Reinforcement Learning

Notes (Unfinished Yet)

Jingye Wang

wangjy5@shanghaitech.edu.cn

Spring 2020

The framework of this note is enlightened by *SI-252* [1], while the contents mainly derived from not only *SI-252* but also *Intro to Reinforcement Learning, Bolei Zhou* [2], *Reinforcement Learning, David Silver* [3], *etc.* Many thanks to these great works!

Contents

1 Introduction **3**

2 Review of Basic Probability **4**

2.1 Interpretation of Probability 4

2.2 Transformations 5

2.3 Limit Theorem 5

2.4 Sampling & Monte Carlo Methods 5

2.5 Basic Inequalities 8

2.6 Concentration Inequalities 10

2.7 Conditional Expectation 11

3 Bandit Algorithms **13**

3.1 Bandit Models 13

3.2 Stochastic Bandits 13

3.3 Greedy Algorithms 14

3.4 UCB Algorithms 15

3.5 Bayesian Bandits and Thompson Sampling Algorithms 16

3.6 Gradient Bandit Algorithms 17

4 Markov Chains **17**

4.1 Markov Model . . . 17
4.2 Basic Computations . . . 18
4.3 Classification of States . . . 18
4.4 Stationary Distribution . . . 19
4.5 Reversibility . . . 19
4.6 Markov chain Monte Carlo . . . 20

5 Markov Decision Process **22**

5.1 Markov Process . . . 22
5.2 Markov Reward Process . . . 22
5.3 Markov Decision Process . . . 24
5.4 Dynamic Programming . . . 25

6 Model-free Prediction **27**

6.1 Monte-Carlo Policy Evaluation . . . 27
6.2 Temporal-Difference Learning . . . 28

7 Model-free Control **30**

7.1 On Policy Monte-Carlo Control . . . 30
7.2 On Policy Temporal-Difference Control . . . 32
7.3 Off-Policy Q-Learning Control . . . 33
7.4 Off-Policy Importance Sampling Control . . . 34

8 Value Function Approximation **34**

8.1 Introduction on Function Approximation . . . 34
8.2 Incremental Method . . . 35
8.3 Batch Methods . . . 37
8.4 Deep Q-Learning . . . 38

9 Policy Optimization **39**

9.1 Policy Optimization . . . 39
9.2 Monte-Carlo Policy Gradient . . . 40
9.3 Actor-Critic Policy Gradient . . . 43
9.4 Extension of Policy Gradient . . . 44

1 Introduction

Course Prerequisite:

- Linear Algebra
- Probability
- Machine Learning relevant course (data mining, pattern recognition, *etc*)
- PyTorch, Python

What is Reinforcement Learning and why we care:

a computational approach to learning whereby *an agent* tries to *maximize* the total amount of *reward* it receives while interacting with a complex and uncertain *environment*.[4]

Difference between Reinforcement Learning and Supervised Learning:

- Sequential data as input (*not i.i.d*);
- The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them;
- *Trial-and-error* exploration (balance between exploration and exploitation);
- There is *no supervisor*, only a reward signal, which is also *delayed*

Big deal: Able to Achieve Superhuman Performance

- Upper bound for Supervised Learning is human-performance.
- Upper bound for Reinforcement Learning ?

Why Reinforcement Learning works now?

- Computation power: many GPUs to do trial-and-error rollout;
- Acquire the high degree of proficiency in domains governed by simple, known rules;
- End-to-end training, features and policy are jointly optimized toward the end goal.

Sequential Decision Making:

- Agent and Environment: the agent learns to interact with the environment;
- Rewards: a scalar feedback signal that indicates how well agent is doing;
- Policy: a map function from state/observation to action models the agent’s behavior;
- Value function: expected discounted sum of future rewards under a particular policy;
- Objective of the agent: selects a series of actions to maximize total future rewards;
- History: a sequence of observations, actions, rewards;
- Full observability: agent directly observes the environment state, formally as Markov decision process (MDP);

- Partial observability: agent indirectly observes the environment, formally as partially observable Markov decision process (POMDP)

All goals of the agent can be described by the maximization of expected cumulative reward.

Types of Reinforcement Learning Agents based on What the Agent Learns

- Value-based agent:
  - Explicit: Value function;
  - Implicit: Policy (can derive a policy from value function);
- Policy-based agent:
  - Explicit: policy;
  - No value function;
- Actor-Critic agent:
  - Explicit: policy and value function.

Types of Reinforcement Learning Agents on if there is model

- Model-based:
  - Explicit: model;
  - May or may not have policy and/or value function;
- Model-free:
  - Explicit: value function and/or policy function;
  - No model.

2 Review of Basic Probability

For more details of this part one can refer to *Introduction to Probability* [5] and *Monte Carlo Statistical Methods* [6].

2.1 Interpretation of Probability

The Frequentist view: *Probability represents a long-run frequency over a large number of repetitions of an experiment.*

The Bayesian view: *Probability represents a degree of belief about the event in question.*

Many machine learning techniques are derived from these two views. As the computing power and algorithms develop, however, Bayesian is becoming dominant.

2.2 Transformations

Change of variables: *Let $\boldsymbol{X} = (X_1, ..., X_n)$ be a continuous random vector with joint PDF $\boldsymbol{X}$, and let $\boldsymbol{Y} = g(\boldsymbol{X})$ where g is an invertible function from $\mathbb{R}^n$ to $\mathbb{R}^n$. Then the joint PDF of $\boldsymbol{Y}$ is*

$$f_{\boldsymbol{Y}}(\boldsymbol{y}) = f_{\boldsymbol{X}}(\boldsymbol{x}) \left| \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}} \right|$$

where the vertical bars say "take the absolute value of the determinant of $\partial \boldsymbol{x}/\partial \boldsymbol{y}$" and $\partial \boldsymbol{x}/\partial \boldsymbol{y}$ is a **Jacobian matrix**

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}} = \begin{pmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} & \cdots & \frac{\partial x_1}{\partial y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_n}{\partial y_1} & \frac{\partial x_n}{\partial y_2} & \cdots & \frac{\partial x_n}{\partial y_n} \end{pmatrix}.$$

It assumes that the determinant of the Jacobian matrix is never 0. It also supposes all the partial derivatives $\frac{\partial x_i}{\partial y_j}$ exist and are continuous.

2.3 Limit Theorem

Strong Law of Large Numbers (SLLN): *The sample mean $\bar{X}_n$ converges to the true mean μ point-wise as $n \to \infty$, w.p.1(i.e. with probability 1). In other words, the event $\bar{X}_n \to \mu$ has probability 1.*

Weak Law of Large Numbers (WLLN): *For all $\epsilon > 0, P(|\bar{X}_n - \mu| > \epsilon) \to 0$ as $n \to \infty$. (This form of convergence is called* **convergence in probability** *).*

The Weak Law of Large Numbers can be proved by using **Chebyshev's inequality**.

Central Limit Theorem (CLT): *As $n \to \infty$, $\sqrt{n}\left(\frac{\bar{X}_n - \mu}{\sigma}\right) \to \mathcal{N}(0,1)$ in distribution. In words, the CDF of the left-hand side approaches the CDF of the standard Normal distribution.*

2.4 Sampling & Monte Carlo Methods

Inverse Transform Method: *Let F be a CDF which is a continuous function and strictly increasing on the support of the distribution. This ensures that the inverse function F^{-1} exists, as a function from $(0,1)$ to* R. *We then have the following results.*

1. *Let $U \sim$ Unif$(0,1)$ and $X = F^{-1}(U)$. Then X is an r.v. with CDF F.*
2. *Let X be an r.v. with CDF F. Then $F(X) \sim$ Unif$(0,1)$.*

Proof:

1. Let $U \sim$ Unif$(0,1)$ and $X = F^{-1}(U)$. Then we have $P(U \leq u) = u$ for $u \in (0,1)$. For all real x,

$$P(X \leq x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)) = F(x),$$

so the CDF of X is F, as claimed.

2. Let X have CDF F, and find the CDF of $Y = F(X)$. Since Y takes values in $(0,1)$, $P(Y \leq y)$ equals 0 for $y \leq 0$ and equals 1 for $y \geq 1$. For $y \in (0,1)$,

$$P(Y \leq y) = P(F(X) \leq y) = P(X \leq F^{-1}(y)) = F(F^{-1}(y)) = y.$$

Thus Y has the Unif$(0,1)$ CDF. □

Box-Muller: *Let* $U \sim \text{Unif}(0, 2\pi)$*, and let* $T \sim \text{Expo}(1)$ *be independent of U. Define* $X = \sqrt{2T}\cos U$ *and* $Y = \sqrt{2T}\sin U$*. Then* X *and* Y *are independent and the joint PDF of* $(X, Y)is$

$$f_{X,Y}(x,y) = \frac{1}{2\pi} e^{\frac{1}{2}(x^2+y^2)}$$

Proof:

The joint PDF of U and T is

$$f_{U,T}(u,t) = \frac{1}{2\pi} e^{-t},$$

for $u \in (0, 2\pi)$ and $t > 0$. And we have the Jacobian matrix

$$\frac{\partial(x,y)}{\partial(u,t)} = \begin{pmatrix} -\sqrt{2t}\sin u & \frac{1}{\sqrt{2t}}\cos u \\ \sqrt{2t}\cos u & \frac{1}{\sqrt{2t}}\sin u \end{pmatrix}.$$

Then we have

$$\begin{aligned} f_{X,Y}(x,y) &= f_{U,T}(u,t) \cdot \left| \frac{\partial(u,t)}{\partial(x,y)} \right| \\ &= \frac{1}{2\pi} e^{-t} \cdot 1 \\ &= \frac{1}{2\pi} e^{-\frac{1}{2}(x^2+y^2)} \\ &= \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \cdot \frac{1}{\sqrt{2\pi}} e^{-y^2/2} \end{aligned}$$

for all real x and y. The joint PDF $f_{X,Y}$ factors into a function x times a function of y, so X and Y are independent. Furthermore, we can find that X and Y are i.i.d. $\mathcal{N}(0,1)$. That shows how the *Box-Muller* method works for generating Normal r.v.s. □

Monte Carlo Integration: *Given a function* $\Phi : \mathbb{R}^n \to \mathbb{R}$*. If* $p(\cdot)$ *denotes a valid PDF with the support over* $\mathbb{R}^n$*, then we have*

$$\begin{aligned} \int_{\mathbb{R}^n} \Phi(x)dx &= \int_{\mathbb{R}^n} \frac{\Phi(x)}{p(x)} \cdot p(x)dx \\ &= \mathbb{E}_p \left[\frac{\Phi(x)}{p(x)} \right] \\ &\approx \frac{1}{N} \sum_{k=1}^{N} \frac{\Phi(x_k)}{p(x_k)}, \end{aligned}$$

where $x_k \sim p$*.*

By the law of large numbers, the estimator converges to the true value of the integral with probability 1 as $n \to \infty$. Therefore we can use random samples to obtain approximations of definite integrals when

exact integration methods are unavailable. Such approach is often referred to as the **Monte Carlo method**.

Importance Sampling: *Let $p(x)$ denote the target distribution and $\mathbb{E}_p[c(x)]$ is what we want to estimate. With a PDF $q(x)$ which subject to that $\frac{p(x)}{q(x)}$ is finite for all $x \in A$, we have*

$$\begin{aligned}\mathbb{E}_p[c(x)] &= \int_A c(x)p(x)dx \\ &= \int_A c(x) \cdot \frac{p(x)}{q(x)} \cdot q(x)dx \\ &= \mathbb{E}_q\left[c(x) \cdot \frac{p(x)}{q(x)}\right] \\ &\approx \frac{1}{N}\sum_{k=1}^{N} c(x_k)\frac{p(x_k)}{q(x_k)}\end{aligned}$$

where $x_k \sim q$ and q is called **importance distribution**.

The estimator converges to the true value for the same reason the Monte Carlo method converges. Furthermore, the estimator with importance sampling has the less variance than that of the standard Monte Carlo method.

Acceptance-Rejection Method: *Let $X \sim p$ and $Y \sim q$ from which we can relatively easily generate samples. Then for a constant c such that $c \geq \sup_\zeta \frac{p(\zeta)}{q(\zeta)}$, we can simulate $X \sim p$ with three steps:*

Step 1: Generate $y \sim q$.

*Step 2: Generate $u \sim$*U$(0, 1)$.

Step 3: If $u \leq \frac{p(y)}{cq(y)}$, set $x = y$. Otherwise go back to Step 1.

Proof:

This method can be easily proved. From the description, we have

$$\begin{aligned}P(X \leq \zeta) &= P\left(Y \leq \zeta | U \leq \frac{p(y)}{cq(y)}\right) \\ &= \frac{P(Y \leq \zeta, U \leq \frac{p(y)}{cq(y)})}{P(U \leq \frac{p(y)}{cq(y)})} \\ &= \frac{\int_0^\zeta \int_0^{\frac{p(y)}{cq(y)}} 1du \cdot q(y)dy}{\int_{-\infty}^{+\infty} \int_0^{\frac{p(y)}{cq(y)}} 1du \cdot q(y)dy} \\ &= \frac{\int_{-\infty}^{\zeta} p(y)dy}{\int_{-\infty}^{+\infty} p(y)dy}\end{aligned}$$

If p is a valid PDF, the denominator will equal 1. Thus we have

$$P(X \leq \zeta) = \int_{-\infty}^{\zeta} p(x)dx,$$

which shows that $X \sim p$. □

2.5 Basic Inequalities

Cauchy-Schwarz Inequality: *For any r.v.s X and Y with finite variances,*

$$\mathbb{E}[XY]| \leq \sqrt{\mathbb{E}[X^2]\mathbb{E}[Y^2]}.$$

Proof:

For any t,

$$\mathbb{E}[(Y - tX)^2] \geq 0$$
$$\mathbb{E}[(Y^2 - 2tXY + t^2X^2)] \geq 0,$$

where the left-hand side is a quadratic function with respect to t. To satisfy the inequality, the discriminant of the quadratic must be less than 0, which means

$$[2\mathbb{E}[XY]]^2 - 4 \cdot \mathbb{E}[X^2]\mathbb{E}[Y^2] \leq 0$$
$$[\mathbb{E}[XY]]^2 \leq \mathbb{E}[X^2]\mathbb{E}[Y^2]$$
$$|\mathbb{E}[XY]| \leq \sqrt{\mathbb{E}[X^2]\mathbb{E}[Y^2]}.$$

Therefore we have the Cauchy-Schwarz inequality. □

Jensen's Inequality: *Let X be a random variable. If g is a convex function, then $\mathbb{E}[g(X)] \geq g(\mathbb{E}[X])$. If g is a concave function, then $\mathbb{E}[g(X)] \leq g(\mathbb{E}[X])$. In both cases, the only way that equality can hold is if there are constants a and b such that $g(X) = a + bX$ with probability 1.*

Proof:

If g is convex, then all lines that are tangent to g lie below g. Denoting the tangent line of g by $a + bx$, we have $g(x) \geq a + bx$ for all x by convexity, so $g(X) \geq a + bX$. Taking the expectation of both sides,

$$\mathbb{E}[g(X)] \geq \mathbb{E}[a + bX]$$
$$\geq a + b\mathbb{E}[X]$$
$$\geq g(\mathbb{E}[X]).$$

If g is concave, then $h = -g$ is convex, so we can apply the proof to h to see that the inequality for g is reversed from the convex case.

Lastly, assume that $g(X) = a + bX$ holds in the convex case. Let $Y = g(X) - a - bX$. Then Y must be a nonnegative r.v. with $\mathbb{E}[Y] = 0$, so $P(Y = 0) = 1$. So equality holds if and only if $P(g(X) = a + bX) = 1$. For the concave case, we can use the similar argument with $Y = a + bX - g(X)$. □

Norm Inequality: *For a random variable X whose moment of order $r > 0$ is finite, we define the following norm*

$$\|X\|_r = (\mathbb{E}[|X|^r)]^{\frac{1}{r}}.$$

With this definition, we have the following inequalities.

- *Holder Inequality*: *Let* $\frac{1}{p} + \frac{1}{q} = 1$. *If* $\mathbb{E}[|X|^p], \mathbb{E}[|X|^q] < \infty$, *then* $|\mathbb{E}[XY]| \leq \mathbb{E}[|XY|] \leq \|X\|_p \cdot \|Y\|_q$.
- *Lyapunov Inequality*: *For* $0 < r \leq p$, $\|X\|_r \leq \|X\|_p$.
- *Minkowski Inequality*: *Let* $p \geq 1$. *If* $\mathbb{E}[|X|^p], \mathbb{E}[|Y|^p] < \infty$, *then* $\|X + Y\|_p \leq \|X\|_p + \|Y\|_p$.

Markov's Inequality: *For any r.v.* X *and constant* $a > 0$,

$$P(|X| \geq a) \leq \frac{\mathbb{E}[|X|]}{a}.$$

Proof:

Let $Y = \frac{|X|}{a}$. We need to show that $P(Y \geq 1) \leq \mathbb{E}[Y]$. Note that

$$I(Y \geq 1) \leq Y,$$

taking the expectation of both sides, then we have Markov's Inequality. □

Markov's inequality is a very crude bound because it requires absolutely no assumptions about X. The right-hand side of the inequality could be greater than 1 sometimes, or even infinite.

Chebyshev's Inequality: *Let* X *have mean* μ *and variance* σ^2. *Then for any* $a > 0$,

$$P(|X - \mu| \geq a) \leq \frac{\sigma^2}{a^2}.$$

Proof:

By Markov's inequality,

$$\begin{aligned} P(|X - \mu| \geq a) &= P((X - \mu)^2 \geq a^2) \\ &\leq \frac{\mathbb{E}[(X - \mu)^2]}{a^2} \\ &= \frac{\sigma^2}{a^2}. \end{aligned}$$

□

Chernoff's Inequality: *For any r.v.* X *and constants* $a > 0$ *and* $t > 0$,

$$P(X \geq a) \leq \frac{\mathbb{E}[e^{tX}]}{e^{ta}}.$$

Proof:

The transformation g with $g(x) = e^{tx}$ is invertible and strictly increasing. So by Markov's inequality, we have

$$\begin{aligned} P(X \geq a) &= P(e^{tX} \geq e^{ta}) \\ &\leq \frac{\mathbb{E}[e^{tX}]}{e^{ta}}. \end{aligned}$$

□

2.6 Concentration Inequalities

Hoeffding Lemma: *Let X be a r.v. with $\mathbb{E}[X] = 0$, taking values in a bounded interval $[a, b]$, where a and b are constants. Then for any $\lambda > 0$,*

$$\mathbb{E}[e^{\lambda X}] \leq e^{\frac{1}{8}\lambda^2(b-a)^2}.$$

Proof:

For the case $a = b = 0$, we have $P(X = 0) = 1$. The equality holds since both sides of the inequality are 1. Then we consider the case that $a < 0$ and $b > 0$, which is the general case.

Let $f(x) = e^{\lambda x}$ where $x \in [a, b]$. According to its convexity, for any $\alpha \in (0, 1)$, we have

$$f(\alpha a + (1-\alpha)b) \leq \alpha f(a) + (1-\alpha)f(b) = \alpha e^{\lambda a} + (1-\alpha)e^{\lambda b}.$$

As $X \in [a, b]$, let $\alpha = \frac{b-X}{b-a}$, then we have $f(\alpha a + (1-\alpha)b) = f(X) = e^{\lambda X}$. Plugging the two equations into the previous inequality, we have

$$e^{\lambda X} \leq \frac{b-X}{b-a}e^{\lambda a} + \frac{X-a}{b-a}e^{\lambda b}.$$

Taking the expectation of both sides,

$$\mathbb{E}[e^{\lambda X}] \leq \frac{b}{b-a}e^{\lambda a} - \frac{a}{b-a}e^{\lambda b} = e^{\lambda a}\left[\frac{b}{b-a} - \frac{a}{b-a}e^{\lambda(b-a)}\right],$$

and defining a function $\Phi(t) = -\theta t + \ln(1 - \theta + \theta e^t)$ where $\theta = \frac{-a}{b-a} > 0$, we have

$$\mathbb{E}[e^{\lambda X}] \leq e^{\Phi(\lambda(b-a))},$$

as $e^{\Phi(\lambda(b-a))} = e^{\lambda a}[\frac{b}{b-a} - \frac{a}{b-a}e^{\lambda(b-a)}]$.

We now focus on $\Phi(t)$. According to Taylor expansion, for any $t > 0$, $\exists \tau \in [0, t)$ s.t.

$$\begin{aligned}\Phi(t) &= \Phi(0) + t\Phi'(0) + \frac{1}{2}t^2\Phi''(\tau) \\ &= \frac{1}{2}t^2 \cdot \frac{(1-\theta)\theta e^\tau}{(1-\theta+\theta e^\tau)^2} \\ &\leq \frac{1}{8}t^2\end{aligned}$$

since

$$(1-\theta+\theta e^\tau)^2 = (1-\theta-\theta e^\tau)^2 + 4(1-\theta)\theta e^\tau \geq 4(1-\theta)\theta e^\tau.$$

Plugging in $t = \lambda(b-a)$, we have $\Phi(\lambda(b-a)) \leq \frac{1}{8}\lambda^2(b-a)^2$. It follows that

$$\mathbb{E}[e^{\lambda X}] \leq e^{\frac{1}{8}\lambda^2(b-a)^2}.$$

□

Hoeffding Bound: *Let $X_1, X_2, ..., X_n$ be independent r.v.s with $\mathbb{E}[X_i] = \mu, a \leq X_i \leq b$ for each $i = 1, 2, ..., n$, where a, b are constants. Then for any $\epsilon \geq 0$,*

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right| \geq \epsilon\right) \leq 2e^{-\frac{2n\epsilon^2}{(b-a)^2}}.$$

Proof:

Let $Z_i = X_i - \mu$ and $Z = \frac{1}{n}\sum_{i=1}^{n} Z_i$, then we have $\mathbb{E}[Z_i] = 0$ and $\mathbb{E}[Z] = 0$. For any $\lambda > 0$, we have

$$P(Z \geq \epsilon) \leq \frac{\mathbb{E}[e^{\lambda Z}]}{e^{\lambda \epsilon}}$$

by Chernoff's inequality. For $\mathbb{E}[e^{\lambda Z}]$, we have

$$\mathbb{E}[e^{\lambda Z}] = \mathbb{E}[e^{\lambda \frac{1}{n}\sum_{i=1}^{n} Z_i}] = \prod_{i=1}^{n} \mathbb{E}[e^{\frac{\lambda}{n} Z_i}].$$

As $Z_i = X_i - \mu \in [a - \mu, b - \mu]$, using Hoeffding Lemma, we have

$$\prod_{i=1}^{n} \mathbb{E}[e^{\frac{\lambda}{n} Z_i}] \leq e^{\frac{\lambda^2}{8n}(b-a)^2}.$$

Therefore we have

$$P(Z \geq \epsilon) \leq e^{-\lambda\epsilon + \frac{\lambda^2}{8n}(b-a)^2}.$$

Now we focus on the quadratic $-\lambda\epsilon + \frac{\lambda^2}{8n}(b-a)^2$ w.r.t λ. It is easy to find the quadratic has the minimum at $\lambda = \frac{4n\epsilon}{(b-a)^2}$. Plugging in $\lambda = \frac{4n\epsilon}{(b-a)^2}$, we have

$$P(Z \geq \epsilon) \leq e^{\frac{-2n\epsilon^2}{(b-a)^2}}.$$

Therefore by the symmetry we have

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right| \geq \epsilon\right) \leq 2e^{\frac{-2n\epsilon^2}{(b-a)^2}}.$$

□

2.7 Conditional Expectation

"Conditional probabilities are probabilities, and all probabilities are conditional."

For conditional expectation, the case is similar.

Taking out what's known: *For any function h,*

$$\mathbb{E}[h(X)Y|X] = h(X)\mathbb{E}[Y|X].$$

Intuitively, when we take expectations given X, we are treating X as if it has crystallized into a known constant. Then any function of X, say h(X), also acts like a known constant while we are conditioning on X.

Law of Total Expectation (LOTE): *Let $A_1, ..., A_n$ be a partition of a sample space, with $P(A_i) > 0$ for all i, and let Y be a random variable on this sample space. Then*

$$\mathbb{E}[Y] = \sum_{i=1}^{n} \mathbb{E}[Y|A_i]P(A_i).$$

Adam's Law: *For any r.v.s X and Y,*

$$\mathbb{E}[\mathbb{E}[Y|X]] = \mathbb{E}[Y].$$

Proof:

Without loss of generality, we consider the case where X and Y are both discrete. Let $E(Y|X) = g(X)$. Expanding the definition of $g(x)$ by applying LOTUS, we have

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[Y|X]] &= \mathbb{E}[g(X)] \\
&= \sum_x g(x)P(X=x) \\
&= \sum_x \mathbb{E}[Y|X=x]P(X=x) \qquad (1) \\
&= \sum_x \left(\sum_y yP(Y=y|X=x) \right) P(X=x) \\
&= \sum_y y \sum_x P(Y=y, X=x) \\
&= \sum_y yP(Y=y) \\
&= \mathbb{E}[Y] \qquad (2)
\end{aligned}$$

as desired. Also, as it shown in the proof, with (1) and (2) we can prove the LOTE. □

Adam's law with extra conditioning: *For any r.v.s X, Y, Z we have*

$$\mathbb{E}[\mathbb{E}[Y|X,Z]|Z] = \mathbb{E}[Y|Z].$$

Proof:

Define the expectation $\hat{\mathbb{E}}[\cdot] = \mathbb{E}[\cdot|Z]$. The key is that *"conditional expectation is expectation"*. We have

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[Y|X,Z]|Z] &= \hat{\mathbb{E}}[\hat{\mathbb{E}}[Y|X] \\
&= \hat{\mathbb{E}}[Y] && \text{(by Adam's Law)} \\
&= \mathbb{E}[Y|Z]
\end{aligned}$$

□

Eve's law: *For any r.v.s X and Y,*

$$Var(Y) = \mathbb{E}[Var(Y|X)] + Var(\mathbb{E}[Y|X]).$$

It is also known as the law of the variance or the variance decomposition formula.

Proof:

Let $g(X) = \mathbb{E}[Y|X]$. By Adam's law, we have $\mathbb{E}[g(X)] = \mathbb{E}[\mathbb{E}[Y|X]] = \mathbb{E}[Y]$. According to the variance of the expectation that

$$Var(Y|X) = \mathbb{E}[Y^2|X] - (\mathbb{E}[Y|X])^2,$$

which can be shown by the way we prove Adam's law, we have

$$\begin{aligned}
\mathbb{E}[Var(Y|X)] &= \mathbb{E}[\mathbb{E}[Y^2|X] - (g(X))^2] \\
&= \mathbb{E}[\mathbb{E}[Y^2|X]] - \mathbb{E}[(g(X))^2] \\
&= \mathbb{E}[Y^2] - \mathbb{E}[(g(X))^2], \qquad (1) \\
Var(\mathbb{E}[Y|X]) &= \mathbb{E}[(g(X))^2] - (\mathbb{E}[\mathbb{E}[Y|X]])^2 \\
&= \mathbb{E}[g(X)^2] - (\mathbb{E}[Y])^2. \qquad (2)
\end{aligned}$$

The Eve's law can be shown by $(1) + (2)$.

3 Bandit Algorithms

The large part of this section was done with references [1, 7, 4, 8, 9].

3.1 Bandit Models

As a special case of Reinforcement Learning, bandit algorithms share some important concepts of that.

Reward: *A reward R_t is a scalar feedback signal which indicates how well agent is doing at step t.*

Reinforcement learning is based on the reward hypothesis that all goals can be described by the maximization of *expected cumulative reward*. Depends on how well we know the reward, there are three types of *feedback*.

Bandit Feedback: *the agent only knows the reward for the chosen arm;*

Full Feedback: *the agent knows the rewards for all arms that could have been chosen;*

Partial Feedback: *apart from the chosen arm, the agent knows rewards for some arms.*

Besides the feedback, the rewards model also varies, such as *IID Rewards, Adversarial rewards, Constrained Adversarial, and Stochastic Rewards*.

3.2 Stochastic Bandits

The basic settings of stochastic bandits are *Bandit feedback* and *IID reward*. Also, we assume that per-round rewards are bounded. A multi-armed bandit $< \mathcal{A}, \mathcal{R} >$ are defined as follows:

$\mathcal{A}$: a known set of m actions;

$\mathcal{R}^a(r) = \mathbb{P}(r|a)$: an *unknown* probability distribution over rewards r;

a_t: the action selected by the agent at time t and $a_t \in \mathcal{A}$;

r_t: the reward generated by the environment at time t and $r_t \sim \mathcal{R}^{a_t}$;

$E\left(\sum_{\tau=1}^{t} r_\tau\right)$: the goal we want to maximize.

For convenience, we define

$Q(a) = E(r_t|a_t = a)$: the mean reward for the action a;

$V^* = \max_{a\in\mathcal{A}} Q(a)$: the optimal reward for the action a;

$L_\tau = E(V^* - Q(a_\tau))$: the regret for the round τ, indicating the opportunity loss;

According to those definitions, the goal of the multi-armed bandit $< \mathcal{A}, \mathcal{R} >$ is equivalent to minimize

$$L_t = E\left(\sum_{\tau=1}^{t}(V^* - Q(a_\tau))\right).$$

Another way to represent regret is *counting regret*:

$N_t(a) = \sum_{\tau=1}^{t} I_{a_\tau=a}$: the number of selections for the action a after the end of the round t;

$\Delta_a = V^* - Q(a)$: the difference in the value between the action a and the optimal action a^*;

Then the regret follows that

$$L_t = \sum_{a\in\mathcal{A}} E(N_t(a))\Delta_a,$$

which is called *Regret Decomposition Lemma*. Almost all bandit proofs are based on this lemma.

Linear regret: *The regret L_t over t rounds is s.t.*

$$\lim_{t\to\infty} \frac{L_t}{t} = 1.$$

Sublinear regret: *The regret L_t over t rounds is s.t.*

$$\lim_{t\to\infty} \frac{L_t}{t} = 0.$$

Logarithmic asymptotic regret: *The performance of any bandit algorithm is determined by similarity between optimal arm and other arms. Asymptotic total regret is at least logarithmic in number of steps*:

$$\lim_{t\to\infty} L_t \geq \log t \sum_{a|\Delta_a>0} \frac{\Delta_a}{KL(\mathcal{R}^{\dashv}||\mathcal{R}^*)}.$$

The lower bound is also known as *Lai-Robbins* lower bound.

The Exploration/Exploitation Dilemma: *As the action-values are unknown, we must both try actions to learn the action-values (explore), and prefer those that appear best (exploit).*

3.3 Greedy Algorithms

Greedy Action: *Define the greedy action at time t as*

$$a_t^\epsilon = \arg\max_{a\in\mathcal{A}} Q(a).$$

If $a_t = a_t^\epsilon$, we are making exploitation, otherwise, we are making exploration.

Usually $Q(a)$ is not available, we can estimate it by Monte-Carlo evaluation:

$$\hat{Q}(a) = \frac{1}{N_{t-1}(a)} \sum_{\tau=1}^{t-1} r_\tau I(a_\tau = a).$$

ϵ-greedy algorithm: *Rather than making exploitation every round, we make exploitation with probability $1-\epsilon$, or make exploration with probability ϵ.*

Algorithm 1: ϵ-greedy algorithm

Initialization: $\hat{Q}(a) \leftarrow 0,\ N(a) \leftarrow 0,\ \text{for } a = 1, ..., k$;

Repeat:

$$A \leftarrow \begin{cases} \arg\max_a \hat{Q}(a) & \text{with probability } 1-\epsilon; \\ \text{a random action} & \text{with probability } \epsilon; \end{cases}$$

$R \leftarrow bandit(A)$;

$N(A) \leftarrow N(A) + 1$;

$\hat{Q}(A) \leftarrow \hat{Q}(A) + \frac{1}{N(A)}(R - \hat{Q}(A))$;

For ϵ-greedy algorithm, the lower bound of the total regret is

$$L_t \geq \frac{t\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \Delta_a.$$

Therefore it has linear total regret. A simple and practical idea is initialize $\hat{Q}(a)$ to high values. Such trick is called *optimistic initialization*.

Decaying ϵ_t-greedy algorithm: *The greedy degree ϵ in this algorithm is not a constant. Rather, we have a decay schedule for ϵ_t as*

$$\epsilon_t = \min\left\{1, \frac{c|\mathcal{A}|}{d^2 t}\right\},$$

where $c > 0$ and $d = \min_{a|\Delta a>0} \Delta_i$.

For decaying ϵ-greedy algorithm, the lower bound of the total regret is *logarithmic asymptotic*, which means, however, the schedule for ϵ_t requires advance knowledge of gaps Δ_a.

3.4 UCB Algorithms

UCB algorithm: *Estimate an upper confidence bound $\hat{U}_t(a)$ for each $Q(a)$, i.e.*

$$Q(a) \leq \hat{Q}_t(a) + \hat{U}_t(a).$$

Then select actions with max upper confidence bound, i.e.

$$a_t = \arg\max_{a \in \mathcal{A}} \{\hat{Q}_t(a) + \hat{U}_t(a)\}.$$

In other words, UCB algorithm favors exploration of actions with a strong potential to have a optimal value. When we don't have any prior knowledge, the bound can be determined by *Hoeffding's Inequality*. It follows that

$$P(Q(a) > \hat{Q}_t(a) + \hat{U}_t(a)) \leq e^{-2N_t(a)U_t^2(a)}.$$

Since we want to pick a bound so that with high chances the true mean $Q(a)$ is blow the sample mean $\hat{Q}_t(a)$ + the upper confidence bound $U_t^(a)$, the right-hand side of the inequality should be a small probability, for example, a tiny threshold p, therefore we have

$$U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}.$$

The upper bound $\hat{U}_t(a)$ is a function of $N_t(a)$ which means a larger number of trials $N_t(a)$ should give us a smaller bound $\hat{U}_t(a)$. Further, we want the small bound to have a high probability, which we can achieve by designing the p.

UCB1: *As we want to make more confident bound estimation with more rewards observed, the threshold p should be reduced in time. A reasonable idea is setting $p = t^{-4}$ and then we get UCB1 algorithm*

$$U_t(a) = \sqrt{\frac{2\log t}{N_t(a)}},$$

$$a_t^{UCB1} = \arg\max_{a\in\mathcal{A}} \left\{ Q(a) + \sqrt{\frac{2\log t}{N_t(a)}} \right\},$$

with the lower bound of the total regret s.t.

$$\lim_{t\to\infty} L_t \leq 8\log t \sum_{a|\Delta_a>0} \Delta_a.$$

The selection rule can give us a intuitive explanation. When $\hat{Q}_t(a)$ is large, it indicates a high reward with high probability and leads to exploitation; when $N_t(a)$ is small, it indicates the action may have a great potential and leads to exploration.

3.5 Bayesian Bandits and Thompson Sampling Algorithms

So far we have made no assumptions about the reward distribution $\mathcal{R}$. While if the distribution is known, then we have the *Bayesian Bandits*, and the goal is to optimize *Bayesian regret*. The algorithm for this setting make use of the posterior to guide exploration. To be specific, at each time step, we want to select action a according to the probability that a is optimal action, i.e., a *probability matching*:

$$\pi(a|h_t) = P(Q(a) > Q(a'),\ \forall a' \neq a|h_t),$$

where $h_t = a_1, r_1, ..., a_{t-1}, r_{t-1}$ is the history.

Thompson sampling: *Thompson sampling use Bayes' law to compute posterior distribution $P(\mathcal{R}|h_t)$ and compute the action-value function $\hat{Q}(a) = f(\mathcal{R}_a)$, then it selects the action*

$$a_t^{TS} = \arg\max_{a\in\mathcal{A}} \hat{Q}(a).$$

A naive example for Thompson sampling is Beta-Bernoulli Bandit. It assumes that the action K being played produces a reward satisfying *Bern*(θ_k), and θ_k, which is also equals $Q(a)$, follows a *Beta*(α_k, β_k). Let x_t denote the action selected at time t and r_t denote the corresponding reward of action x_t. Then we can update the parameters of the *Beta* distribution

$$(\alpha_k, \beta_k) \leftarrow \begin{cases} (\alpha_k, \beta_k) & \text{if } x_t \neq k, \\ (\alpha_k, \beta_k) + (r_t, 1 - r_t) & \text{if } x_t = k. \end{cases}$$

However, for many practical and complex problems, it can be computationally intractable to estimate the posterior distributions with observed true rewards using Bayesian inference. In that case, we may need to approximate the posterior distributions using methods like Gibbs sampling, Laplace approximate, and the bootstraps.

3.6 Gradient Bandit Algorithms

Previously our bandit algorithms are based on the estimation of action values. while the gradient bandit algorithm is based on a new criterion.

Gradient Bandit Algorithm: *Let $H_t(a)$ be a learned preference for taking action a, then we have the rules*

$$P(A_t = a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} = \pi_t(a),$$

and

$$H_{t+1}(a) = H_t(a) + \alpha(\mathcal{R}_t - \bar{\mathcal{R}}_t)(I_{A_t=a} - \pi_t(a)),$$

where $\bar{\mathcal{R}}_t = \frac{1}{t}\sum_{i=1}^{t} \mathcal{R}_i$ and $\alpha > 0$.

The algorithm is also known as *stochastic gradient ascent algorithm* since the term $(\mathcal{R}_t - \bar{\mathcal{R}}_t)(I_{A_t=a} - \pi_t(a))$ is essentially the stochastic gradient of $E(\mathcal{R}_t)$ with respect to $H_t(a)$.

Actually, gradient bandit algorithm has a bad performance when compared with other algorithms. However, the introduction of the gradient, further, the function, inspires many important algorithms in Reinforcement Learning.

4 Markov Chains

The large part of this section was done with references [1, 5, 6].

4.1 Markov Model

Stochastic Processes: *A stochastic process is a collection of random variables $\{X_t, t \in I\}$. The set I is the index set of the process. All the r.v. are defined on a common state space $\mathcal{S}$.*

Markov Process: *A Markov process has three basic components:*

Markov property;

A sequence of random variables $\{X_t, t \in \mathcal{T}\}$, where $\mathcal{T}$ is an index set;

All possible sample values of $\{X_t, t \in \mathcal{T}\}$ are called states, which are elements of a sate space $\mathcal{S}$.

Markov Property: *given the present value(information) of the process, the future evolution of the process is independent of the past evolution of the process.*

Markov Chain: *A sequence of random variables $X_0, X_1, X_2, ...$ taking values in the state space $\mathcal{S}$ is called a Markov chain if for all $n \geq 0$*

$$P(x_{n+1} = j | X_n = i_n, X_{n-1} = i_{n-1}, ..., X_0 = i_0) = P(X_{n+1} = j | X_n = i).$$

Transition Matrix: *For a Markov Chain, let $P_{ij} = P(X_{n+1} = j | X_n = i)$ be the transition probability from state i to state j. Then the matrix P is called the transition matrix of the chain.*

Transition Matrix is a way to express Markov Chain. Besides that, Markov Chain also has Graphical form. Therefore the model can be applied to many problems.

4.2 Basic Computations

Now we introduce some useful computations.

n-step Transition Probability: *The n-step transition probability from i to j is the probability of being at j exactly n steps after being at i, and it equals*

$$q_{ij}^{(n)} = P(X_n = j | X_0 = i).$$

Proof. Now we show the n-step Transition Probability. It follows that

$$\begin{aligned} P(X_n = j | X_0 = i) &= \sum_k P(X_n = j, X_{n-1} = k | X_0 = i) && \text{(by LOTP)} \\ &= \sum_k P(X_n = j | X_{n-1} = k, X_0 = i) P(X_{n-1} = k | X_0 = i) \\ &= \sum_k q_{kj}^{(1)} P(X_{n-1} = k | X_0 = i) && \text{(by Markov Property)} \end{aligned}$$

then by *induction*, we have

$$P(X_n = j | X_0 = i) = q_{ij}^{(n)}.$$

□

With this equation, we have the *Chapman-Kolmogorov Relationship*.

Chapman-Kolmogorov Relationship: *For $m, n \geq 0$, we have*

$$P(X_{m+n} = j | X_0 = i) = \sum_k P(X_m = k | X_0 = i) P(X_n = j | X_0 = k),$$

and

$$P(X_{m+n} = j | X_0 = i) = \sum_k P(X_m = k | X_0 = i) P(X_{m+n} = j | X_m = k).$$

If we have the initial distribution $X_0 \sim \mathbb{P}$, then for $n \geq 0$ and $\forall j$

$$P(X_n = j) = (\mathbb{P} P^{(n)})_j.$$

Further, the Markov Property can be generalized to

$$P(X_{n+1} = j | X_0 = i_0, ..., X_{n-m-1} = i_{n-m-1}, X_{n-m} = i) = P(X_{n+1} = j | X_{n-m} = i) = P_{ij}^{(m+1)}$$

for $m < n$ and $m \geq 0$.

4.3 Classification of States

Depending on whether they are visited over and over again in the long run or are eventually abandoned, the states of a Markov chain can be classified as recurrent or transient.

Recurrent and Transient states: *State i of a Markov chain is recurrent if starting from i, the probability is 1 that the chain will eventually return to i. Otherwise, the state is transient, which means that if the chain starts from i, there is a positive probability of never returning to i.*

Equivalently, we have

State j is recurrent *iff.* $\sum_{n=0}^{\infty} P_{jj}^{(n)} = \infty$; State j is transient *iff.* $\sum_{n=0}^{\infty} P_{jj}^{(n)} < \infty$.

Irreducible and Reducible Chain: *A Markov chain with transition matrix P is irreducible if for any two sates i and j, it is possible to go from i to j in a finite number of steps (with positive probability). That is, for any states i, j there is some positive integer n such that the (i, j) entry of $P^{(n)}$ is positive. A Markov chain that is not irreducible is called reducible.*

In an irreducible Markov chain with a finite state space, all states are recurrent.

Period: *For a Markov chain with transition matrix P, the period of state i, denoted $d(i)$, is the greatest common divisor of the set of possible return times to i. That is,*

$$d(i) = \text{gcd}\{n > 0 \mid P_{ii}^{(n)} > 0\}.$$

If $d(i) = 1$, state i is said to be aperiodic. If the set of return times is empty, set $d(i) = +\infty$.

4.4 Stationary Distribution

Stationary Distribution: *A row vector* $\mathbf{s} = (s_1, ..., s_M)$ *such that* $\sum_i s_i = 1$ *is a stationary distribution for a Markov chain with transition matrix Q if*

$$\sum_i s_i q_{ij} = s_j$$

for all j.

Any irreducible Markov chain has a unique stationary distribution.

Double Stochastic Matrix: *If each column of the transition matrix Q sums to 1, then the uniform distribution over all states, $(1/M, 1/M, ..., 1/M)$, is a stationary distribution. (A nonnegative matrix such that the row sums and the column sums are all equal to 1 is called a doubly stochastic matrix.)*

Convergence to Stationary Distribution: *Let $X_0, X_1, ...$ be a Markov chain with stationary distribution* $\mathbf{s}$ *and transition matrix Q, such that some power Q^m is positive in all entries. (These assumptions are equivalent to assuming that the chain is irreducible and aperiodic.) Then $P(X_n = i)$ converges to s_i as $n \to \infty$. In terms of the transition matrix, Q^n converges to a matrix in which each row is* $\mathbf{s}$.

Ergodic Markov chain: *A Markov chain is called ergodic if it is irreducible, aperiodic, and all states have finite expected return times (positive recurrent).*

For an ergodic Markov chain $X_0, X_1, ...$, there exists a unique, positive, stationary distribution π, which is the limiting distribution of the chain. That is

$$\pi_j = \lim_{n \to \infty} P_{ij}^{(n)}, \ \forall i, j.$$

4.5 Reversibility

Reversibility: *Let $Q = (q_{ij})$ be the transition matrix of a Markov chain. Suppose there is* $\mathbf{s} = (s_1, ..., s_M)$ *with $s_i \geq 0, \sum_i s_i = 1$, such that*

$$s_i q_{ij} = s_j q_{ji}$$

for all pairs of states i and j. This equation is called the reversibility or detailed balance condition, and we say that the chain is reversible with respect to **s** *if it holds. And* **s** *is a stationary distribution of the chain.*

Detailed Balance Equation: *If for an irreducible Markov chain with transition matrix $Q = q_{ij}$, there exists a probability solution π to the detailed balance equations*

$$\pi_i q_{ij} = \pi_j q_{ji}$$

for all pairs of states i and j, then this Markov chain is positive recurrent, time-reversible and the solution π is the unique stationary distribution.

4.6 Markov chain Monte Carlo

Monte Carlo method is a simulation approach where we generate random values to approximate a quantity. A basic form of such method is directly generating i.i.d. draws $X_1, X_2, ..., X_n$ from a given distribution, then by the law of large numbers we can make a desired approximate if n is large. However, staring at a density function does not immediately suggest how to get a random variable with that density. Even we have the *universality of Uniform*, the CDF is difficult to find, let alone its inverse.

Fortunately, for this limitation, we have *Markov chain Monte Carlo* (MCMC), a powerful collection of algorithms, to enable us to simulate from complicated distributions using Markov chains. The basic idea is to *build your own Markov chain* so that the distribution of interest is the stationary distribution of the chain.

Convergence to stationary distribution: *Let $X_0, X_1, ...$ be a Markov chain with stationary distribution s and transition matrix Q, such that the chain is irreducible and aperiodic. Then $P(X_n = i)$ converges to s_i as $n \to \infty$.*

With the above theorem, we can approach the desired s by running our chain for a long time.

Metropolis-Hastings: Metropolis-Hastings allows us to start with any *irreducible* Markov chain on the state space of interest and then modify it into a new Markov chain that has the desired stationary distribution.

Recall: In an irreducible Markov chain, for any two states i and j it is possible to go from i to j in a finite number of steps.

Metropolis-Hastings Algorithm: *Let $s = (s_1, ..., s_M)$ be a desired stationary distribution on state space. Suppose that $P = (p_{ij}$ is the transition matrix for any irreducible Markov chain on state space $\{1, ..., M\}$. Then we use P chain to construct a new Markov chain $X_0, X_1, ...$ with stationary distribution s by the Metropolis-Hastings algorithm:* **Require:**

· The desired distribution $s = (s_1, ..., s_M)$;

· The transition matrix P of the original chain;

· State X_0 where the new chain start at, and it can be chosen randomly or deterministically;

Suppose that the new chain is currently at X_n, do the following.

Repeat:

1. If $X_n = i$, propose a new state j according to P;

2. Compute the acceptance probability $a_{ij} = \min\left(\frac{s_j p_{ji}}{s_i p_{ij}}, 1\right)$;

3. Accept j with probability a_{ij}. If we accept, $X_{n+1} = j$ otherwise $X_{n+1} = i$.

4. Repeat $1 - 4$ until convergence.

The key of the modification is that the moves are proposed according to the original chain, but the proposal may or may not be accepted. By the *reversibility condition*, it can be showed that the sequence $X_0, X_1, ...$ constructed by the Metropolis-Hastings algorithm is a *reversible* Markov chain with stationary distribution.

The generated sequence $X_0, X_1, ..., X_n$ gives an approximate sample from s. Sometimes it may need a long time to converge, and that may because the initial bias. To avoid it, a practical approach is *Burn-in*, which discards the initial iterations and retains $X_m, X_{m+1}, ...$ for some m.

Gibbs Sampler: Gibbs sampling is an MCMC algorithm for obtaining approximate draws from a joint distribution, based on sampling from conditional distributions one at a time: at each stage, one variable is updated (keeping all the other variables fixed) by drawing from the conditional distribution of that variable given all the other variables. This approach is especially useful in problems where these conditional distributions are pleasant to work with.

Depending on the order in which updates are done, there are two major kinds of Gibbs sampler: *systematic scan*, in which the updates sweep through the components in a deterministic order, and *random scan*, in which a randomly chosen component is updated at each stage.

Systematic scan Gibbs sampler: *Let X and Y be discrete r.v.s with joint PMF $p_{X,Y}(x, y) = P(X = x, Y = y)$. We wish to construct a two-dimensional Markov chain (X_n, Y_n) whose stationary distribution is $p_{X,Y}$. The systematic scan Gibbs sampler proceeds by updating the X-component and the Y-component in alternation. If the current state is $(X_n, Y_n) = (x_n, y_n)$, then we update the X-component while holding the Y-component fixed, and then update the Y -component while holding the X-component fixed:*

Require:

· The joint PMF $p_{X,Y}$;

· The initial state (X_0, Y_0);

Suppose that the new chain is currently at (X_n, Y_n), do the following.

Repeat:

1. Draw a value $x_{n+1} \sim X|Y = y_n$, and set $X_{n+1} = x_{n+1}$;

2. Draw a value $y_{n+1} \sim Y|X = x_{n+1}$, and set $Y_{n+1} = y_{n+1}$;

3. Repeat 1 and 2 over and over, the stationary distribution of the chain $(X_0, Y_0), (X_1, Y_1), ...$ is $p_{X,Y}$.

Random scan Gibbs sampler: *As above, let X and Y be discrete r.v.s with joint PMF $p_{X,Y}(x, y)$. We wish to construct a two-dimensional Markov chain (X_n, Y_n) whose stationary distribution is $p_{X,Y}$. Each move of the random scan Gibbs sampler picks a uniformly random component and updates it, according to the conditional distribution given the other component:*

Require:

· The joint PMF $p_{X,Y}$;

· The initial state (X_0, Y_0);

Suppose that the new chain is currently at (X_n, Y_n)*, do the following.*

Repeat:

1. Choose which component to update, with equal probabilities;

2. If the X-component was chosen, draw a value $x_{n+1} \sim X|Y = y_n$, and set $X_{n+1} = x_{n+1},\ Y_{n+1} = y_n$. Similarly, if the Y-component was chosen, draw a value $y_{n+1} \sim Y|X = x_n$, and set $X_{n+1} = x_n,\ Y_{n+1} = y_{n+1}$.

3. Repeat 1 and 2 over and over, the stationary distribution of the chain $(X_0, Y_0), (X_1, Y_1), ...$ is $p_{X,Y}$.

The random scan Gibbs sampler is a special case of the Metropolis-Hastings algorithm, in which the proposal is always accepted. In particular, it follows that the stationary distribution of the random scan Gibbs sampler is as desired.

5 Markov Decision Process

The large part of this section was done with references [1, 2, 3, 4].

5.1 Markov Process

Markov decision processes formally describe an environment for reinforcement learning. Almost all RL problems can be formalized as MDPs.

Markov Property for State: *A state* S_t *is Markovian if and only if*

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ...S_t].$$

The future is independent of the past given the present. Once the state is known, the history may be thrown away.

Markov Chain in Reinforcement Learning: *A discrete-time Markov chain is a tuple* $\langle \mathcal{S}, \mathcal{P} \rangle$ *where*

· $\mathcal{S}$ *is a (finite) set of states;*

· $\mathcal{P}$ *is a state transition probability matrix.*

5.2 Markov Reward Process

Markov Reward Process (MRP): *A Markov Reward Process is a tuple* $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ *where*

· $\mathcal{S}$ *is a (finite) set of states;*

· $\mathcal{P}$ *is a state transition probability matrix;*

· $\mathcal{R}$ *is a reward function;*

· γ *is a discount factor.*

A Markov reward process is a Markov chain with values. For a Markov reward process chain, we can define *return* as follows.

Return: *The return* G_t *is the total discounted reward from time-step* t*:*

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

There are many reasons to consider discounted rewards. One reason is that uncertainty about the future may not be fully represented. And for the case that the reward is financial, immediate rewards may earn more interest than delayed rewards. Besides, animal and human behavior shows preference for immediate reward.

Value Function: *The state value function $v(s)$ of an MRP is the expected return starting from state s*

$$v(s) = E[G_t|S_t = s].$$

Bellman Equation for MRPs: *The value function $v(S_t)$ can be decomposed into two parts: the immediate reward R_{t+1} and the discounted value of successor state $\gamma v(S_{t+1})$, which is*

$$v(S_t) = E[R_{t+1} + \gamma v(S_{t+1})|S_t = t].$$

Proof:

According to the Adam's Law with extra conditioning, it follows that

$$E[E(G_{t+1}|S_{t+1}, S_t)|S_t] = E(G_{t+1}|S_t). \tag{1.1}$$

By the Markov property, the term $(G_{t+1}|S_{t+1}, S_t)$ equals to $(G_{t+1}|S_{t+1})$ when it comes to the conditional probability. Therefore we can arrive at

$$\begin{aligned} E[E(G_{t+1}|S_{t+1}, S_t)|S_t] &= E[E(G_{t+1}|S_{t+1})|S_t] \\ &= E[v(S_{t+1})|S_t]. \end{aligned} \tag{1.2}$$

With (1.1) and (1.2), we can make a conclusion that

$$E(G_{t+1}|S_t) = E[v(S_{t+1})|S_t]. \tag{1.3}$$

Then we have

$$\begin{aligned} v(s) &= E[G_t|S_t = s] \\ &= E[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= E[R_{t+1}|S_t = s] + \gamma E[G_{t+1}|S_t = s] \\ &= E[R_{t+1}|S_t = s] + \gamma E[v(S_{t+1})|S_t = s] \qquad \text{(by (1.3))} \\ &= E[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \end{aligned}$$

□

We can also derive another form of Bellman equation by introducing the law of total expectation:

$$
\begin{aligned}
v(s) &= E[R_{t+1}|S_t = s] + \gamma E[v(S_{t+1})|S_t = s] \\
&= R_s + \gamma \sum_{s' \in S} E[v(S_{t+1})|S_t = s, S_{t+1} = s']P(S_{t+1} = s'|S_t = s) \qquad \text{(by LOTE)} \\
&= R_s + \gamma \sum_{s' \in S} v(s')P_{ss'}
\end{aligned}
$$

Though Bellman equation is elegant, to solve it directly is only possible for small MRPs. For large MRPs, one can refer to iterative methods such as *dynamic programming*, *Monte-Carlo evaluation* and *Temporal-Difference learning* .

5.3 Markov Decision Process

Markov Decision Process (MDP): *A Markov Decision Process is a tuple* $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$*, where*

· $\mathcal{S}$ *is a (finite) set of states;*

· $\mathcal{A}$ *is a finite set of actions;*

· $\mathcal{P}$ *is a state transition probability matrix* $P(S_{t+1} = s'|S_t = s, A_t = a)$;

· $\mathcal{R}$ *is a reward function* $R(S_t = s, A_t = a) = E[R_t|S_t = s, A_t = a]$;

· γ *is a discount factor.*

A Markov decision process is a Markov reward process with decisions (actions). It is an environment in which all states are Markovian.

Policy: *A policy* π *is a distribution over actions given states,*

$$\pi(a|s) = P(A_t = a|S_t = s).$$

A policy can fully define the behavior of an agent, and it depends only on the current state. Then similar to MRP, we have the following functions.

State-value Function: *The state-value function* $v^\pi(s)$ *of an MDP is the expected return starting from state* s*, and following policy* π

$$v^\pi(s) = E_\pi[G_t|S_t = s]$$

Action-value Function: *The action-value function* $q_\pi(s, a)$ *is the expected return starting from state* s*, taking action* a*, and then following policy* π*, which is*

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a].$$

The relation between $v^\pi(s)$ and $q^\pi(s, a)$ follows that

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) q^\pi(s, a)$$

$$q^\pi(s, a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v^\pi(s')$$

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) \left(R(s, a) + \gamma \sum_{s' \in S} P\left(s'|s, a\right) v^\pi\left(s'\right) \right)$$

$$q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P\left(s'|s,a\right) \sum_{a' \in A} \pi\left(a'|s'\right) q^{\pi}\left(s',a'\right)$$

where the last two equations derived by the first two.

Bellman Expectation Equation: *The state-value function can be decomposed into immediate reward plus discounted value of the successor state, and the action-value function can similarly be decomposed:*

$$v^{\pi}(s) = E_{\pi}[R_{t+1} + \gamma v^{\pi}(S_{t+1})|S_t = s];$$
$$q^{\pi}(s,a) = E_{\pi}[R_{t+1} + \gamma q^{\pi}(S_{t+1}, A_{t+1})|S_t = t, A_t = a].$$

Optimal Value Function: *The optimal state-value function $v_*(s)$ is the maximum value function over all polices:*

$$v_*(s) = \max_{\pi} v_{\pi}(s).$$

Optimal Action-value Function: *The optimal action-value function $q_*(s,a)$ is the maximum value function over all polices:*

$$q_*(s,a) = \max_{\pi} q_{\pi}(s,a).$$

Optimal Policy: *For any Markov Decision Process, there exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi,\ \forall\pi$. Further, all optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$ and all optimal polices achieve the optimal action-value function, $q_{\pi_*}(s,a) = q_*(s,a)$.*

Prediction: *Given an MDP and a policy π, find the value function v^{π}.*

Control: *Given an MDP, find the optimal value v^* and optimal policy π^*.*

An MDP is *solved* when we know the optimal value function.

5.4 Dynamic Programming

Dynamic programming is a method for solving complex problems by breaking them down into subproblems. Specifically, the problems should be characterized by the two properties:

· Optimal substructure, which entails that optimal solution can be decomposed into subproblems;

· Overlapping subproblems, which entails that the solutions for subproblems can be cached and reused.

Obviously, MDPs satisfy both properties. Prediction and control in MDP can be solved by dynamic programming.

5.4.1 Policy Evaluation

Bellman expectation backup: *At each iteration $k+1$, for all states $s \in \mathcal{S}$, update $v_{k+1}(s)$ from $v_k(s')$ as*

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right),$$

where s' is a successor state of s, $\mathcal{A}$ is the action space and $\mathcal{P}$ is the transition matrix.

Policy evaluation aims to evaluate a given policy's reward distribution.

5.4.2 Policy Iteration

For a deterministic policy $a = \pi(s)$, we can iterate through the two steps:

· Evaluate the policy π so that get the value function;

· Improve the policy by acting greedily with respect to q:

$$\pi'(s) = \arg\max_a q_\pi(s, a),$$

where $q_\pi(s, a)$ is the expected return starting from state s, taking action a following the policy π.

With this iteration, we can improve the value from any state s over one step and therefore improve the value function, as

$$\begin{aligned}
v_\pi(s) &= q_\pi(s, \pi(s)) \\
&\leq \max_a q_\pi(s, a) \\
&= q_\pi(s, \pi'(s)) \\
&= E_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = t] \\
&\leq E_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1}))|S_t = t] \\
&\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2}))|S_t = s] \qquad (1.1) \\
&\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + ...|S_t = s] \\
&= v_{\pi'}(s).
\end{aligned}$$

The step (1.1) can be derived by Adam and Markov theory. When the improvements stop, we have

$$q^\pi(s, \pi'(s)) = \max_a q^\pi(s, a) = q^\pi(s, \pi(s)) = v^\pi(s),$$

thus the Bellman optimality equation has been satisfied. Therefore it gives us the optimal policy.

Generalized Policy Iteration(GPI): *The combination of policy evaluation and policy improvement is called generalized policy iteration, which is the basic idea of Actor-Critic.*

5.4.3 Value Iteration

Bellman Optimality Equation: *The optimal value functions are reached by the Bellman optimality equations*:

$$v^*(s) = \max_a q^*(s, a)$$
$$q^*(s, a) = R_s^a + \gamma \sum_{s' \in S} P\left(s'|s, a\right) v^*\left(s'\right)$$

then

$$v^*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P\left(s'|s, a\right) v^*\left(s'\right)$$
$$q^*(s, a) = R_s^a + \gamma \sum_{s' \in S} P\left(s'|s, a\right) \max_{a'} q^*(s', a')$$

If we know the solution to subproblem $v^*(s')$, which is optimal, then the solution for the optimal $v^*(s)$ can be found by iteration over the following *Bellman Optimality backup* rule

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} P\left(s'|s,a\right) v\left(s'\right) \right).$$

5.4.4 Comparison

The comparison of the two methods are shown as follows.

Policy Iteration: policy evaluation + policy improvement

- Picks a policy and then determines the true, steady *state value* of being in each state given the policy;
- Given this value, a new policy is chosen;
- Converges faster in terms of the number of iterations since it doing a lot more work in each iteration.

Value Iteration: optimal value function + one policy extraction

- Updates the value *greedily* (does not care the policy) at each iteration and then, *after* finding the optimal value function, determines a new policy given the new estimation of the value function;
- At any iteration, the value function is not the true, steady state value of the policy;
- Convergences fast per iteration, may be far from the true value function.

The summary for prediction and control in MDP by dynamic programming is shown in Table 1.

Problem	Bellman Equation		Algorithm
Prediction	Expectation	$v^\pi(s) = E_\pi[R_{t+1} + \gamma v^\pi(S_{t+1})\|S_t = s]$	Policy
		$q^\pi(s,a) = E_\pi[R_{t+1} + \gamma q^\pi(S_{t+1}, A_{t+1})\|S_t = t, A_t = a]$	Evaluation
Control	Expectation	$v^\pi(s) = E_\pi[R_{t+1} + \gamma v^\pi(S_{t+1})\|S_t = s]$	Policy
		$q^\pi(s,a) = E_\pi[R_{t+1} + \gamma q^\pi(S_{t+1}, A_{t+1})\|S_t = t, A_t = a]$	Iteration
Control	Optimality	$v^*(s) = \max_a R_s^a + \gamma \sum_{s' \in \mathcal{S}} P(s'\|s,a) v^*(s')$	Value
		$q^*(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} P(s'\|s,a) \max_{a'} q^*(s',a')$	Iteration

Table 1: Dynamic Programming algorithms for MDPs

6 Model-free Prediction

In the unknown MDP, we have no idea about its transition matrix and its state, action spaces, thus we can only *learn by interaction*. No more direct access to the known transition dynamics and reward function. Episodes are collected by the agent's interaction with the environment.

6.1 Monte-Carlo Policy Evaluation

Monte-Carlo (MC) methods learn directly from *complete* episodes of experience, thus it needs no knowledge of MDP transitions or rewards. The limit of MC methods is that it entails MDPs are episodic.

Monte-Carlo Policy Evaluation: *Given some episodes which contain state s under the policy π, we can approximate the value of s by the average returns observed after visits to s.*

Depending on when average returns for state s in an episode, there are two different implementation.

First-Visit Monte-Carlo Policy Evaluation: *Only at the first time-step t that state s is visited in an episodes, we do the following procedure:*

- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Update the value by mean return $V(s) = S(s)/N(s)$

Every-Visit Monte-Carlo Policy Evaluation: *Every time-step t that state s is visited in an episode, we do the following procedure:*

- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Update the value by mean return $V(s) = S(s)/N(s)$

By *law of large numbers*, both of two methods can achieve $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$. For the calculation part, we can use a trick *incremental mean* to update the mean

$$N(S_t) \leftarrow N(S_t) + 1,$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t)).$$

Differences between DP and MC for policy evaluation:

- DP computes v_i by bootstrapping the rest of the expected return by the value estimate v_{i-1};
- DP iterates on Bellman expectation backup:

$$v_i(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_{i-1}(s') \right).$$

- MC updates the empirical mean return with one sampled episode:

$$v(S_t) \leftarrow v(S_t) + \alpha(G_{i,t} - v(S_t)).$$

Advantages of MC over DP:

- MC can work when the environment is unknown;
- Working with sample episodes has a huge advantage. Even when one has complete knowledge of the environment’s dynamics, for example, transition probability is complex to compute;
- Cost of estimating a single state’s value is independent of the total number of states. So you can sample episodes starting from the states of interest then average returns.

6.2 Temporal-Difference Learning

The difference between temporal-difference (TD) and MC methods is that the former does not entail the episodes are complete. TD methods can learn from incomplete episodes by bootstrapping.

Temporal-Difference Learning: *Given some incomplete episodes which contain state s under the policy π, we update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1}$:*

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)),$$

where $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target, α is the step-size, and we call

$$\delta_t = R_{t+1} + \gamma V(S_{t+1} - V(S_t)$$

the TD error.

Differences between MC and TD for policy evaluation:

- TD can learn online after every step;
- MC must wait until end of episode before return is known;
- TD can learn from incomplete sequences;
- MC can only learn from complete sequences;
- TD works in continuing (non-terminating) environments;
- MC only works for episodic (terminating) environments;
- TD exploits Markov property, more efficient in Markov environments;
- MC does not exploit Markov property, more effective in non-Markov environments.

Bootstrapping: update involves an estimate.

Sampling: update samples an expectation.

A summary of *bootstrapping* and *sampling* for DP, MC, and TD is shown in Table 2.

Algorithm	Bootstraps	Sampling
Dynamic Programming	$\surd$	
Monte-Carlo		$\surd$
Temporal-Difference	$\surd$	$\surd$

Table 2: *Bootstrapping* and *sampling* for DP, MC, and TD.

n-step TD methods: *Unlike the simplest TD method, the updating rule of value $V(S_t)$ is*

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t)),$$

where $G_t^{(n)}$ is the n-step return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^n V(S_{t+n}).$$

Since at time t the $V(S_{t+n})$ is not available, thus the natural algorithm is to wait until then, then it turns out to be

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha \left[G_t^{(n)} - V_{t+n-1}(S_t) \right].$$

Also, notice that we can generalize TD to MC when $n \to \infty$.

TD(λ) methods: *To make use of the information from all time-steps, we can use weight $(1-\lambda)\lambda^{n-1}$ to average n-step returns over different n as*

$$G_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)},$$

thus the updating rule for the value becomes

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{\lambda} - V(S_t)).$$

7 Model-free Control

Model-free control leverage the idea of *generalized policy iteration*(GPI). Since there is neither R_s^a nor $P(s'|s,a)$ known/available, a way to solve that refers to sample episodes and learn from them. For a large state space, as the way we sample matters a lot, we define *behavior policy* that determines which action to take and *target policy* that determines the best policy we have so far, then based on the two concepts, we have two learning form:

- **On-policy** learning: when the target policy and the behavior policy are the same, which entails us to update the behavior policy every round;
- **Off-policy** learning: when the target policy and the behavior policy are different, which requires us to only update the target policy.

On-policy may not ensure the enough exploration of state space as when we update the behavior policy we may greedily discard those states with great potential. Compared to on-policy learning, off-policy learning is more powerful and general, while it is often of greater variance and slower to converge.

7.1 On Policy Monte-Carlo Control

The following algorithms are based on the implementation of Monte-Carlo, and the differences lie in how they generate samples.

Exploring Starts: *One assumption to obtain the guarantee of convergence in π is that episodes explore starts, which means the beginning of each episode is randomly chose:*

Algorithm 2: Monte-Carlo Exploring Starts algorithm

Initialization:

$\pi(s) \in \mathcal{A}(s)$(arbitrarily), for all $s \in \mathcal{S}$;

$Q(s,a) \in \mathbb{R}$(arbitrarily), for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$;

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$;

Loop forever:

Choose $S_0 \in \mathcal{S}, A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0;

Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$;

$G \leftarrow 0$;

Loop for each step of episode, $t = T-1, T-2, ..., 0$**:**

$G \leftarrow \gamma G + R_{t+1}$;

Unless the pair S_t, A_t appears in the episode:

Append G to $Returns(S_t, A_t)$;

$Q(S_t, A_t) \leftarrow average(Returns(S_t, A_t))$;

$\pi(S_t) \leftarrow \arg\max_n Q(S_t, a)$;

Random starts sometimes are not common in practice. In the iteration part, we can leverage $\epsilon-$*Greedy Exploration* to avoid the unlikely assumption of exploring starts.

$\epsilon-$**Greedy Exploration**: *For a state with m actions, there is non-zero probability to try them:*

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon, & a = \arg\max_{a' \in \mathcal{A}} Q(s,a) \\ \frac{\epsilon}{m}, & \text{otherwise} \end{cases}$$

which means we will choose the greedy action with probability $1-\epsilon$ and choose an action at random with probability ϵ.

Algorithm 3: Monte-Carlo ϵ-greedy Exploration algorithm

Initialization:

$Q(s,a) = 0, N(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$;

$k = 1, \epsilon = 1$;

$\pi_k = \epsilon - greedy(Q)$;

Loop forever:

Generate k-th episode following π_k: $S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T$;

Loop for each step of episode, $t = T-1, T-2, ..., 0$**:**

$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$;

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$;

$k \leftarrow k+1, \epsilon = 1/k$;

$\pi_k = \epsilon - greedy(Q)$;

Policy improvement theorem: *For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v'_\pi(s) \geq v_\pi(s)$.*

Proof:

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s,a) \\ &= \frac{\epsilon}{m} \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \max_a q_\pi(s,a) \\ &\geq \frac{\epsilon}{m} \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \frac{\epsilon}{m}}{1-\epsilon} q_\pi(s,a) \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a) = v_\pi(s) \end{aligned}$$

□

Greedy in the Limit with Infinite Exploration(GLIE): *All state-action pairs are explored infinitely many times,*

$$\lim_{k \to \infty} N_k(s,a) = \infty.$$

The policy converges on a greedy policy,

$$\lim_{k \to \infty} \pi_k(a|s) = \mathbf{1}(a = \arg\max_{a' \in \mathcal{A}} Q_k(s,a')).$$

For ϵ-greedy, when we explore, we choose actions at random without regard to their estimated values. To focus on those states with high value, we can refer to *Boltzmann exploration*.

Boltzmann Exploration: *Boltzmann exploration also known as Gibbs sampling and soft-max, it chooses an action based on its estimated value:*

$$\pi(a|s) = \frac{e^{\beta \hat{Q}(s,a)}}{\sum_{a'} e^{\beta \hat{Q}(s,a')}},$$

where $\hat{Q}(s,a)$ is an estimation of the value of being in state s and taking action a, β is a tunable parameter.

7.2 On Policy Temporal-Difference Control

As we mentioned before, Temporal-Difference(TD) learning has several advantages over Monte-Carlo (MC) such as lower variance, online, incomplete sequences. So we can use TD instead of MC in our control loop. What we need to do is applying TD to $Q(S,A)$ and updating every time-step rather than at the end of one episode.

Sarsa: *Consider transitions from state-action pair to state-action pair, and learn the values of state-action pairs:*

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

Algorithm 4: TD Sarsa algorithm

Initialization:

$Q(s,a) \in \mathbb{R}$(arbitrarily), for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$;

$Q(terminal - state) = 0$;

Loop:

Initialize S;

Choose $A \in \mathcal{A}(S)$ following $\epsilon - greedy(Q)$;

Loop for each step of episode:

Take action A, observe R, S';

Choose $A' \in \mathcal{A}(S')$ following $\epsilon - greedy(Q)$;

$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S', A') - Q(S,A)]$;

$S \leftarrow S', A = A'$;

Until S is terminal;

This rule uses every element of the quintuple of events, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, which give rise to the name $Sarsa$. There are also n-step Sarsa, Sarsa(λ) that derived from TD methods.

n-step Sarsa: *Define the n-step Q-return as*

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n}),$$

then n-step Sarsa updates $Q(s,a)$ towards the n-step Q-return

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(q_t^{(n)} - Q(S_t, A_t)).$$

Like what we mentioned in n-step TD, when $n \to \infty$, Sarsa$\to$MC.

7.3 Off-Policy Q-Learning Control

Though in off-policy the target policy and the behavior policy are different, we now allow both policies to be improved.

For target policy, Q-learning updates it in a *greedy* way:

$$\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a').$$

For behavior policy, it will be updated in an ϵ-*greedy* way on $Q(s,a)$.

The state-action value will be updated as

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max_{a'} Q(S', a') - Q(S, A)).$$

Algorithm 5: Q-learning algorithm

Initialization:

$Q(s,a) \in \mathbb{R}$(arbitrarily), for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$;

$Q(terminal - state) = 0$;

Loop:

Initialize S;

Choose $A \in \mathcal{A}(S)$ following $\epsilon - greedy(Q)$;

Take action A, observe R, S';

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$;

$S \leftarrow S'$;

Until S is terminal;

One should notice that the action it uses to update is not the same as the one it truly takes.

The comparison of Sarsa and Q-Learning is shown as follows:

Sarsa: On-Policy TD control

- Choose action A_t from S_t using policy derived from Q with ϵ-greedy;
- Take action A_t, observe R_{t+1} and S_{t+1};
- Choose action A_{t+1} from S_{t+1} using policy derived from Q with ϵ-greedy;
- $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$.

Q-Learning: Off-Policy TD control

- Choose action A_t from S_t using policy derived from Q with ϵ-greedy;
- Take action A_t, observe R_{t+1} and S_{t+1};
- 'Imagine' A_{t+1} as $\arg\max_{a'} Q(S_{t+1}, a')$ in the update target;
- $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)]$.

7.4 Off-Policy Importance Sampling Control

Off-Policy control allows us to learn from observing humans or other agents, re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_t$, learn about optimal policy while following exploratory policy and learn about multiple policies while following one policy.

Importance Sampling for Off-Policy Monte-Carlo: *Let G_t be the returns generated from μ. Define the corrected return*

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)\pi(A_{t+1}|S_{t+1})...\pi(A_\tau|S_\tau)}{\mu(A_t|S_t)\mu(A_{t+1}|S_{t+1})...\mu(A_\tau|S_\tau)}G_t,$$

then we update value by the corrected return

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{\pi/\mu} - V(S_t)).$$

By the update rule, we know the μ can not be zero when π is non-zero. It should be noticed that importance sampling can dramatically increase the variance.

Importance Sampling for Off-Policy TD: *Weight TD target $R+\gamma V(S')$ by importance sampling, then we only need a single importance sampling correction to update the value*

$$V(S_t) \leftarrow V(S_t) + \alpha\left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}(R_{t+1} + \gamma V(S_{t+1})) - V(S_t)\right).$$

8 Value Function Approximation

8.1 Introduction on Function Approximation

Reinforcement learning can be used to solve large problems, *e.g.*:

- Backgammon: 10^{20} states;
- Chess: 10^{47} states;
- Came of Go: 10^{170} states;
- Robot Arm and Helicopter: continuous state space;

How can we scale up the model-free methods for prediction and control?

A solution for these refers to *supervised learning*, in particularly, to estimate with *function approximation*:

- $\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$
- $\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$
- $\hat{\pi}(s, a, \boldsymbol{w}) \approx \pi(a|s)$

where $\boldsymbol{w}$ is the parameter learned by MC or TD learning. With such approximation, we can generalize from seen states to unseen states.

Obviously, we have many possible function approximators:

- Linear combinations of features;

- Neural networks;
- Decision trees;
- Nearest neighbors;

among them, we will focus on the first two as they are differentiable and thus easy to be optimized.

8.2 Incremental Method

Now we consider a naive case where we have an oracle for knowing the true value for $v^\pi(s)$ for any given state s. Then the object is to find the best approximate representation of $v^\pi(s)$. Thus we can define the loss function by the mean squared error:

$$J(\boldsymbol{w}) = E_\pi[(v^\pi(s) - \hat{v}(s, \boldsymbol{w}))^2],$$

and the gradient descend for the loss function is:

$$\Delta \boldsymbol{w} = -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w}),$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \Delta \boldsymbol{w}.$$

Follow the gradient descend, we can find a local minimum. Further, if the value function is represented by a linear combination of features, then such method can converge to the global optimum.

8.2.1 Incremental Method for Prediction

However, in practice, no access to oracle of the true value $v^\pi(s)$ for any state s. What we have is the reward. Thus we can substitute the target for $v^\pi(s)$:

- For MC, the target is the actual return G_t and hence

$$\Delta \boldsymbol{w} = \alpha(G_t - \hat{v}(s_t, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{v}(s_t, \boldsymbol{w});$$

- For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{v}(s_{t+1}, \boldsymbol{w})$ and hence

$$\Delta \boldsymbol{w} = \alpha(R_{t+1} + \gamma \hat{v}(s_{t+1}, \boldsymbol{w}) - \hat{v}(s_t, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{v}(s_t, \boldsymbol{w})$$

For TD(0), the gradient is also called as semi-gradient, as we ignore the effect of changing the weight vector $\boldsymbol{w}$ on the target.

With these new gradient, we can update the approximation we mentioned in section 8.1, then we have *Gradient Monte Carlo* method and *Semi-gradient TD(0)* method for value prediction.

8.2.2 Incremental Method for Control

The control exploits the *generalized policy iteration* similarly, with $\hat{q}(\cdot, \cdot, \boldsymbol{w})$. Same to the prediction, there is no oracle for the true value $q^\pi(s, a)$, so we substitute a target

- For MC, the target is return G_t

$$\Delta \boldsymbol{w} = \alpha(G_t - \hat{q}(s_t, a_t, \boldsymbol{w}))\nabla \hat{q}(s_t, a_t, \boldsymbol{w})$$

- For Sarsa, the target is TD target $R_{t+1} + \gamma\hat{q}(s_{t+1}, a_{t+1}, \boldsymbol{w})$:

$$\Delta\mathbf{w} = \alpha\left(R_{t+1} + \gamma\hat{q}\left(s_{t+1}, a_{t+1}, \mathbf{w}\right) - \hat{q}\left(s_t, a_t, \mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{q}\left(s_t, a_t, \mathbf{w}\right)$$

- For Q-learning, the target is TD target $R_{t+1} + \gamma \max_a \hat{q}\left(s_{t+1}, a, \mathbf{w}\right)$:

$$\Delta\mathbf{w} = \alpha\left(R_{t+1} + \gamma\max_a\hat{q}\left(s_{t+1}, a, \mathbf{w}\right) - \hat{q}\left(s_t, a_t, \mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{q}\left(s_t, a_t, \mathbf{w}\right)$$

Algorithm 6: Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input:

a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$;

Initialization:

$\boldsymbol{w} \in \mathbb{R}^d$(arbitrarily);

Loop forever:

Choose $S \in \mathcal{S}, A \in \mathcal{A}(S)$ under $\epsilon - greedy$ policy;

Loop for each step of episode:

Take action A, observe R, S';

If S' is terminal:

$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha[R - \hat{q}(S, A, \boldsymbol{w})]\nabla\hat{q}(S, A, \boldsymbol{w})$;

Go to next episode;

Choose A' as a function of $\hat{q}(S', \cdot, \boldsymbol{w})$ under $\epsilon - greedy$ policy;

$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha[R + \gamma\hat{q}(S', A', \boldsymbol{w}) - \hat{q}(S, A, \boldsymbol{w})]\nabla\hat{q}(S, A, \boldsymbol{w})$;

$S \leftarrow S'$;

$A \leftarrow A'$;

Then the policy is made according to the $\hat{q}$.

Now consider the convergence of *Control Methods* with *value function approximation*(VFA).

- For Sarsa:

$$\Delta\mathbf{w} = \alpha\left(R_{t+1} + \gamma\hat{q}\left(s_{t+1}, a_{t+1}, \mathbf{w}\right) - \hat{q}\left(s_t, a_t, \mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{q}\left(s_t, a_t, \mathbf{w}\right)$$

- For Q-Learning:

$$\Delta\mathbf{w} = \alpha\left(R_{t+1} + \gamma\max_a\hat{q}\left(s_{t+1}, a, \mathbf{w}\right) - \hat{q}\left(s_t, a_t, \mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{q}\left(s_t, a_t, \mathbf{w}\right)$$

Firstly, TD with VFA follows the *semi-gradient* rather than the true gradient of any objective function. Secondly, the updates involve doing an approximate Bellman backup (model-free) followed by fitting the underlying value function (approximation). That is why TD can diverge when off-policy or using non-linear function approximation. Further, for off-policy, behavior policy and target policy are not identical, thus value function approximation can diverge.

The Deadly Triad for the Danger of Instability and Divergence:

- Function approximation: A scalable way of generalizing from a state space much larger than the memory and computational resources;
- Bootstrapping: Update targets that include existing estimates (as in dynamic programming or TD methods) rather than relying exclusively on actual rewards and complete returns (as in MC methods);
- Off-policy training: training on a distribution of transitions other than that produced by the target policy.

8.3 Batch Methods

The incremental method or, without loss of generalization, on-policy learning requires the agent gradually gathers experience in the environment. When the agent cannot interact further with the environment, then we need to learn from the limited date to find the best fitting value function. To achieve that, a trick named *experience replay* is used.

Experience Replay: *It allows reusing samples from a different behavior policy.*

Experience replay is sample efficient since any experience can be used. However, it usually introduces a bias when used with a relay buffer, as the trajectories are usually not obtained solely under the current policy.

8.3.1 Least Squares Prediction

Given value function approximation $\hat{v}(s, \boldsymbol{w})$ and the experience $\mathcal{D}$ consisting of $\langle s, v^\pi \rangle$ pairs, *Least Squares* algorithm try to find the best $\boldsymbol{w}$ to minimize the sum-squared error between the approximation $\hat{v}(s, \boldsymbol{w})$ and the target value $\hat{v}^\pi(s)$:

$$\min_{\boldsymbol{w}} \sum_{s, v^\pi(s) \in \mathcal{D}} (v^\pi(s) - \hat{v}(s, \boldsymbol{w}))^2.$$

Referring to SGD, we can apply stochastic gradient descent update as

$$\Delta \boldsymbol{w} = \alpha(v^\pi - \hat{v}(s, \boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{v}(s, \boldsymbol{w}),$$

where we sample state, value from experience as

$$\langle s, v^\pi \rangle \sim \mathcal{D}.$$

8.3.2 Least Squares Control

For policy evaluation, we want to efficiently use all experience. For control, we also want to improve the policy. However, the experience $\mathcal{D}$ may be gathered from many policies. So to evaluate $q_\pi(S, A)$ we must learn off-policy. One way is to use the same idea as Q-Learning:

- Use experience generated by old policies

$$S_t, A_t, R_{t+1}, S_{t+1} \sim \pi_{\text{old}};$$

- Consider the alternative successor action

$$A' = \pi_{\text{new}}(S_{t+1});$$

- Update $\hat{q}$ towards the value of the alternative action

$$\hat{q}(S_t, A_t, \boldsymbol{w}) \sim R_{t+1} + \gamma \hat{q}(S_{t+1}, A', \boldsymbol{w}).$$

8.4 Deep Q-Learning

Deep Q-learning leverages nonlinear function approximator, deep neural networks, to avoid manual designing of features. Such method is also called DQN, and it is a well-known case of *deep reinforcement learning*.

Deep Reinforcement Learning:

- Frontier in machine learning and artificial intelligence;
- Deep neural networks can be used to represent value function, policy function, and even model;
- Optimize loss function by stochastic gradient descent;

Apart from the introducing of neural network, two important techniques *experience replay* and *fixed target* also matter a lot.

DQNs with Experience Replay

- To reduce the correlations among samples, store transition (s_t, a_t, r_t, s_{t+1}) in replay memory $\mathcal{D}$;
- To perform experience replay, repeat the following
  - sample an experience tuple from the dataset: $(s, a, r, s') \sim \mathcal{D}$;
  - compute the target value for the sampled tuple: $r + \gamma \max_{a'} \hat{Q}(s', a', \boldsymbol{w})$;
  - use stochastic gradient descent to update the network weights

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a', \boldsymbol{w}) - Q(s, a, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a, \boldsymbol{w}).$$

DQNs with Fixed Targets

- To help improve stability, fix the target weights used in the target calculation for multiple updates;
- Let a different set of parameter $\boldsymbol{w}^-$ be the set of weights used in the target, and $\boldsymbol{w}$ be the weights that are being updated;
- To perform experience replay with fixed target, repeat the following
  - sample an experience tuple from the dataset: $(s, a, r, s') \sim \mathcal{D}$;
  - compute the target value for the sampled tuple:

$$r + \gamma \max_{a'} \hat{Q}(s', a', \boldsymbol{w}^-)$$

  - use stochastic gradient descent to update the network weights:

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a', \boldsymbol{w}^-) - Q(s, a, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a, \boldsymbol{w}).$$

9 Policy Optimization

For value-based reinforcement learning, deterministic policy is generated directly from the value function using greedy $a_t = \arg\max_a Q(a, s_t)$. Now instead we can parameterize the policy function as $\pi_\theta(a|s)$ where θ is the learnable policy parameter and the output is a probability over the action set.

Value-based Reinforcement Learning versus Policy-based Reinforcement Learning

- Value-based Reinforcement Learning(RL): solve RL problems through dynamic programming
  - related to classic RL and control theory;
  - learns value function;
  - generates an implicit policy based on the value function;
  - learns a deterministic policy;
  - developed by Richard Sutton, David Silver, DeepMind;
- Policy-based Reinforcement Learning(RL): solve RL problems mainly through learning
  - refers to machine learning and deep learning;
  - has no value function;
  - learns policy directly;
  - can learn a stochastic policy;
  - developed by Pieter Abbeel, Sergey Levine, OpenAI, Berkeley;

The two methods can also be combined together. A popular algorithm called *Actor-Critic* entails learning both policy and value function.

Pros and cons of Policy-based RL

- Advantages:
  - can converge on a local optimum (worst case) or global optimum (best case);
  - is effective in high-dimensional action space;
- Disadvantages:
  - typically converges to a local optimum;
  - evaluating a policy has high variance;

9.1 Policy Optimization

Objective of Optimization Policy: *Given a policy approximator $\pi(s, a)$ with parameter θ, find the best θ.*

One thing we care is how do we measure the quality of a policy π_θ? Let τ be a trajectory sampled from the policy function π_θ, then we defined the value of policy π_θ as

$$J(\theta) = E_\tau \left[\sum_t r(s_t, a_t^\tau) \right].$$

Thus we have the goal of policy-based reinforcement learning as

$$\theta^* = \arg\max_\theta E_\tau \left[\sum_t r(s_t, a_t^\tau) \right].$$

However, such $J(\theta)$ may not available or convenient. Hence a trick is using approximation. For example,

- In episodic environment, we can use the value of the starting state s_0:

$$J(\theta) = V^{\pi_\theta}(s_0) = E_{\pi_\theta}[V(s_0)]$$

- In the environment without the terminal state, we can use the average value:

$$J(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

  where d^{π_θ} is the stationary distribution of Markov chain for π_θ.

Depends on the form of $J(\theta)$, we have different methods to maximize it

- If $J(\theta)$ is differentiable, we can use gradient-based methods:
  - Gradient Ascend;
  - Conjugate Gradient;
  - Quasi-newton.
- If $J(\theta)$ is non-differentiable or hard to compute the derivative, we can use some derivative-free black-box optimization methods:
  - Cross-entropy Method (CEM);
  - Hill Climbing;
  - Evolution Algorithm;
  - Approximate Gradients by Finite Difference.

In this note, we mainly focus on gradient-based methods.

9.2 Monte-Carlo Policy Gradient

We now consider the policy gradient for the case where we can get episodes.

Assume policy π_θ is differentiable whenever it is no-zero, and we can compute the gradient $\nabla_\theta \pi_\theta(s, a)$. Then with the *ratio likelihood trick* we have

$$\begin{aligned} \nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a), \end{aligned}$$

where we call $\nabla_\theta \log \pi_\theta(s, a)$ the *score function*.

9.2.1 One-Step Policy Gradient

We first consider policy gradient for one-step MDPs. Starting from the state s sampled from a distribution $d(s)$ and taking action $a \sim \pi_\theta(s)$, we have the reward $r = R_s^a$. Hence we have

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta E_{\pi_\theta}[r] \\
&= \nabla_\theta \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a) r \\
&= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) r \\
&= E_{\pi_\theta}[r \nabla_\theta \log \pi_\theta(s,a)],
\end{aligned}$$

where we use the ratio likelihood trick again.

9.2.2 Multi-Steps Policy Gradient

Now we consider policy gradient for multi-steps MDPs. Starting from the state s_0 sampled from a distribution $d(s)$, we denote one episode as

$$\tau = (s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T, s_t) \sim (\pi_\theta, P(s_{t+1}|s_t, a_t)),$$

and the sum of rewards over the trajectory τ as

$$R(\tau) = \sum_{t=0}^{T} R(s_t, a_t).$$

Let $\mathcal{P}(\tau;\theta) = d(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t)$ denote the probability over trajectories when executing the policy π_θ. Then the policy gradient of $J(\theta)$ is

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta E_{\pi_\theta} \left[\sum_{t=0}^{T} R(s_t, a_t) \right] \\
&= \nabla_\theta \sum_{\tau} \mathcal{P}(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \nabla_\theta \mathcal{P}(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \frac{\mathcal{P}(\tau;\theta)}{\mathcal{P}(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \mathcal{P}(\tau;\theta) R(\tau) \frac{\nabla_\theta \mathcal{P}(\tau;\theta)}{\mathcal{P}(\tau;\theta)} \\
&= \sum_{\tau} \mathcal{P}(\tau;\theta) R(\tau) \nabla_\theta \log \mathcal{P}(\tau;\theta) \\
&\approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \nabla_\theta \log \mathcal{P}(\tau_i;\theta),
\end{aligned}$$

which means the gradient of the policy can be obtained by approximating with empirical estimate for m sample paths under policy π_θ.

We now show that such method does not need the dynamics of the model. Considering the term that matters in the gradient we got above, it follows that

$$\begin{aligned}\nabla_\theta \log \mathcal{P}(\tau;\theta) &= \nabla_\theta \log \left[\mu\left(s_0\right) \prod_{t=0}^{T-1} \pi_\theta\left(a_t|s_t\right) p\left(s_{t+1}|s_t,a_t\right)\right] \\ &= \nabla_\theta \left[\log \mu\left(s_0\right) + \sum_{t=0}^{T-1} \log \pi_\theta\left(a_t|s_t\right) + \log p\left(s_{t+1}|s_t,a_t\right)\right] \\ &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta\left(a_t|s_t\right),\end{aligned}$$

and thus

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\right).$$

It shows that the dynamics, the transition matrix, of the model is not needed, which means policy gradient is a model-free method.

9.2.3 Variance Reduction

For approximating update

$$\nabla_\theta J(\theta) = \nabla_\theta E_{\tau\sim\pi_\theta}[R] \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\right),$$

it is equivalent to

$$\nabla_\theta E_{\tau\sim\pi_\theta}[R] = E_{\tau\sim\pi_\theta}\left[R(\tau)\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\right].$$

It is unbiased but very noisy. To reduce the variance, one can refer to *use temporal causality* and *include a baseline*.

Temporal Causality

In above update rule, notice that $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$ is the sum of rewards over a trajectory τ. While at time t', the reward $r_{t'}$ should be irrelevant to the state after time t'. What matters is only the history before time t'. Therefore we can shrink the term $\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)$ to reduce the variance. Derive the gradient estimator for a single reward term $r_{t'}$ as

$$\nabla_\theta E_{\tau\sim\pi_\theta}[r_{t'}] = E_{\tau\sim\pi_\theta}\left[r_{t'}\left(\sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\right],$$

then summing this formula over t, we obtain

$$\begin{aligned}\nabla_\theta J(\theta) = \nabla_\theta E_{\tau\sim\pi_\theta}[R] &= E_{\tau\sim\pi_\theta}\left[\sum_{t'=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta\left(a_t|s_t\right)\right] \\ &= E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta\left(a_t|s_t\right) \sum_{t'=t}^{T-1} r_{t'}\right] \\ &= E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta\left(a_t|s_t\right)\right],\end{aligned}$$

where it drops the term r_T in the first equation, and $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a trajectory at time t. Then we have the following estimated update

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T-1} G_t^{(i)} \cdot \nabla_\theta \log \pi_\theta(a_t^i | s_t^i).$$

This update rule is utilized in a well-known Monte-Carlo policy gradient algorithm *REINFORCE*.

Baseline

For the update rule we mentioned before, the term $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a trajectory which might have high variance. Thus we can subtract a baseline $b(s)$ from the policy gradient to reduce the variance. The gradient with baseline follows that

$$\nabla_\theta E_{\tau \sim \pi_\theta}[R] = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T-1} (G_t - b(s_t)) \cdot \nabla_\theta \log \pi_\theta\left(a_t | s_t\right)\right].$$

A practical choice of the baseline is the expected return

$$b(s_t) = E[r_t + r_{t+1} + ... + r_{T-1}].$$

It can be shown that the baseline $b(s)$ can reduce the variance without changing the expectation, which means

$$E_{\tau \sim \pi_\theta}[\nabla_\theta \log \pi_\theta\left(a_t | s_t\right) b(s_t)] = 0,$$

$$Var_{\tau \sim \pi_\theta}[\nabla_\theta \log \pi_\theta\left(a_t | s_t\right) (G_t - b(s_t))] < Var_{\tau \sim \pi_\theta}[\nabla_\theta \log \pi_\theta\left(a_t | s_t\right) G_t].$$

9.3 Actor-Critic Policy Gradient

In Monte-Carlo Policy Gradient with *temporal causality*, G_t is a sample from Monte Carlo policy gradient, which is the unbiased but noisy estimate of $Q^{\pi_\theta}(s_t, a_t)$. Now instead we estimate the state-action value by value approximation function

$$Q_{\boldsymbol{w}}(s, a) \approx Q^{\pi_\theta}(s, a),$$

then the update becomes

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T-1} Q_{\boldsymbol{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta\left(a_t | s_t\right)\right]$$

which is called *Actor-Critic Policy Gradient*.

Actor-Critic Policy Gradient

- Actor
  - the policy function used to generate the action;
  - updates policy parameter θ, in direction suggested by critic;
- Critic

- the value function used to evaluate the reward of the actions;
- updates state-action value function parameter $\boldsymbol{w}$;

As we can see, the critic is solving a familiar problem *policy evaluation*. The actor is improving the policy.

Baseline

Similar to Monte-Carlo Policy Gradient, we can reduce the variance of Actor-Critic by a *baseline*. Notice that

$$Q^\pi(s,a) = E_\pi[r_1 + \gamma r_2 + ...|S_1 = s, A_1 = a],$$

$$V^\pi(s) = E_\pi[r_1 + \gamma r_2 + ...|S_1 = s],$$

therefore the state value function is actually the mean of the state-action value:

$$V^\pi(s) = E_{a\sim\pi}[Q^\pi(s,a)],$$

thus it can serve as a perfect baseline. Define *advantage function* as

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s,a),$$

then the policy gradient with a baseline is

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^\pi(s,a)].$$

9.4 Extension of Policy Gradient

Nowadays, State-of-the-art RL methods are almost all policy-based.

A2C, A3C: Asynchronous Methods for Deep Reinforcement Learning, ICML'16. Representative high-performance actor-critic algorithm.

TRPO: Trust region policy optimization: deep RL with natural policy gradient and adaptive step size.

PPO: Proximal policy optimization algorithms: deep RL with importance sampled policy gradient.

References

[1] Z. Shao, "Si252 reinforcement learning," https://piazza.com/class/k5ug5osvzhp3z8.

[2] B. Zhou, "Intro to reinforcement learning," https://github.com/zhoubolei/introRL.

[3] D. Silver, "Reinforcement learning," https://www.davidsilver.uk/teaching/.

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[5] J. K. Blitzstein and J. Hwang, *Introduction to Probability*. Chapman and Hall/CRC, 2014.

[6] C. Robert and G. Casella, *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[7] A. Slivkins *et al.*, "Introduction to multi-armed bandits," *Foundations and Trends® in Machine Learning*, vol. 12, no. 1-2, pp. 1–286, 2019.

[8] T. Lattimore and C. Szepesvári, "Bandit algorithms," *preprint*, p. 28, 2018.

[9] L. Weng, "The Multi-Armed Bandit Problem and Its Solutions," https://lilianweng.github.io/lil-log/2018/01/23/the-multi-armed-bandit-problem-and-its-solutions.html.