, and rewards from actual or simulated interaction with an environment.

4.1 Monte Carlo Prediction (Evaluation)

Recall that the value of a state is the expected return starting from that state. An obvious way to estimate it from experience, then, is simply to average the returns observed after visits to that state. As more returns are observed, the average should converge to the expected value. This idea is the core of Monte Carlo methods.

4.1.1 First Visit vs. Every Visit

Suppose we wish to estimate $v_\pi(s)$, the value of a state s under policy π, given a set of episodes obtained by following π and passing through s. Each occurrence of state s in an episode is called a viist to s. The first-visit MC method estimates $v_\pi(s)$ as **the average of the returns** following **first visits** to s, whereas the every-visit MC method averages the returns following all visits to s.

Why do we care about first visit and every visit?

- This is about how to estimate the true value of state by averaging.
- In other words, how to compute the average of the number of visits!

Both are valid approaches, but

- First visit treats each trajectory as an independent and identically distributed sample of $v(s)$.
- First visit uses only one return per state per episode.
- Every visit averages the returns following all visits to a state, even if in the same episode.

4.2 Monte Carlo Control

$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} q_{\pi_*},$$

- π_0: arbitrary policy.
- Policy improvement is done by making the policy greedy with respect to the current value function (action-value function).
- For any action-value function q, the corresponding greedy policy is the one that, for each $s \in \mathcal{S}$, deterministically chooses an action with maximal action-value:

$$\pi(S) = \operatorname*{argmax}_a q(s, a).$$

- Policy improvement then can be done by constructing each π_{k+1} as the greedy policy with respect to q_{π_k}.
- By Theorem 2, each π_{k+1} is uniformly better than π_k or just as good as π_k.

4.3 On-Policy vs. Off-Policy

- On-policy: learn about the optimal policy by executing the policy and evaluting and improving it.
  - Learning to be great by itself.
- Off-policy: learn about the optimal policy by using data generated by another policy.
  - Learning from others.

The more episodes are collected, the better because the estimates of the functions will be. However, there is a problem. If the algorithm for policy improvement always updates the policy greedily, meaning it takes only actions leading to immediate reward, actions and states not on the greedy path will not be sampled sufficiently, and potentially better rewards would stay hidden from the learning process.

Essentially, we are forced to make a choice between making the best decision given the current information or start exploring and finding more information. This is also known as the Exploration vs. Exploitation Dilemma.

We are looking for something like a middle ground between those. Full-on exploration would mean that we would need a lot of time to collect the needed information, and full-on exploitation would make the agent stuck into a local reward maximum. There are two approaches to ensure all actions are sampled sufficiently called on-policy and off-policy methods.

4.3.1 On Policy

On-policy methods solve the exploration vs exploitation dilemma by including randomness in the form of a policy that is soft, meaning that non-greedy actions are selected with some probability.

These policies are called ϵ-greedy policies as they select random actions with an ϵ probability and follow the optimal action with $1-\epsilon$ probability

Since the probability of selecting from the action space randomly is ϵ, the probability of selecting any particular non-optimal (non-greedy) action is $\epsilon/|\mathcal{A}(s)|$. The probability of following the optimal action will always be slightly higher, however, because we have a $1-\epsilon$ probability of selecting it outright and $\epsilon/|\mathcal{A}(s)|$ probability of selecting it from sampling the action space [1]

$$P(a_t^*) = 1 - \epsilon + \epsilon/|\mathcal{A}(s)|.$$

It is also worth noting that because the optimal action will be sampled more often than the others making on-policy algorithms will generally converge faster but they also have the risk of trapping the agent into a local optimum of the function.

4.3.2 Off Policy

All learning contol methods face a dilemma: They seek to learn action values conditional on subsequent *optimal* behavior, but they need to behave non-optimally in order to explore all actions (to find the optimal actions). How can they learn about the optimal policy while behaving according to an exploratory policy? The on-policy approach in the preceding section is actually a compromise. It learns action values not for the optimal policy, but for a near-optimal policy that still explores. A more straightforward approach is to use two policies, one that is learned about and that becomes the optimal policy, and one that is more exploratory and is used to generate behavior. The policy being learned about is called the *target policy*, and the policy used to generate behavior is called the *behavior policy*. In this case we say that learning is from data "off" the target policy, and the overall process is termed *off-policy learning*.

Given a starting state S_t, the probability of the subsequent state-action trajectory, $A_t, S_{t+1}, A_{t+1}, \cdots, S_T$, occuring under any policy π is

$$\begin{aligned} P(A_t, S_{t+1}, A_{t+1}, \cdots, S_T) &= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1})\cdots(S_T|S_{T-1}, A_{T-1}) \\ &= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k), \end{aligned}$$

where p here is the state-transition probability function. Thus, the relative probability of the trajectory under the target and behavior policies is

$$\begin{aligned} \rho_{t:T-1} &= \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} \\ &= \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}. \end{aligned}$$

The importance sampling ratio ends up depending only on the two polices and the sequence, not on the MDP (state-transition probability).

Recall that we wish to estimate the expected returns under the target policy, but all we have are returns G_t from the behavior policy, which reulsts in

$$\mathbb{E}[G_t|S_t = s] = v_b(s).$$

[1] Since the greedy (or optimal) action can be selected by either 1-ϵ or $\epsilon/|\mathcal{A}(s)|$, $a_1 = \epsilon/\mathcal{A}(s), a_2 = \epsilon/\mathcal{A}(s), \cdots, a_{best} = 1 - \epsilon + \epsilon/\mathcal{A}(s), a_n = \epsilon/\mathcal{A}(s)$

So, the ratio $\rho_{t:T-1}$ transforms the returns

$$\mathbb{E}[\rho_{t:T-1}G_t|S_t = s] = v_\pi(s).$$

4.4 Temporal-Difference Learninig

TD learning is a combination of Monte-Carlo method and dynamic programming ideas. It learns directly from raw experience without a model of the environment's dynamics.

One of the main drawbacks of MC methods is the fact that the agent has to wait until the end of an episode when it can obtain the actual G_t before it can update the state-value function estimate $V_T(S_t)$.

Constant-α MC:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

One-step TD (or TD(0)):

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underbrace{R_{t+1} + \gamma V(S_{t+1}) - V(S_t)}_{\text{TD error}}\Big]$$

4.4.1 Q-learning: Off-policy TD Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

- Off-policy. For instance, we can sample an action A_t from a ϵ-greedy policy.
- Update Q for each step.
- Maximization bias in Q-learning

4.4.2 Sarsa: On-policy TD Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

- On-policy agorithm, unlike Q-learning, we sample both A_t and A_{t+1} from another policy like ϵ-greedy policy.

4.4.3 Double Q-learning

Randomly select Q_1 or Q_2.

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q_2(S_{t+1}, a^*) - Q_1(S_t, A_t)\Big],$$

where a^* is

$$a^* = \operatorname*{argmax}_a Q_1(s', a).$$

$$Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q_1(S_{t+1}, a^*) - Q_2(S_t, A_t)\Big],$$

where a^* is

$$a^* = \operatorname*{argmax}_a Q_2(s', a).$$

Chapter 5

Deep Reinforcement Learning

Three challenges in DRL:

- Sequential feedback
- Evaluative feedback
- Sampled feedback

Select:

1. A value function to approximate.
2. A neural network architecture
    - e.g., value (single output node) or action (multiple output nodes)
3. What to optimize
4. Policy evaluation algorithm
5. Exploration strategy
6. A loss function
7. Optimization method

Some considerations:

- Non-stationary target
- Data correlated with time
    - Samples in a batch are correleated, given that most of these samples come from the same trajectory and policy. It breaks the i.i.d. assumptions.

Chapter 6

Deep Q-Network

Chapter 7

Policy Gradient

We will use a gradient ascent algorithm:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[r(\tau)] \\ &= \int \pi_\theta(\tau) r(\tau) \end{aligned}$$

It is a expected reward under the policy π_θ.

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

Note that by using REINFORCE algorithm which can be expressed as follows:

$$\nabla_\theta \pi_\theta(\tau) = \pi_\theta \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \pi_\theta \nabla_\theta \log \pi_\theta(\tau)$$

We can express $J(\theta)$ as follows:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int \nabla_\theta \pi_\theta(\tau) r(\tau) = \int \pi_\theta \nabla_\theta \log \pi_\theta(\tau) r(\tau) \\ &= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \end{aligned}$$

Note that $\nabla_\theta \log \pi_\theta(\tau)$ is the maximum loglikelihood of trajectory, because gradient is the maximum direction of the function. This expectation can be estimated by Monte-Carlo method.

We just sample trajectories using current policy and adjust the likelihood of trajectories by episodic rewards.

$$\pi_\theta(\tau) = p_\theta(s_1, a_1, s_2, a_2, \cdots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

Note that the product term looks causing gradient explosion or vanishing problems for a long sequence problem, but it turns into a multiplication as below by the log-derivative trick, so it can avoid the issues.

In general, the agent has no access to $p(s_1)$ and $p(s_{t+1}|s_t, a_t)$ (we don't know trainsition robability.).

$$\begin{aligned}\nabla_\theta \log \pi_\theta(\tau) &= \nabla_\theta \left[\log p(s_1) + \sum_{t=1}^{T} (\log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)) \right] \\ &= \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \\ r(\tau) &= \sum_{t=1}^{T} r(s_t, a_t)\end{aligned}$$

By using the above equations, we can re-express the gradient of the cost function as

$$\begin{aligned}\nabla_\theta J(\theta) &= \int \nabla_\theta \pi_\theta(\tau) r(\tau) = \int \pi_\theta \nabla_\theta \log \pi_\theta(\tau) \\ &= \mathbb{E}_{\tau \sim \pi_\theta(\tau)} [\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \\ &= \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left(\sum_{t=1}^{T} r(s_t, a_t) \right) \right]\end{aligned}$$

By using Monte-Carlo method, we can replace the expectation by sampling multiple trajectories in practice:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right) \right]$$

7.0.1 Causality

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right] \\ &= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \left[\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \sum_{t'=1}^{T} r(s_{i,t'}, a_{i,t'}) \right] \\ &= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \left[\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right] \quad \text{by causality}\end{aligned}$$

7.1 Natural Policy Gradient

reference: Nathan Ratliff, Information Geometry and Natural Gradients.

7.1.1 KL-divergence between perturbed distributions

Let $p(x; \theta)$ be some family of probability distributions over x parameterized by a vector of real numbers θ. We're interested in knowing how much the distribution changes when we perturb

the parameter vector from a fixed θ_t to some new value $\theta_t + \delta\theta$. As a measure of change in probability distribution, we can use the KL-divergence measure. Specifically, we want to measure $D_{KL}(p(x;\theta_t)||p(x;\theta_t+\delta\theta)$, but we want to write it in a form amenable to the gradient-based update formulation. We can do this by taking it's second-order Taylor expansion around θ_t. During the derivation, we'll find that a lot of terms in the expansion disappear leaving us with a very simple expression that's perfect for our purposes.

Looking first at the full KL-divergence, we see that the term we want to

$$\begin{aligned} D_{KL}(p(x;\theta_t)||p(x;\theta_t+\delta\theta)) &= \int p(x;\theta_t)\log\frac{p(x;\theta_t)}{p(x;\theta_t+\delta\theta)}dx \\ &= \int p(x;\theta_t)\log p(x;\theta_t)dx - \int p(x;\theta_t)\log p(x;\theta_t+\delta\theta)dx \end{aligned}$$

Note that the second-order Taylor series expansion is

$$f(\theta) \approx f(\theta_t) + \nabla f(\theta_t)^T\delta\theta + \frac{1}{2}\delta\theta^T\nabla^2 f(\theta_t)\delta\theta,$$

where $\theta = \theta_t + \delta\theta$, or equivalently $\delta\theta = \theta - \theta_t$. Applying that expansion to the pertinent term in the KL-divergence expression, we get

$$\log p(x;\theta_t+\delta\theta) \approx \log p(x;\theta_t) + \left(\frac{\nabla p(x;\theta_t)}{p(x;\theta_t)}\right)^T\delta\theta + \frac{1}{2}\delta\theta^T(\nabla^2\log p(x;\theta_t))\delta\theta.$$

Plugging this second-order Taylor expansion back into the above expression for the D_{KL} gives

$$\begin{aligned} &D_{KL}(p(x;\theta_t)||p(x;\theta_t+\delta\theta)) \\ &\quad= \int p(x;\theta_t)\log p(x;\theta_t)dx - \int p(x;\theta_t)\log p(x;\theta_t+\delta\theta)dx \\ &\quad\approx \int p(x;\theta_t)\log p(x;\theta_t)dx \\ &\qquad - \int p(x;\theta_t)\left(\log p(x;\theta_t) + \left(\frac{\nabla p(x;\theta_t)}{p(x;\theta_t)}\right)^T\delta\theta + \frac{1}{2}\delta\theta^T(\nabla^2\log p(x;\theta_t))\delta\theta\right)dx \\ &\quad= \int p(x;\theta_t)\log\frac{p(x;\theta_t)}{p(x;\theta_t)}dx - \underbrace{\int p(x;\theta_t)\left(\frac{\nabla p(x;\theta_t)}{p(x;\theta_t)}\right)^T dx}_{=0} - \frac{1}{2}\delta\theta^T\left(\int p(x;\theta_t)\nabla^2\log p(x;\theta_t)dx\right)\delta\theta \\ &\quad= -\frac{1}{2}\delta\theta^T\left(\int p(x;\theta_t)\nabla^2\log p(x;\theta_t)dx\right)\delta\theta. \end{aligned}$$

$\int \nabla p(x;\theta_t)$ is zero since

$$\int \nabla p(x;\theta_t) = \nabla\int p(x;\theta_t) = \nabla 1 = 0$$

The Hessian can be computed as follows:

$$\begin{aligned}\frac{\partial^2}{\partial\theta_t^i\partial\theta_t^j}[\log p(x;\theta_t)] &= \frac{\partial}{\partial\theta_t^i}\left(\frac{\frac{\partial}{\partial\theta_t^j}p(x;\theta_t)}{p(x;\theta_t)}\right)\\ &= \frac{p(x;\theta_t)\frac{\partial^2}{\partial\theta_t^i\partial\theta_t^j}p(x;\theta_t) - \frac{\partial}{\partial\theta_t^i}p(x;\theta_t)\frac{\partial}{\partial\theta_t^j}p(x;\theta_t)}{p(x;\theta_t)^2}\\ &= \frac{1}{p(x;\theta_t)}\frac{\partial^2}{\partial\theta_t^i\partial\theta_t^j}p(x;\theta_t) - \left(\frac{\frac{\partial}{\partial\theta_t^i}p(x;\theta_t)}{p(x;\theta_t)}\right)\left(\frac{\frac{\partial}{\partial\theta_t^j}p(x;\theta_t)}{p(x;\theta_t)}\right).\end{aligned}$$

The second term is an element of the outer product between $\nabla\log p(x;\theta_t)$ and itself. In matrix form, this becomes

$$\nabla^2\log p(x;\theta_t) = \frac{1}{p(x;\theta_t)}\nabla^2 p(x;\theta_t) - \nabla\log p(x;\theta_t)\nabla\log p(x;\theta_t)^T.$$

Finally, we get

$$\begin{aligned}&D_{KL}(p(x;\theta_t)||p(x;\theta_t+\delta\theta))\\ &\quad\approx -\frac{1}{2}\delta\theta^T\int p(x;\theta_t)\nabla^2\log p(x;\theta_t)dx\delta\theta\\ &\quad= \frac{1}{2}\delta\theta^T\left(\int\nabla^2\log p(x;\theta_t)dx\right)\delta\theta\\ &\qquad+\frac{1}{2}\delta\theta^T\left(\int p(x;\theta_t)[\nabla\log p(x;\theta_t)\nabla\log p(x;\theta_t)^T]dx\right)\delta\theta\\ &\quad= \frac{1}{2}\delta\theta^T\underbrace{\left(\int p(x;\theta_t)[\nabla\log p(x;\theta_t)\nabla\log p(x;\theta_t)^T]dx\right)}_{G(\theta_t)}\delta\theta.\end{aligned}$$

The central matrix here $G(\theta_t)$ is known as the **Fisher Information matrix** and can has been thoroughly studied within the field of Information Geometry as the natural Riemannian structure on a manifold of probability distributions. As such it defines a natural norm on perturbations to probability distributions, which was our original motivation for examning the second-order Taylor expansion of the KL-divergence in the first place.

$$\theta_{t+1} = \theta_t - \eta_t G(\theta_t)^{-1}\nabla f(\theta_t).$$

7.2 Proximal Policy Optimization

PPO objective is

$$\theta_{k+1} = \underset{\theta}{\operatorname{argmax}}\underset{s,a\sim\pi_{\theta_k}}{\mathbb{E}}[L(s,a,\theta_k,\theta)],$$

where L is given by

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a),\ \operatorname{Clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)},1-\varepsilon,1+\varepsilon\right)A^{\pi_{\theta_k}}(s,a)\right).$$

Roughly, ε is a hyperparameter which says how far away the new policy is allowed to go from the old one. A simpler expression of the above expression is

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta\left(a|s\right)}{\pi_{\theta_\mathrm{k}}\left(a|s\right)}A^{\pi_{\theta_k}}(s,a), g(\varepsilon, A^{\pi_{\theta_k}}(s,a))\right), \tag{7.1}$$

where

$$g(\varepsilon, A) = \begin{cases}(1+\varepsilon)A & A \geq 0\\ (1-\varepsilon)A & A < 0.\end{cases} \tag{7.2}$$

Positive Advantage: Suppose the advantage for that state-action pair is positive, in which case its contribution to the objective reduces to

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta\left(a|s\right)}{\pi_{\theta_\mathrm{k}}\left(a|s\right)}, 1+\varepsilon\right)A^{\pi_{\theta_k}}(s,a). \tag{7.3}$$

Because the advantage is positive, the objective will increase if the action becomes more likely that is, if $\pi_\theta(a|s)$ increases. But the min in this term puts a limit to how much the objective can increase. Once $\pi_\theta(a|s) > (1+\epsilon)\pi_{\theta_k}(a|s)$, the min kicks in and this term hits a ceiling of $(1+\epsilon)A^{\pi_{\theta_k}}(s,a)$. Thus, the new policy does not benefit by going far away from the old policy.

Negative Advantage: Suppose the advantage for that state-action pair is negative, in which case its contribution to the objective reduces to

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta\left(a|s\right)}{\pi_{\theta_\mathrm{k}}\left(a|s\right)}, 1-\varepsilon\right)A^{\pi_{\theta_k}}(s,a). \tag{7.4}$$

Because the advantage is negative, the objective will increase if the action becomes less likely—that is, if $\pi_\theta(a|s)$ decreases. But the max in this term puts a limit to how much the objective can increase. Once $\pi_\theta(a|s) < (1-\epsilon)\pi_{\theta_k}(a|s)$, the max kicks in and this term hits a ceiling of $(1-\epsilon)A^{\pi_{\theta_k}}(s,a)$. Thus, again, the new policy does not benefit by going far away from the old policy.

What we have seen so far is that clipping serves as a regularizer by removing incentives for the policy to change dramatically, and the hyperparameter ϵ corresponds to how far away the new policy can go from the old while still profiting the objective.

Chapter 8

Search Algorithm

Reference: John Levine

8.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a search technique in the field of Artificial Intelligence (AI). It is a probabilistic and heuristic driven search algorithm that combines the classic tree search implementations alongside machine learning principles of reinforcement learning.

It consists of four phases:

- Tree traversal
- Node expansion
- Rollout
- Backpropagation

In tree search, there's always the possibility that the current best action is actually not the most optimal action. In such cases, MCTS algorithm becomes useful as it continues to evaluate other alternatives periodically during the learning phase by executing them, instead of the current perceived optimal strategy. This is known as the "exploration-exploitation trade-off". It exploits the actions and strategies that is found to be the best till now but also must continue to explore the local space of alternative decisions and find out if they could replace the current best.

Exploration helps in exploring and discovering the unexplored parts of the tree, which could result in finding a more optimal path. In other words, we can say that exploration expands the tree's breadth more than its depth. Exploration can be useful to ensure that MCTS is not overlooking any potentially better paths. But it quickly becomes inefficient in situations with large number of steps or repetitions. In order to avoid that, it is balanced out by exploitation. Exploitation sticks to a single path that has the greatest estimated value. This is a greedy approach and this will extend the tree's depth more than its breadth. In simple words, UCB formula applied to trees helps to balance the exploration-exploitation trade-off by periodically exploring relatively unexplored nodes of the tree and discovering potentially more optimal paths than the one it is currently exploiting.

For this characteristic, MCTS becomes particularly useful in making optimal decisions in Artificial Intelligence (AI) problems.

8.1.1 Tree Traversal

In this process, the MCTS algorithm traverses the current tree from the root node using a specific strategy. The strategy uses an evaluation function to optimally select nodes with the highest estimated value. MCTS uses the Upper Confidence Bound (UCB) formula applied to trees as the strategy in the selection process to traverse the tree. It balances the exploration-exploitation trade-off. During tree traversal, a node is selected based on some parameters that return the maximum value. The parameters are characterized by the formula that is typically used for this purpose is given below:

$$\text{UCB1}(S_i) = \underbrace{\bar{V}_i}_{Exploitation} + \underbrace{C\sqrt{\frac{\ln N}{n_i}}}_{Exploration},$$

- C is a balancing factor between exploitation and exploration.
- N the number of visit of parent node
- n_k the number of visit of node k

Let's say there are two successor nodes. One is visited more times than another one. Then, it means it is exploited more times than the other one. Thus, exploration term of the less exploited one would be higher than the highly visited one. By computing the UCB1 socre, the agent chooses a successor node with higher UCB1 score.

8.1.2 Expanasion

Expand successor node

8.1.3 Rollout (Random Simulation)

Simulation is completely random. In other words, we don't know how an agent reacts to an environment, so each successor state, the agent randomly decides which action to do till the termination.

8.1.4 Backpropagation

Backpropagate rewards, and the number of visit at a node.

- $t := t + 1$: total score
- $n_k := n_k + 1$

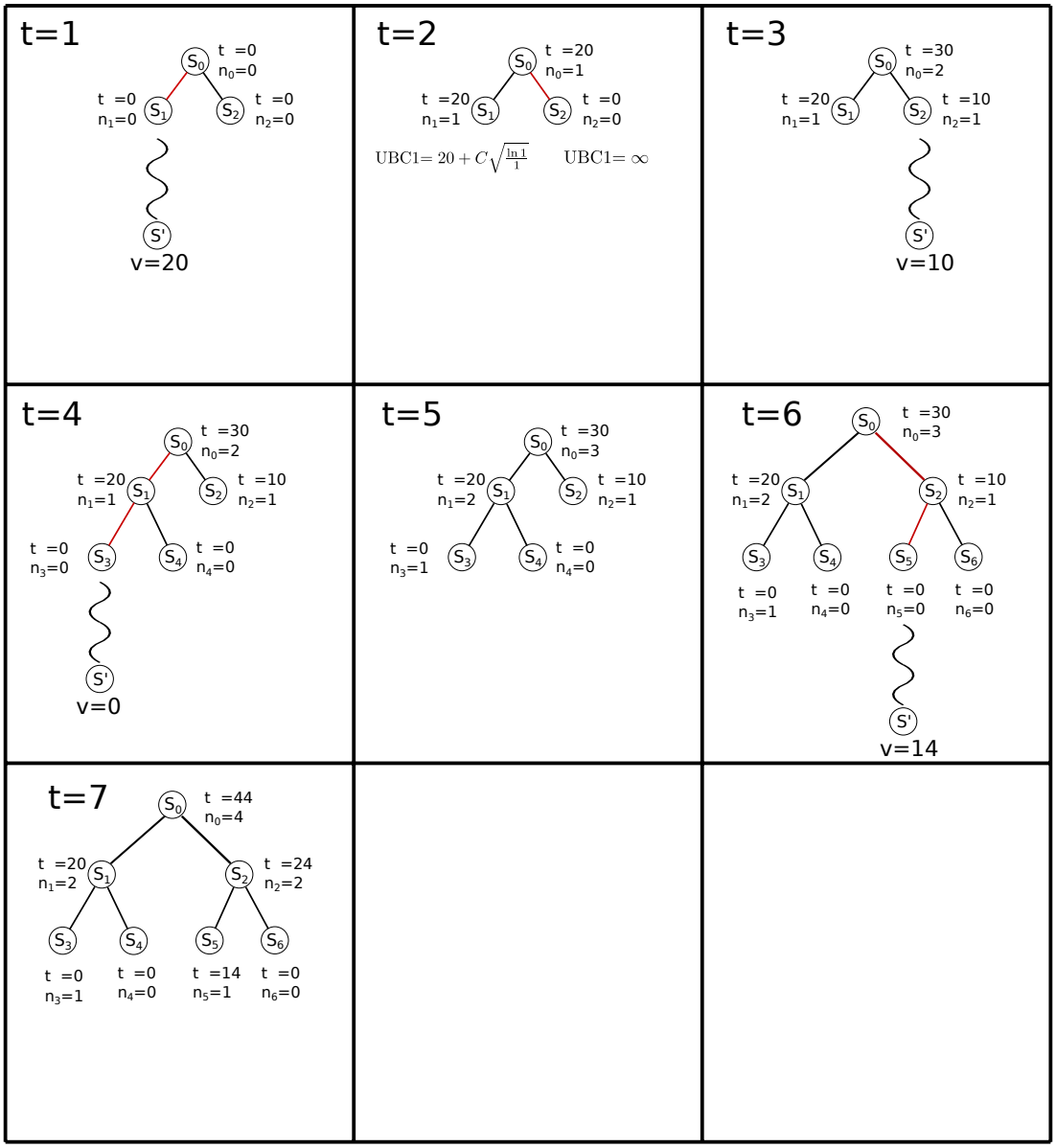


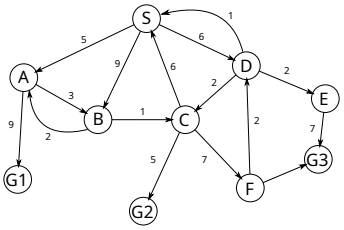
Figure 8.1: MCTS example

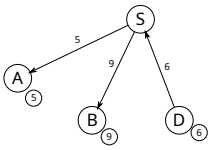
8.2 Uniform Cost Search

You have three goal states, G_1, G_2, G_3. Your goal is to reach one of them. UCS is cheapest first search algorithm.

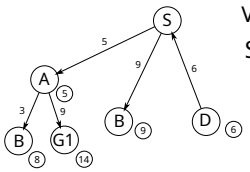
8.3 A^* Search

Each number next to the nodes is called A^* score, which is an estimate of cost to get to one of states.

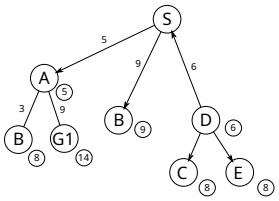




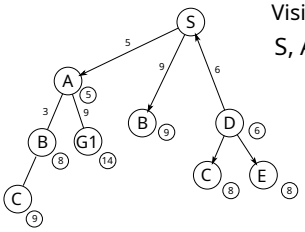
Visited
S



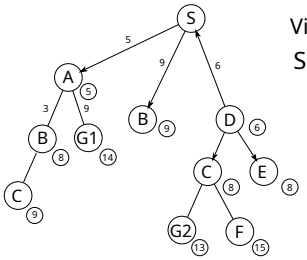
Visited
S, A



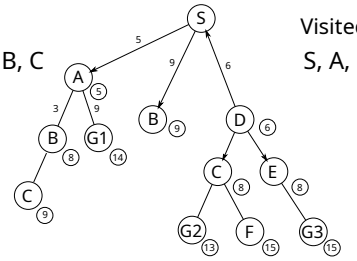
Visited
S, A, D



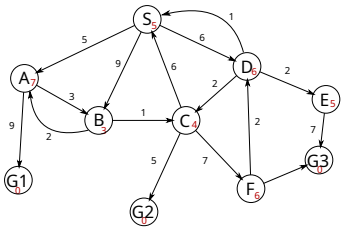
Visited
S, A, D, B

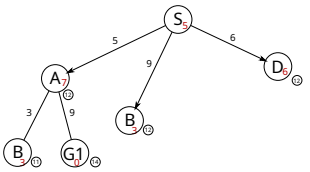


Visited
S, A, D, B, C



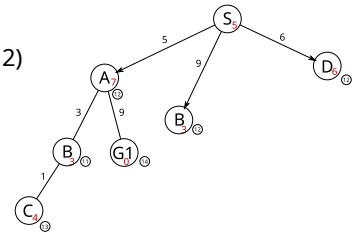
Visited
S, A, D, B, C, E





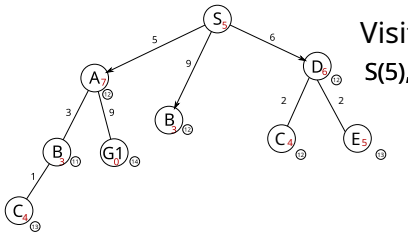
Visited

S(5), A(12)



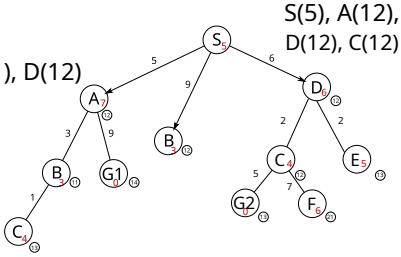
Visited

S(5), A(12), B(11)



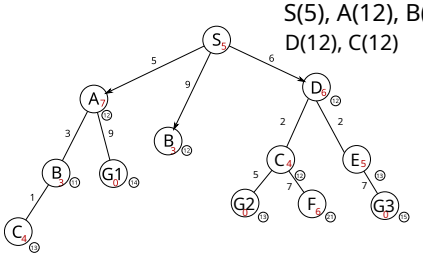
Visited

S(5), A(12), B(11), D(12)



Visited

S(5), A(12), B(11),
D(12), C(12)



Visited

S(5), A(12), B(11),
D(12), C(12)

Bibliography

Appendix

A.1 Bellman Equation

Bellman equation can be derived as follows:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1}|S_t = s] + \mathbb{E}_\pi[\gamma G_{t+1}|S_t = s], \quad \text{By Linearity of Expectation.} \\
&= \sum_{r_{t+1}} r_{t+1} P(R_{t+1}|S_t = s) + \mathbb{E}_\pi[\gamma G_{t+1}|S_t = s] \\
&= \sum_{r_{t+1}} r_{t+1} \sum_a P(R_{t+1}|S_t = s, A_t = a) P(A_t = a|S_t = s) + \mathbb{E}_\pi[\gamma G_{t+1}|S_t = s] \\
&= \sum_a \sum_{r_{t+1}} r_{t+1} \sum_{s'} P(R_{t+1}, S_{t+1} = s'|S_t = s, A_t = a) P(A_t = a|S_t = s) + \mathbb{E}_\pi[\gamma G_{t+1}|S_t = s] \\
&= \sum_a \sum_r r \sum_{s'} P(s', r|s, a)\pi(a|s) + \mathbb{E}_\pi[\gamma G_{t+1}|S_t = s] \\
&= \sum_a \sum_{s'} \sum_r r P(s', r|s, a)\pi(a|s) + \mathbb{E}_\pi[\gamma G_{t+1}|S_t = s] \\
&= \sum_a \sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'} \pi(a|s) + \gamma \mathbb{E}_\pi[G_{t+1}|S_t = s] \\
&= \sum_a \sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'} \pi(a|s) + \gamma \sum_a \mathbb{E}_\pi[G_{t+1}|S_t = s, A_t = a] P(A_t|S_t) \\
&= \sum_a \pi(a|s) \left[\sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'} + \gamma \mathbb{E}_\pi[G_{t+1}|S_t = s, A_t = a] \right] \\
&= \sum_a \pi(a|s) \left[\sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'} + \gamma \sum_{g_{t+1}} g_{t+1} P(G_{t+1}|S_t = s, A_t = a) \right] \\
&= \sum_a \pi(a|s) \left[\sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'} + \gamma \sum_{g_{t+1}} g_{t+1} \frac{P(G_{t+1}, S_t = s, A_t = a)}{P(S_t = s, A_t = a)} \right] \\
&= \sum_a \pi(a|s) \left[\sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'} + \gamma \sum_{g_{t+1}} g_{t+1} \frac{\sum_{s'} P(G_{t+1}, S_t = s, S_{t+1} = s', A_t = a)}{P(S_t = s, A_t = a)} \right] \\
&= \sum_a \pi(a|s) \left[\sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'} + \gamma \sum_{g_{t+1}} g_{t+1} \frac{\sum_{s'} P(G_{t+1}|s, s', a) P(s, s', a)}{P(s, a)} \right] \\
&= \sum_a \pi(a|s) \left[\sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'} + \gamma \sum_{g_{t+1}} g_{t+1} \sum_{s'} P(G_{t+1}|s, s', a) P(s'|s, a) \right] \\
&= \sum_a \pi(a|s) \left[\sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'} + \gamma \sum_{s'} P(s'|s, a) \sum_{g_{t+1}} g_{t+1} P(G_{t+1}|s') \right] \quad \text{by Markov Property} \\
&= \sum_a \pi(a|s) \left[\sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'} + \gamma \sum_{s'} \mathcal{P}^a_{ss'} v_\pi(s') \right] \\
&= \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma v_\pi(s') \right] \\
&= \sum_a \pi(a|s) \sum_{r,s'} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big]
\end{aligned}
$$

Or simply,

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) q(s,a) \\ &= \sum_a \pi(a|s) \sum_{r,s'} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big] \end{aligned}$$

Reference

Similarly,

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} | S_t = s, A_t = a] + \gamma \mathbb{E}_\pi[G_{t+1} | S_t = s, A_t = a] \\ &= \sum_r r p(r|s,a) + \gamma \mathbb{E}_\pi[G_{t+1} | S_t = s, A_t = a] \\ &= \sum_r r \sum_{s'} p(s', r|s, a) + \gamma \mathbb{E}_\pi[G_{t+1} | S_t = s, A_t = a] \\ &= \sum_{s',r} r p(s', r|s, a) + \gamma \mathbb{E}[\mathbb{E}_\pi[G_{t+1} | S_t = s, A_t = a, R_{t+1}, S_{t+1}]] \quad \text{By Law of Total Expectation.} \\ &= \sum_{s',r} r p(s', r|s, a) + \gamma \sum_{s',r} \mathbb{E}_\pi[G_{t+1} | S_t = s, A_t = a, R_{t+1} = r, S_{t+1} = s'] p(s', r|s, a) \\ &= \sum_{s',r} p(s', r|s, a) [r + \gamma \mathbb{E}_\pi[G_{t+1} | S_t = s, A_t = a, R_{t+1} = r, S_{t+1} = s'] \\ &= \sum_{s',r} p(s', r|s, a) [r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \quad \text{By Markov Property.} \\ &= \sum_{s',r} p(s', r|s, a) [r + \gamma v_\pi(s')] \end{aligned}$$

B.2 Importance Sampling

You have two distributions, $P(A)$ and $P(B)$, and you have a sequence sampled from A. You can estimate an expectation of A,

$$\mathbb{E}[A] = \sum p(a) h(a).$$

Can we use the above equation for estimating an expectation of B? Yes.

$$\mathbb{E}[B] = \sum \frac{p(b)}{p(a)} h(a).$$

The ratio $\frac{p(b)}{p(a)}$ tells us how likely to observe some results under $p(b)$ compared to $p(a)$.

Useful factorization of conditional probability:

$$P[A, B|C] = \frac{P[A, B, C]}{P[C]} \tag{1}$$

$$= \frac{P[A, B, C]}{P[C]} \frac{P[B, C]}{P[B, C]} \tag{2}$$

$$= \frac{P[A, B, C]}{P[B, C]} \frac{P[B, C]}{P[C]} \tag{3}$$

$$= P[A|B, C]P[B|C] \tag{4}$$

C.3 Fisher Information

Suppose we have a model parameterized by parameter vector θ that models a distribution $p(x; \theta)$. In frequentist statistics, the way we learn θ is to maximize the likelihood of $p(x; \theta)$. To assess the goodness of our estimate of θ we define a **score function** as follows:

$$f(\theta) = \nabla_\theta \log p(x; \theta).$$

The expected value of score function is zero.

$$\begin{aligned}
\mathbb{E}_{p(x;\theta)}[f(\theta)] &= \mathbb{E}_{p(x;\theta)}[\nabla_\theta \log p(x; \theta)] \\
&= \int p(x; \theta) \nabla_\theta \log p(x; \theta) dx \\
&= \int p(x; \theta) \frac{\nabla_\theta p(x; \theta)}{p(x; \theta)} dx \\
&= 0
\end{aligned}$$

The covariance of the score function is given by

$$\text{Cov}[f(\theta), f(\theta)] = \mathbb{E}_{p(x;\theta)}[(f(\theta) - 0)(f(\theta) - 0)^T] = \text{Var}[f(\theta), f(\theta)].$$

This the definition of Fisher information and it can be written

$$F = \mathbb{E}_{p(x;\theta)}[\nabla \log p(x; \theta) \nabla \log p(x; \theta)^T].$$

Empirically,

$$F = \frac{1}{N} \sum_{i=1}^{N} \nabla \log p(x; \theta) \nabla \log p(x; \theta)^T.$$

D.4 Score Function

In statistics, *the score (or informant) is the gradient of the log-likelihood function with respect to the parameter vector.*

- Evaluated at a particular point of the parameter vector, the score indicates the **steepness of the log-likelihood function and thereby the sensitivity to infinitesimal changes to the parameter values**.

- If the log-likelihood function is continuous over the parameter space, the score will **vanish at a local maximum or minimum**; this fact is used in maximum likelihood estimation to find the parameter values that maximize the likelihood function.

Since the score is a function of the observations that are subject to sampling error, it lends itself to a test statistic known as score test in which the parameter is held at a particular value. Further, the ratio of two likelihood functions evaluated at two distinct parameter values can be understood as a definite integral of the score function.[2]

E.5 Incremental Monte-Carlo

Incremental Mean:

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j \tag{5}$$

$$= \frac{1}{k}\Big(x_k + (k-1)\frac{1}{(k-1)}\sum_{j=1}^{k-1} x_j\Big) \tag{6}$$

$$= \frac{1}{k}\Big(x_k + (k-1)\mu_{k-1}\Big) \tag{7}$$

$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1}) \tag{8}$$

Incremental MC:

- $N_{n+1}(S_t^n) = N_n(S_t^n) + 1$ for rest $N_{n+1}(s) = N_n(s)$
- $V_{n+1}(S_t) = V_n(S_t) + \frac{G_{t:T}^n - V_n(S_t)}{N_n(S_t)}$ for rest $V_{n+1}(s) = V_n(s)$
- $V_{n+1}(S_t) = V_n(S_t) + \alpha(G_{t:T}^n - V_n(S_t))$ for rest $V_{n+1}(s) = V_n(s)$

F.6 Derivative of Softmax

Softmax function is given by

$$S(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \quad \text{for } i = 1, \ldots, K$$

The derivative of softmax function is

$$\frac{\partial S_i}{\partial x_j} = \begin{cases} S_i(1 - S_j) & \text{if } i = j \\ -S_j S_i & \text{if } i \neq j \end{cases}$$

- Diagoal elements: $S_i(1 - S_j)$
- Off-diagonal elements: $-S_j S_i$:

The Jacobian matrix $(j \times i)$ for softmax is

$$\frac{\partial S}{\partial x} = \begin{bmatrix} \frac{\partial S_1}{\partial x_1} & \frac{\partial S_1}{\partial x_2} & \dots & \frac{\partial S_1}{\partial x_K} \\ \frac{\partial S_2}{\partial x_1} & \frac{\partial S_2}{\partial x_2} & \dots & \frac{\partial S_K}{\partial x_K} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial S_K}{\partial x_1} & \frac{\partial S_K}{\partial x_2} & \dots & \frac{\partial S_K}{\partial x_K} \end{bmatrix}$$

The matrix can be expressed as follows:

$$\begin{bmatrix} S(x_1) - S(x_1)S(x_1) & \dots & 0 - S(x_1)S(x_N) \\ \dots & S(x_j) - S(x_j)S(x_i) & \dots \\ 0 - S(x_N)S(x_1) & \dots & 0 - S(x_N)S(x_N) \end{bmatrix} =$$
$$\begin{bmatrix} S(x_1) & \dots & 0 \\ \dots & S(x_j) & \dots \\ 0 & \dots & S(x_N) \end{bmatrix} - \begin{bmatrix} S(x_1)S(x_1) & \dots & S(x_1)S(x_N) \\ \dots & S(x_j)S(x_i) & \dots \\ S(x_N)S(x_1) & \dots & S(x_N)S(x_N) \end{bmatrix}$$

This can be

```
np.diag(S) - np.outer(S, S)
```

G.7 Policy Gradient Theorem

Function approximation is essential to reinforcement learning, but the standard approach of approximating a value function and determining a policy from it has so far proven theoretically intractable. In this paper we explore an alternative approach in which the policy is explicitly represented by its own function approximator, independent of the value function, and is updated according to the gradient of expected reward with respect to the policy parameters. Williams's REINFORCE method and actor-critic methods are examples of this approach.

We consider the standard reinforcement learning framework, in which a learning agent interacts with a Markov decision process (MDP).

- The state, action, and reward at each time $t \in \{0, 1, 2, \cdots\}$ are denoted $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$, and $r_t \in \mathbb{R}$.
- The environment's dynamics are characterized by state transition probabilities and expected rewards:

$$\mathcal{P}^a_{ss'} = P(s_{t+1} = s' | s_t = s, a_t = a)$$
$$\mathcal{R}^a_s = \mathbb{E}(r_{t+1} | s_t = s, a_t = a), \forall s, s' \in \mathcal{S}, a \in \mathcal{A}$$

With function approximation, two ways of formultaing the agent's objective are useful: One is the average reward formulation, in which policies are ranked according to their long-term expected reward per-step, $\rho(\pi)$:

$$\rho(\pi) = \lim_{n \to \infty} \frac{1}{n} \mathbb{E}[r_1 + r_2, \cdots, + r_n | \pi] = \sum_s d^\pi(s) \sum_a \pi(a|s) \mathcal{R}^a_s,$$

where $d^\pi(s) = \lim_{t\to\infty} P(s_t = s|s_0, \pi)$ is the stationary distribution of states under π, which we assume exists and is independent of s_0 for all policies. Imagine that you can travel along the Markov chain's states forever, and eventually, as the time progresses, the probability of you ending up with one state becomes unchanges. This is the stationary probability that the $s_t = s$ when starting from s_0 and following policy π_θ for t steps. With the average reward formultion, the state-action value function is defined as

$$Q^\pi(s, a) = \sum_{t=1}^{\infty} \mathbb{E}[r_t - \rho(\pi)|s_0 = s, a_0 = a, \pi], \forall s \in \mathcal{S}, a \in \mathcal{A}.$$

G.7.1 Proof of Policy Gradient Theorem

$$\begin{aligned} J(\theta) &= \sum_{s\in\mathcal{S}} d^\pi(s) V^\pi(s) \\ &= \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a), \end{aligned}$$

where $d_\pi(s)$ is the stationary distribution of Markov chain for π_θ (on-policy state distribution under π). Imagine that you can travel along the Markov chain's states forever, and eventually, as the time progresses, the probability of you ending up with one state becomes unchanged, this is the stationary probability for π_θ.

$$\begin{aligned} \nabla_\theta V^\pi(s) &= \nabla_\theta \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a) \\ &= \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) + \pi_\theta(a|s) \nabla_\theta Q^\pi(s, a) \\ &= \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) + \pi_\theta(a|s) \nabla_\theta \sum_{s',r} P(s', r|s, a)(r + V^\pi(s')) \\ &= \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) + \pi_\theta(a|s) \sum_{s',r} P(s', r|s, a) \nabla_\theta V^\pi(s') \quad P(s', r|s, a) \text{ and } r \text{ is not a function of } \theta \\ &= \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) + \pi_\theta(a|s) \sum_{s'} P(s'|s, a) \nabla_\theta V^\pi(s') \end{aligned}$$

This equation has a recursive from. Let's consider a visitation sequence and transition probability from state s to state x with polict π_θ after k steps asr:

$$\rho^\pi(s \to x, k)$$

- This is a state transition probability with a policy π_θ
- When $k = 0$, $\rho^\pi(s \to s, k = 0) = 1$
- When $k = 1$, $\rho^\pi(s \to s', k = 1) = \sum_a \pi_\theta(a|s) P(s'|s, a)$
- $\rho^\pi(s \to x, k + 1) = \sum_{s'} \rho^\pi(s \to s', k) \rho^\pi(s' \to x, 1)$, where s' is the step right behind the state x (intermediate step).

$$
\begin{aligned}
\nabla_\theta V^\pi(s) &= \underbrace{\sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a)}_{\doteq\phi(s)} + \pi_\theta(a|s) \sum_{s'} P(s'|s,a) \nabla_\theta V^\pi(s') \\
&= \phi(s) + \sum_a \pi_\theta(a|s) \sum_{s'} P(s'|s,a) \nabla_\theta V^\pi(s') \\
&= \phi(s) + \sum_{s'} \sum_a \pi_\theta(a|s) P(s'|s,a) \nabla_\theta V^\pi(s') \\
&= \phi(s) + \sum_{s'} \rho^\pi(s \to s', k=1) \nabla_\theta V^\pi(s') \\
&= \phi(s) + \sum_{s'} \rho^\pi(s \to s', k=1) \left[\phi(s') + \sum_{s''} \rho^\pi(s' \to s'', k=1) \nabla_\theta V^\pi(s'') \right] \\
&= \phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \phi(s') + \sum_{s''} \sum_{s'} \rho^\pi(s \to s', 1) \rho^\pi(s' \to s'', k=1) \nabla_\theta V^\pi(s'') \\
&= \phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \phi(s') + \sum_{s''} \rho^\pi(s \to s'', 2) \nabla_\theta V^\pi(s'') \\
&\quad \vdots \\
&= \sum_{x\in\mathcal{S}} \sum_{k=0}^{\infty} \rho^\pi(s \to x, k) \phi(x)
\end{aligned}
$$

We can rewrite the above equation as

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta V^\pi(s) \\
&= \sum_{x\in\mathcal{S}} \underbrace{\sum_{k=0}^{\infty} \rho^\pi(s_0 \to s, k)}_{\doteq\eta(s)} \phi(x) \\
&= \sum_s \eta(s) \phi(s) \\
&= \underbrace{\sum_s \eta(s)}_{\text{Constant}} \sum_s \underbrace{\frac{\eta(s)}{\sum_s \eta(s)}}_{\text{Normalization}} \phi(s) \\
&\propto \sum_s \frac{\eta(s)}{\sum_s \eta(s)} \phi(s) \\
&= \sum_s d^\pi(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) \\
&= \mathbb{E}_\pi[\nabla_\theta \ln \pi_\theta(a|s) Q^\pi(s,a)]
\end{aligned}
$$