k-armed bandit problem

1. 函数可以用算法，公式，图等表示

2. 例人并不能完全了解自然规律，我们所学的只是基础，若我们把一切都学到，那我们将无路可走，也就无法变得更好？

3. Our objective is to maximize the expected total reward over some time period.

4. $q_*(a) = E(R_t | A_t = a)$

A_t is action, R_t reward.

$q_*(a)$ value of action a.

$Q_t(a)$ estimated value of action a.

5. estimate the values of actions

5.1 sample average method

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

5.2 $Q_{n+1} = \frac{1}{n}\sum^n R_i = \frac{1}{n}(R_n + (n-1)Q_n)$

$= Q_n + \frac{1}{n}(R_n - Q_n)$, $n \geq 1$

5.3

6. $Q_{n+1}(a) = \frac{\sum^n R_i \mathbb{1}_{A_i=a}}{\sum^n \mathbb{1}_{A_i=a}}$

$= Q_n(a) + \frac{1}{n(a)}(R_n - Q_n(a))$, if $A_n = a$.

⑤ $Q_{n+1}(a) = Q_n(a)$, if $A_n \neq a$

6. action selection rule

① greedy action selection:

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

② ε-greedy method

7. k-armed bandit problem

7.1 R = bandit (A)

7.2 ex.

k=10.

$q_*(a) \sim N(0,1)$, a=1,...,10 fixed,

then

$R_t(a) \sim N(q_*(a), 1)$

8. A simple bandit alg

① Initialize

Q(a) ← 0, N(a) ← 0,

for a = 1 to k

② Repeat forever

$A \leftarrow \begin{cases} \underset{a}{\operatorname{argmax}}\, Q(a) & \text{with pr} = 1-\varepsilon \\ \text{a random action,} & \text{with pr} = \varepsilon \end{cases}$

R ← bandit (A)

N ← N+1

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

9. exponential, recency-weighted average (weight average)

$Q_{n+1} \doteq Q_n + \alpha(R_n - Q_n)$

$\alpha \in (0,1]$ is constant

$\therefore Q_{n+1} = \alpha R_n + (1-\alpha)Q_n$

$= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$

10. estimate values of actions

① sample average method.

② weight average method.

11. upper confidence bound (UCB) action selection (see 6).

$A_t \doteq \underset{a}{\operatorname{argmax}} \left(Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}}\right)$,

$c > 0$ controls the degree of exploration.

12. gradient bandit alg

12.1 $H_t(a)$: preference of action a

12.2 $\Pr(A_t = a) \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$

12.3 $\pi_t(a)$ is the probability of taking action a at time t.

12.4

$H_{t+1}(a) \doteq H_t(a) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(a))$ if $A_t = a$

$H_{t+1}(a) \doteq H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a)$ if $A_t \neq a$

$\alpha > 0$ is stepsize, $\bar{R}_t \in \mathbb{R}$ is the average of all the rewards up through and including time t.

④ $\because \frac{\partial E(R_t)}{\partial H_t(a)}$

$= \sum_b \pi_t(b)(q_*(b) - X_t)\frac{\partial \pi_t(b)}{\partial H_t(a)} \cdot \frac{1}{\pi_t(b)}$

$= E_{A_t \sim \pi_t}\left[(q_*(A_t) - X_t)\frac{\partial \pi_t(A_t)}{\partial H_t(a)} \cdot \frac{1}{\pi_t(A_t)}\right]$

⑤ $\because E_{R_t \sim q_*(A_t)}(R_t | A_t) = q_*(A_t)$

$\therefore \frac{\partial E(R_t)}{\partial H_t(a)} = E_{A_t \sim \pi_t}\left[(R_t - X_t)\frac{\partial \pi_t(A_t)}{\partial H_t(a)} \cdot \frac{1}{\pi_t(A_t)}\right]$ (2)

⑥ $\because \frac{\partial \pi_t(b)}{\partial H_t(a)} = \pi_t(b)(1_{a=b} - \pi_t(a))$ (1)

choose $X_t = \bar{R}_t$

$\therefore \frac{\partial E(R_t)}{\partial H_t(a)} = E_{A_t \sim \pi_t}\left[(R_t - \bar{R}_t)(1_{a=A_t} - \pi_t(a))\right]$

13. gradient bandit alg as stochastic gradient ascent

13.1 $H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial E(R_t)}{\partial H_t(a)}$ (exact gradient ascent)

$E(R_t) \doteq \sum_b \pi_t(b) q_*(b)$

13.2 : the updates of gradient bandit alg (12.4) are equal to 13.1 in expected value.

$\therefore$ gradient bandit alg is an instance of stochastic gradient ascent.

13.3 proof:

① $\frac{\partial E(R_t)}{\partial H_t(a)} = \frac{\partial}{\partial H_t(a)} \sum_b \pi_t(b) q_*(b)$

$= \sum_b q_*(b) \frac{\partial \pi_t(b)}{\partial H_t(b)}$

② $\because \sum_b \pi_t(b) = 1 \quad \therefore \sum_b \frac{\partial \pi_t(b)}{\partial H_t(a)} = 0$

③ $\therefore \frac{\partial E(R_t)}{\partial H_t(a)} = \sum_b (q_*(b) - X_t)\frac{\partial \pi_t(b)}{\partial H_t(a)}$

X_t does not depend on b.

6.

6.1 one step dynamics of the environment

$\Leftrightarrow p(s',r|s,a)$

$\doteq Pr\{S_{t+1}=s', R_{t+1}=r | S_t=s, A_t=a\}$

6.2 ~~the~~ 6.1 ~~contain~~ completely specify the dynamics of a finite MDP.

6.3. Given 6.1, one can compute anything else one might want to know about the environment. Markov property.

7.

7.1 expected rewards for state-action pairs.

$r(s,a) \doteq E(R_{t+1}|S_t=s, A_t=a)$

$= \sum_{r\in R} r \sum_{s'\in S} p(s',r|s,a)$

7.2 state-transition probability

$p(s'|s,a) \doteq Pr(S_{t+1}=s'|S_t=s, A_t=a)$

$= \sum_{r\in R} p(s',r|s,a)$

4.2 finite MDP

$\Leftrightarrow$ MDP with finite state and action sets.

5. Markov property.

$Pr\{S_{t+1}=s', R_{t+1}=r | S_0, A_0, R_1, \cdots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\}$

$= Pr\{S_{t+1}=s', R_{t+1}=r | S_t=s, A_t=a\}$

$\forall r, s, S_0, A_0, R_1, \cdots, S_{t-1}, A_{t-1}, R_t, S_t, A_t$

$\Leftrightarrow$ the ~~sig~~ state signal has the Markov property.

$\Leftrightarrow$ the environment and task as a whole have the Markov property.

5.2 An environment satisfies the Markov property if its state signal compactly summarizes the past without degrading the ability to predict the future. This is rarely exactly true, but often nearly so.

3. Markov property:

All that matters is in the current state signal.

P(future | now, history)

= P(future | now).

3.2 like linear, it is a simple property. A fully understanding of the theory of Markov case is an essential foundation for extending it to the more complex and realistic non-Markov case.

4. A reinforcement learning task that satisfies the Markov property is called a Markov decision process, or MDP. Finite MDP are all we need to understand 90% of modern rl.

finite MDP

1. rl: maximize cumulative reward in the long run. It is what we want to achieve, not how we want to achieve.

2. ① return: sum of rewards.

$G_t \doteq R_{t+1} + \cdots + R_T$

② $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}$

$+ \cdots$

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

③ unified notation for episodic and continuing tasks

$G_t \doteq \sum_{k=0}^{T-t+1} \gamma^k R_{t+k+1}$

including the possibility that $T=\infty$ or $\gamma=1$ (but not both)

$$= \mathop{E}_{A_t \sim P(A_t|S_t=s)} \mathop{E}_{(S_{t+1},R_{t+1}) \sim P(S_{t+1},R_{t+1}|S_t=s,A_t)}$$

$$\mathop{E}_{\pi}(G_t \mid S_t=s, A_t, S_{t+1}, R_{t+1})$$

~~$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)($~~

$$= \mathop{E}_{A_t \sim P(A_t|S_t=s)} \mathop{E}_{(S_{t+1},R_{t+1}) \sim P(S_{t+1},R_{t+1}|S_t=s,A_t)} (R_{t+1} + \gamma \mathop{E}_{\pi}(G_{t+1}|S_{t+1}))$$

~~$E(R_{t+1} + \gamma E_\pi(G_{t+1}|S_{t+1}))$~~

$$= \sum_{a,s',r} \pi(a|s)\, p(s',r|s,a)\,(r + \gamma v_\pi(s'))$$

13.6 the value function v_π is the unique solution to its Bellman equation.

14. ① discrete-time optimization problem: Bellman equation.

② continuous-time optimization problem: Hamilton-Jacobi-Bellman equation. (a partial differential equation).

13.3 $G_t = R_{t+1} + \gamma G_{t+1}$

13.4

① $\mathop{E}_{X \sim P(X)} f(X)$

$= \mathop{E}_{(X,Y) \sim P(X,Y)} f(X)$

proof: $\int_x f(x) dx\, p(x)\, dx$

$= \int_x \left(\int_y p(x,y)\, dy \right) f(x)\, dx$

$= \int_x \int_y p(x,y) f(x)\, dx dy$

$= \mathop{E}_{(X,Y) \sim P(X,Y)} f(X)$

② $\mathop{E}_{X \sim P(X)} f(X)$

$= \mathop{E}_{X \sim P(X)} \mathop{E}_{Y \sim P(Y|X)} f(X)$

13.5 proof of 13.1

$v_\pi(s) = \mathop{E}_{\pi}(G_t \mid S_t=s)$

$= \mathop{E}_{A_t \sim P(A_t|S_t=s)} \mathop{E}_{\pi}(G_t \mid S_t=s, A_t)$

11. (agent, environment, action) $\Leftrightarrow$ (controller, controlled system, control signal).

12. value functions v_π and q_π can be estimated from experience.

13.

13.1 Bellman equation for v_π.

$$v_\pi(s) = \sum_{a,s',r} \pi(a|s)\, p(s',r|s,a)\,(r + v_\pi(s'))$$

13.2 $\mathop{E}_{(X,Y) \sim P(X,Y)} f(X,Y)$

$= \int_x \int_y p(x,y) f(x,y)\, dx dy$

$= \int_x p(x) \int_y p(y|x) f(x,y)\, dy\, dx$

$= \mathop{E}_{X \sim P(X)} \mathop{E}_{Y \sim P(Y|X)} f(X,Y)$

Similarly,

$\mathop{E}_{(X,Y,Z) \sim P(X,Y,Z)} f(X,Y,Z)$

$= \mathop{E}_{X \sim P(X)} \mathop{E}_{Y \sim P(Y|X)} \mathop{E}_{Z \sim P(Z|X,Y)} f(X,Y,Z)$

13 expected rewards for state-action-next-state triples,

$r(s,a,s')$

$\doteq E(R_{t+1} \mid S_t=a, A_t=a, S_{t+1}=s')$

$= \dfrac{\sum_{r \in R} r\, p(s',r|s,a)}{p(s'|s,a)}$

8. policy is a mapping π.

$s \in S$, $a \in A(s)$, $\pi(a|s)$.

9. state-value function for policy π.

$v_\pi(s) \doteq \mathop{E}_{\pi}(G_t \mid S_t=s)$

$= \mathop{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t=s \right)$

10. action-value function for policy π.

$q_\pi(s,a) \doteq \mathop{E}_{\pi}(G_t \mid S_t=s, A_t=a)$

21 The online nature of rl makes it possible to approximate optimal policies in ways that put more efforts into learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states. This is one key property that distinguishes rl from other approaches to approximately solving MDPs.

22. Bellman optimality equations
⇒ optimal value functions
⇒ an optimal policy.

23 The rl problem is deeply indebted to the idea of MDPs from the field of optimal control. RL adds to MDPs a focus on approximation and incomplete info for realistically large problems.

18 Bellman optimality equation for q_*

$$q_*(s,a) = \mathbb{E}(R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t=s, A_t=a)$$
$$= \sum_{s',r} p(s',r|s,a)(r + \gamma \max_{a'} q_*(s',a'))$$

(↳)

19. If you have the optimal value function v_*, then the actions that appear best after a one-step search will be optimal actions

20. With q_*, the optimal action is the one that maximizes $q_*(s,a)$. The optimal action-value function allows optimal actions to be selected without having to know anything about the environment's dynamics.

$q_*(s,a)$

$$= \mathbb{E}(R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t=s, A_t=a)$$

17. Bellman optimality equation for v_*.

$$v_*(s) = \max_{a \in A(s)} q_{\pi_*}(s,a)$$
$$= \max_a \mathbb{E}(G_t \mid S_t=s, A_t=a)$$
$$= \max_a \mathbb{E}(R_{t+1} + \gamma G_{t+1} \mid S_t=s, A_t=a)$$
$$= \max_a \mathbb{E}(R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t=s, A_t=a)$$
$$= \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_*(s'))$$

(↳, see 13.5)

17.2 for finite MDPs, the Bellman optimality equation

$$v_*(s) = \max_{a \in A(s)} \sum_{s',r} p(s',r|s,a)(r + \gamma v_*(s'))$$

has a unique solution independent of the policy.

15. solving a reinforcement learning task means, roughly, finding a policy that achieves a lot of reward over the long run.

16. $\pi \geq \pi' \Leftrightarrow v_\pi(s) \geq v_{\pi'}(s)$
$\forall s \in S$.

16.2 optimal policy π_* exists. (≥) It may not be unique.

16.3 optimal state-value function v_*.

$$v_*(s) \doteq \max_\pi v_\pi(s), \quad \forall s \in S$$

16.4 optimal action-value function q_*.

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a)$$
$\forall s \in S, \forall a \in A(s)$.

16.5 (↳ see 18)

Moreover, if there is strict inequality of (1) at any state, then there must be strict inequality of (2) at at least one state.

(there is at any state $\Leftrightarrow$ $\exists$ at some state).

proof: $v_\pi(s) \le q_\pi(s, \pi'(s))$

$= \mathbb{E}_{\pi'}(R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s)$

$\le \mathbb{E}_{\pi'}(R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s)$

$= \mathbb{E}_{\pi'}(R_{t+1} + \gamma \mathbb{E}_{\pi'}(R_{t+2} + \gamma v_\pi(S_{t+2})) \mid S_t = s)$

$= \mathbb{E}_{\pi'}(R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s)$

$\le \cdots$

$\le \mathbb{E}_{\pi'}(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s)$

$= v_{\pi'}(s)$

13. policy improvement: the process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy.

9. update the values in place $\Leftrightarrow$ with each new backup value immediately overwriting the old one.

10. for DP algs, we usually use the in-place version.

11. stopping criterion of iterative policy evaluation: stop when $\max_{s \in S} |v_{k+1}(s) - v_k(s)|$ is sufficiently small.

12. policy improvement theorem:

π, π' deterministic policies,

$q_\pi(s, \pi'(s)) \ge v_\pi(s), \forall s \in S$ (1)

$\Rightarrow \pi'$ must be as good as, or better than π. That is, $v_{\pi'}(s) \ge v_\pi(s), \forall s \in S$. (2)

6. if $p(s', r \mid s, a)$ is known then equation in 5 is a system of $|S|$ linear equations in $|S|$ unknowns.

7. iterative policy evaluation:

update rule:

$$v_{k+1} \doteq \mathbb{E}_\pi(R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s)$$

if condition A satisfied

$\Rightarrow v_k \to v_\pi$, as $k \to \infty$

the initial v_0 is chosen arbitrarily.

8. Each iteration of iterative policy evaluation backs up the value of every state once to produce v_{k+1}, so we call it a full backup.

All the backups in DP algs are full backups because they are based on all possible next states rather than on a sample next state.

DP

1. dynamic programming (DP) refers to a collection of algs that can be used to compute optimal policies given a perfect model of the environment as a MDP.

2. we usually assume the environment is a finite MDP.

3. $v_\pi(s) \doteq \mathbb{E}_\pi(R_{t+1} + \gamma R_{t+2} + \cdots \mid S_t = s)$

$= \mathbb{E}_\pi(R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s)$

$= \sum_a \pi(a|s) \sum_{s', r} p(s', r \mid s, a)(r + \gamma v_\pi(s'))$

4. condition A:

$\gamma < 1$ or eventual termination is guaranteed from all states under the policy π.

5. for $v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r \mid s, a)(r + \gamma v_\pi(s'))$, condition A satisfied $\Rightarrow v_\pi$ $\exists$, unique.

19.5 condition A satisfied ⇒ value iteration converges.

20. asynchronous DP algs are in-place iterative DP algs that back up states in an arbitrary order.

21. GPI: generalized policy iteration. We use GPI to refer to the general idea of letting policy evaluation and policy improvement processes interact, independent of the granularity and other details of the two processes.

21.2 Almost all rl methods are well described as GPI.

22. DP is exponentially faster than any direct search in policy space could be.

23 bootstrapping: update estimates on the basis of other estimates.

$\xrightarrow{E}$ denotes policy evaluation,

$\xrightarrow{I}$ denotes policy improvement.

18 ∵ a finite MDP has a finite number of policies, ∴ policy iteration must converge to an optimal policy and optimal value function in a finite number of iterations.

19. value iteration:

$$v_{k+1}(s) \doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a)$$

19.2 policy evaluation:

$$v_{k+1} \doteq \mathop{E}_{\pi}(R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s)$$

19.3 value iteration is identical to policy evaluation except that it requires the maximum to be taken over all actions.

19.4 value iteration is obtained simply by turning the Bellman optimality equation into an update rule.

14. greedy policy $\pi'(s) \doteq \operatorname{argmax}_a q_\pi(s,a)$ $\forall s \in S$

15. Policy improvement must give us a strictly better policy except when the original policy is already optimal.

proof: ∵ $\pi'(s) = \operatorname{argmax}_a q_\pi(s,a)$

$$= \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_\pi(s'))$$

∴ $v_\pi(s) = \max_a q_\pi(s,a)$

$$= \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_\pi(s'))$$

If $v_\pi = v_{\pi'}$

then $v_{\pi'}(s) = \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_{\pi'}(s'))$

∴ this is the same as the Bellman optimality equation

∴ $v_{\pi'}$ must be v_*

∴ π' and π must be optimal policies.

16. policy improvement theorem holds for the stochastic policies.

17 policy iteration: $\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{} \cdots \xrightarrow{} \pi_* \xrightarrow{E} v_*$

10.1 Monte Carlo version of classical policy iteration: $\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \to \cdots \xrightarrow{I} \pi_* \xrightarrow{E} q_{\pi_*}$

10.2 $\because q_{\pi_k}(s, \pi_{k+1}(s)) = q_{\pi_k}(s, \operatorname{argmax}_a q_{\pi_k}(s,a))$

$= \max_a q_{\pi_k}(s,a) \geq q_{\pi_k}(s, \pi_k(s)) = v_{\pi_k}(s)$, $\forall s \in S$.

$\therefore$ policy improvement theorem assures us that

$\pi_{k+1}(s) \doteq \operatorname{argmax}_a q_{\pi_k}(s,a)$ is uniformly better than π_k, or just as good as π_k.

$\therefore$ policy iteration converges.

10.3 We made two unlikely assumptions in order to obtain this guarantee of convergence for the Monte Carlo methods

① the episodes have exploring starts.

② policy evaluation can be done with an infinite number of episodes. (To avoid the infinite number of episodes, we can use the idea of GPI).

11. Monte Carlo ES (Monte Carlo with exploring starts) use GPI on an episode-by-episode basis. Monte Carlo ES converges, but has not yet been formally proved.

7. advantages Monte Carlo methods can have over DP methods:

① ability to learn from actual experience.

② ability to learn from simulated experience.

③ $\because$ estimating the value of a single state is independent of the number of states.

$\therefore$ attractive when one requires the value of only one or a subset of states.

8. Both every-visit MC and first-visit MC methods converge quadratically to $q_\pi(s,a)$ as the number of visits to each state-action pair approaches infinity.

9. assure continual exploration:

① exploring starts.

② consider only policies that are stochastic with a nonzero probability of selecting all actions in each state.

10. We use MC estimation in control, that is, to ~~estimate~~ approximate optimal policies

Monte Carlo Methods

1. Monte Carlo methods are ways of solving the rl problem based on averaging sample returns

2. Monte Carlo methods are our first learning methods for estimating value functions and discovering optimal policies.

3. rule $\rightleftarrows$ data

If we know rule, data will be useless.

4. We compute value functions from knowledge of MDP, or we learn them from sample returns with the MDP.

5. policy evaluation:

first-visit MC, every-visit MC

Both converge quadratically to $v_\pi(s)$ as the number of visits (or first visits) to s goes to infinity. (not proved)

6. $\because$ the estimates for each state are independent $\therefore$ Monte Carlo methods do not bootstrap.

16. ~~coverage~~ coverage: $\pi(a|s)>0 \Rightarrow \mu(a|s)>0$

where π is target policy, μ is behavior policy.

17.

17.2 $\mathop{\mathbb{E}}_{\frac{p}{q}} f = \int pf = \int q\frac{p}{q} f = \mathop{\mathbb{E}}_{q}\left[\frac{p}{q} f\right]$

17.3 $\mathop{\mathbb{E}}_{\pi} G_t = \mathop{\mathbb{E}}_{\mu} \rho_t^T G_t$, $\rho_t^T \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k) p(S_{k+1}|S_k,A_k)}{\prod_{k=t}^{T-1} \mu(A_k|S_k) p(S_{k+1}|S_k,A_k)}$

17.4 ordinary importance sampling

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_t^{T(t)} G_t}{|\mathcal{T}(s)|}$$

17.5 weighted importance sampling

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_t^{T(t)} G_t}{\sum_{t\in\mathcal{T}(s)} \rho_t^{T(t)}}$$

18. return-specific importance sampling

18.2 flat partial returns: $\bar{G}_t^h \doteq R_{t+1} + R_{t+2} + \cdots + R_h$

$0 \le t < h \le T$

18.3 $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

$= (1-\gamma) \sum_{h=t+1}^{T-1} \gamma^{h-t-1} \bar{G}_t^h + \gamma^{T-t-1} \bar{G}_t^T$

18.4 discounting-aware importance sampling estimator:

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \left((1-\gamma) \sum_{h=t+1}^{T(t)-1} \gamma^{h-t-1} \rho_t^h \bar{G}_t^h + \gamma^{T(t)-t-1} \rho_t^{T(t)-1} \bar{G}_t^{T(t)}\right)}{|\mathcal{T}(s)|}$$

18.5 per-reward importance sampling

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \tilde{G}_t}{|\mathcal{T}(s)|}, \quad \tilde{G}_t = \rho_t^{t+1} R_{t+1} + \gamma \rho_t^{t+2} R_{t+2} + \cdots + \gamma^{T-t-1} \rho_t^T R_T$$

theorem, $\pi' \ge \pi$.

14.3 define new environment

: $\begin{cases} =\text{old}, & \text{with } pr = 1-\varepsilon. \\ \text{repicks action at random with equal pr,} & \\ \text{and then behave like old environment,} & \text{with } pr = \varepsilon. \end{cases}$

if $q_\pi(s, \pi'(s)) = v_\pi(s)$, $\forall s$, i.e. $\pi' = \pi$,

(π is no longer improved)

$$\Rightarrow v_\pi(s) = q_\pi(s, \pi'(s))$$

$$= (1-\varepsilon) \max_a q_\pi(s,a) + \frac{\varepsilon}{|A(s)|} \sum_a q_\pi(s,a)$$

$$= (1-\varepsilon) \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_\pi(s')) + \frac{\varepsilon}{|A(s)|} \sum_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_\pi(s'))$$

$$= (1-\varepsilon) \max_a \tilde{q}_*(s,a) + \frac{\varepsilon}{|A(s)|} \sum_a \tilde{q}_*(s,a)$$

$$= \tilde{v}_*(s)$$

$\because \tilde{v}_*$ is optimal value functions for the new environment, and $\tilde{v}_*$ unique

$\therefore v_\pi = \tilde{v}_*$

15. off-policy learning: learning is from data "off" the target policy.

12. On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data.

13. Monte Carlo ES is an on-policy method with exploring starts.

14. policy iteration works for ε-soft policies.

14.2 $\pi' \ge \pi$.

proof: $q_\pi(s, \pi'(s)) = \sum_a \pi'(a|s) q_\pi(s,a)$

$$= \frac{\varepsilon}{|A(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \max_a q_\pi(s,a)$$

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a)$$

$$= \frac{\varepsilon}{|A(s)|} \sum_a q_\pi(s,a) + \sum_a \left(\pi(a|s) - \frac{\varepsilon}{|A(s)|}\right) q_\pi(s,a)$$

$\because \sum_a \left(\pi(a|s) - \frac{\varepsilon}{|A(s)|}\right) q_\pi(s,a)$

$\le \sum_a \left(\pi(a|s) - \frac{\varepsilon}{|A(s)|}\right) \max_{a'} q_\pi(s,a')$

$= \max_a (1-\varepsilon) q_\pi(s,a)$

$\therefore v_\pi(s) \le q_\pi(s, \pi'(s))$ $\therefore$ by policy improvement

19. incremental implementation

19.2. weighted importance sampling

$$C_{n+1} \doteq C_n + W_{n+1}$$

$$W_n \doteq \rho_t^{T(t)}$$

$$V_{n+1} \doteq \frac{\sum_{k=1}^{n} W_k G_k}{\sum_{k=1}^{n} W_k} \quad \text{#}$$

$$\therefore V_{n+1} C_n = \sum_{k=1}^{n} W_k G_k$$

$$V_n C_{n-1} = \sum_{k=1}^{n-1} W_k G_k$$

$$\therefore V_{n+1} C_n = V_n C_{n-1} + W_n G_n$$

$$\therefore V_{n+1} C_n = V_n C_n + W_n (G_n - V_n)$$

$$\therefore V_{n+1} = V_n + \frac{W_n}{C_n} (G_n - V_n), \quad n \geq 1$$

Temporal-Difference Learning

1. TD learning is a combination of Monte Carlo ideas and dynamic programming ideas.
2. constant-α MC: $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$, α is constant.
3. TD(0): $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
4. $v_\pi(s) \doteq E_\pi(G_t | S_t = s)$ (1)

$$= E_\pi\left(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right) = E_\pi\left(R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right)$$

$$= E_\pi(R_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s) \quad (2)$$

Roughly speaking, MC methods use an estimate of (1) as a target, whereas DP methods use an estimate of (2) as a target.

MC: a sample return is used in place of the real expected return.

DP: $v_\pi(S_{t+1})$ is not known and $V(S_{t+1})$ is used instead.

TD: combine the sampling of MC with the bootstrapping of DP.

5. TD, MC: sample backups.

   DP: full backups.

6. TD error: $\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

6.2 MC error can be written as a sum of TD errors.

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) \\ &= R_{t+1} + \gamma V(S_{t+1}) - V(S_t) + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \cdots \\ &= \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(G_T - V(S_T)) \\ &= \sum_{k=0}^{T-t-1} \gamma^k \delta_{t+k} \end{aligned}$$

7. advantages of TD prediction methods.

① TD have an advantage over DP in that they do not require a model of the environment.

② TD over MC: they are naturally implemented in an online, fully incremental ~~fation~~ fashion.

③ for any fixed policy π, TD has been proved to converge to v_π under mild conditions.

8. under batch updating, TD(0) converges deterministically to a single answer independent of the step-size parameter, α, as long as α is chosen to be sufficiently small. The constant-α MC method also converges ~~deter~~ deterministically under the same condition, but to a different answer.

8.2 Under normal updating, the methods do not move all the way to their respective batch answers, but in some sense they take steps in these directions.

8.3 ~~How is it that batch TD~~ MC is optimal only in a limited way, and that TD is optimal in a way that is more relevant to predicting returns.

8.4 Batch MC always find the estimates that minimize mse error on the training set, whereas batch TD(0) always finds the estimates that would be exactly correct for the maximum-likelihood model of the Markov process.

8.5 batch TD(0) converges to the certainty-equivalence estimate.

8.6 certainty-equivalence estimate is the estimate of the value function that would be exactly correct if the model were exactly correct. It ~~is assumes~~ that the estimate of the underlying process was known with certainty rather than being approximated. is equivalent to assuming

9. Sarsa is an on-policy TD control algorithm.

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

The rule uses $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ that make up a transition from one state-action pair to the next. This quintuple gives rise to the name Sarsa for the algorithm.

10. One of the early breakthroughs in rl was the development of an off-policy TD control algorithm known as Q-learning (1989), defined by $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$

10.2. Q directly approximates q_*, independent of the policy being followed.

11. Any method guaranteed to find optimal behavior in the general case must has a minimal requirement that all ~~pairs~~ ~~action~~ state-action pairs continue to be updated.

12. Expected Sarsa:

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

Given S_{t+1}, this algorithm moves deterministically in the same direction as Sarsa moves in expectation, and accordingly it is called expected Sarsa.

12.2 Expected Sarsa can be an on-policy or off-policy algorithm. Except for the same additional computational cost, Expected Sarsa may completely dominate both Sarsa and Q-learning.

13. A maximum over estimated values is used as an estimate of the maximum value, which can lead to a significant positive bias. We call this maximization bias. (ii)

14. doubled learning : $A^* = \arg\max_a Q_1(a)$, $Q_2(A^*)$ is unbiased in the sense that $E(Q_2(A^*)) = q(A^*)$. Similarly, we can yield a second unbiased estimate $Q_1(\arg\max_a Q_2(a))$

here Q_1, Q_2 are independent esimate of the true value $q(a)$, $\forall a \in A$.

14.2 doubled learning seems to eliminate the harm caused by maximization bias.

15. TD methods are alternatives to MC methods for solving the prediction problem. In both cases, the extension to the control problem is via the idea of generalized policy iteration (GPI) that we abstracted from dynamic programming. This is the idea that approximate policy and value functions should interact in such a way that they both move toward their optimal values.

16. TD methods are general methods for learning to make long-term predictions about dynamical systems.

Chapter 7
Multi-step ~~Bootstrapping~~

1. Multi-step TD methods generalize both MC methods and one-step TD methods so that one can switch from one to the other smoothly.

2. n-step TD methods are still TD methods because they still change an earlier estimate based on how it differs from a later estimate.

3.1 The target of Monte Carlo backups :

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T.$$

3.2 the target of one-step backups:

$$G_t^{(1)} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

3.3 the target of n-step backup:

$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \quad n \geq 1, \ 0 \leq t < T-n$$

$$G_t^{(n)} \doteq G_t, \text{ if } t+n \geq T.$$

4. n-step TD algorithm :

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\left(G_t^{(n)} - V_{t+n-1}(S_t)\right), \quad 0 \leq t < T$$

$$V_{t+n}(s) = V_{t+n-1}(s), \ \forall s \neq S_t.$$

5. error reduction property of n-step returns ~~(n-step Sarsa)~~

$$\max_s \left| \mathop{E}_{\pi}\left(G_t^{(n)} \middle| S_t = s\right) - v_\pi(s) \right| \leq \gamma^n \max_s \left| V_{t+n}(s) - v_\pi(s) \right|$$

6.1 n-step returns in terms of estimated action values:

~~n-step Sarsa~~

$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \ n \geq 1, \ 0 \leq t < T-n$$

$$G_t^{(n)} \doteq G_t \text{ if } t+n \geq T.$$

~~6.2~~ 6.2 n-step Sarsa algorithm :

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha\left(G_t^{(n)} - Q_{t+n-1}(S_t, A_t)\right), \quad 0 \leq t < T$$

$$Q_{t+n}(s, a) = Q_{t+n-1}(s, a), \ \forall s, a \text{ such that } s \neq S_t \text{ or } a \neq A_t.$$

6.3 $G_t^{(n)} = $ ~~$G_{t:h}$~~ $Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n-1, T-1)} \gamma^{k-t}\left(R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k)\right)$

7. the target of n-step Expected Sarsa.

$$G_t^{(n)} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a|S_{t+n}) Q_{t+n-1}(S_{t+n}, a), \quad n \geq 1,$$

8 ~~Offline~~ off-policy version of n-step TD; $0 \leq t \leq T-n$.

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_t^{t+n} \left(G_t^{(n)} - V_{t+n-1}(S_t) \right), \quad 0 \leq t < T$$

where ρ_t^{t+n} is called importance sampling ratio.

$$\rho_t^{t+n} \doteq \prod_{k=t}^{\min(t+n-1, T-1)} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}$$

9. off-policy form of n-step Sarsa:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1}^{t+n+1} \left(G_t^{(n)} - Q_{t+n-1}(S_t, A_t) \right), \quad 0 \leq t < T$$

10. off-policy form of n-step Expected Sarsa.

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1}^{t+n} \left(G_t^{(n)} - Q_{t+n-1}(S_t, A_t) \right), \quad 0 \leq t < T.$$

11. It is probably inevitable that off-policy training is slower than on-policy training — after all, the data is less relevant to what you are trying to learn.

12. n-step tree backup algorithm.

12.1 off-policy learning without importance sampling.

12.2 expected action value:

$$V_t \doteq \sum_a \pi(a|S_t) Q_{t-1}(S_t, a)$$

12.3 TD error: $\delta_t \doteq R_{t+1} + \gamma V_{t+1} - Q_{t-1}(S_t, A_t)$

12.4 $G_t^{(1)} \doteq R_{t+1} + \gamma V_{t+1} = Q_{t-1}(S_t, A_t) + \delta_t$

$$G_t^{(2)} \doteq R_{t+1} + \gamma V_{t+1} \doteq \gamma \pi(A_{t+1}|S_{t+1}) Q_t(S_{t+1}, A_{t+1}) + \gamma \pi(A_{t+1}|S_{t+1})(R_{t+2} + \gamma V_{t+2})$$

$$= Q_{t-1}(S_t, A_t) + \delta_t + \gamma \pi(A_{t+1}|S_{t+1}) \delta_{t+1}$$

$$\therefore G_t^{(n)} \doteq Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n-1, T-1)} \delta_k \prod_{i=t+1}^{k} \gamma \pi(A_i|S_i)$$

12.5 $Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha(G_t^{(n)} - Q_{t+n-1}(S_t, A_t)), \quad 0 \le t < T$

$Q_{t+n}(s, a) = Q_{t+n-1}(s, a), \quad \forall s, a$ such that $s \neq S_t$ or $a \neq A_t$.

13. A unifying algorithm: n-step Q(σ)

13.2 n-step returns of Q(σ)

① $G_t^{(1)} \doteq R_{t+1} + \gamma(\sigma_{t+1} Q_t(S_{t+1}, A_{t+1}) + (1-\sigma_{t+1}) V_{t+1})$

$= \delta_t + Q_{t-1}(S_t, A_t)$

where $\delta_t \doteq R_{t+1} + \gamma(\sigma_{t+1} Q_t(S_{t+1}, A_{t+1}) + (1-\sigma_{t+1}) V_{t+1}) - Q_{t-1}(S_t, A_t)$

② $G_t^{(2)} \doteq R_{t+1} + \gamma(\sigma_{t+1} Q_t(S_{t+1}, A_{t+1}) + (1-\sigma_{t+1}) V_{t+1})$

$- \gamma(1-\sigma_{t+1}) \pi(A_{t+1}|S_{t+1}) Q_t(S_{t+1}, A_{t+1})$

$+ \gamma(1-\sigma_{t+1}) \pi(A_{t+1}|S_{t+1}) (R_{t+2} + \gamma(\sigma_{t+2} Q_t(S_{t+2}, A_{t+2}) + (1-\sigma_{t+2}) V_{t+2}))$

$- \gamma \sigma_{t+1} Q_t(S_{t+1}, A_{t+1})$

$+ \gamma \sigma_{t+1} (R_{t+2} + \gamma(\sigma_{t+2} Q_t(S_{t+2}, A_{t+2}) + (1-\sigma_{t+2}) V_{t+2}))$

$= Q_{t-1}(S_t, A_t) + \delta_t + \gamma((1-\sigma_{t+1}) \pi(A_{t+1}|S_{t+1}) + \sigma_{t+1}) \delta_{t+1}$

③ $\therefore G_t^{(n)} \doteq Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n-1, T-1)} \delta_k \prod_{i=t+1}^{k} \gamma((1-\sigma_i) \pi(A_i|S_i) + \sigma_i)$

13.3 for off-policy:

$\rho_t^{t+n} \doteq \prod_{k=t}^{\min(t+n-1, T-1)} \left(\sigma_k \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} + 1 - \sigma_k \right)$

13.4

4-step TD

4ρ

4-step Sarsa

4ρ

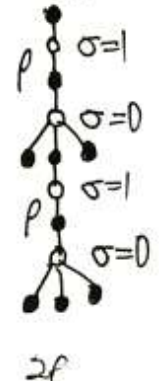
4-step Tree backup

0ρ

4-step Expected Sarsa

3ρ

4-step Q(σ)



2ρ

Chapter 8

Planning and learning with tabular methods

1. MC and TD can be unified by n-step methods. ~~Similarly~~ We can develop a unified view of planning and learning methods.

2. By a model of the environment we mean anything that an agent can use to predict how the environment will respond to its actions.

2.2 distribution model

2.3 sample model.

2.4 model can be used to simulate the environment and produce simulated experience.

3. We ~~refer~~ use planning to refer to any computational process that takes a model as input and produces or improves a policy for interacting with the modeled environment.

3.2 state-space planning

3.3 plan-space planning : evolutionary methods and "partial-order planning".

3.4 plan-space planning methods are difficult to apply efficiently to the stochastic optimal control problems that are the focus in rl.

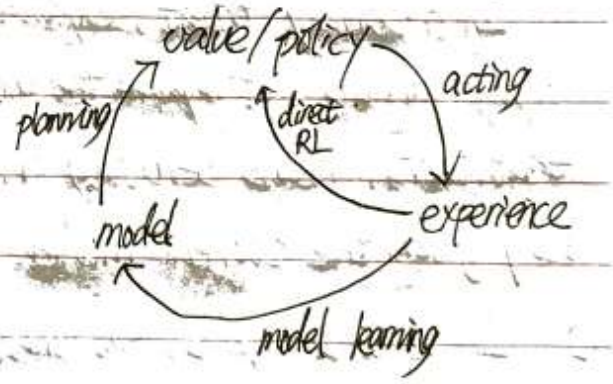
4. all state-space planning methods share a common structure :

model → simulated experience —backups→ values → policy

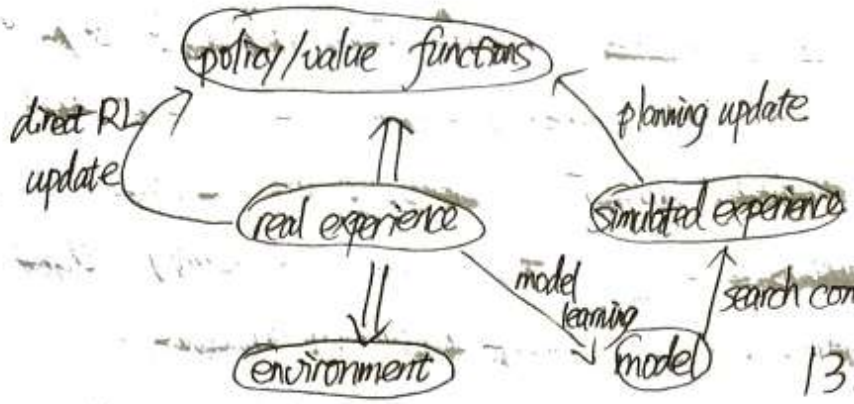
5. The heart of both learning and planning methods is the estimation of value functions by backup operations.

6. random-sample one-step tabular Q-planning is a planning method based on ~~one-step~~ tabular Q-learning and on random samples from a sample model.

7. relationship between experience, model, values, and policy.



8. Dyna architecture 在虚拟环境，然后让环境指导RL.



9. Dyna-Q is a simple architecture of Dyna.

9.2 Dyna-Q includes planning, acting, model-learning, and direct RL — all ~~occuring~~ occurring continually. The planning method is the random-sample one-step tabular Q-planning method. The direct RL method is one-step tabular Q-learning. The model-learning method is also table-based and assumes the world is deterministic.

10. Dyna-Q+ is Dyna-Q with an exploration bonus that encourages exploration.

11. In a planning context, exploration means trying actions that improve the model, whereas exploitation means behaving in the optimal way given the current model.

12. In Dyna-Q agents, simulated transitions are started in state-action pairs selected uniformly at random from all previously experienced pairs. But a uniform selection is usually not the best.

13. (집중) backward focusing of planning computations: work backward from arbitrary states that have changed in value, either performing useful backups or terminating the propagation.

14. (선택) prioritized sweeping: prioritize the backups according to a measure of their urgency, and perform them in order of priority.

15. prioritized sweeping for a deterministic environment. (선택과 집중) (ㄴ)

16. prioritized sweeping uses sample backups in deterministic environments, but uses full backups in stochastic environments.

16.2 One of prioritized sweeping's limitations is that it uses full backups, which in stochastic environments may waste lots of computation on low-probability transitions.

17. all kinds of state-space planning can be viewed as sequences of backups, varying only in the type of backup, full or sample, large or small, and in the order in which the backups are done.

18. forward focusing: focus on states according to how easily they can be reached from the current state. the states that are visited frequently under the current policy.

19. there tends to be two ways of thinking about planning. One way conceives of planning as the gradual improvement of a policy or value function that is good in all states generally rather than focused on any particular state. The other way thinks

about planning as part of action selection. (L)

20. The classical state-space planning methods in AI are planning-as-part-of-action-selection methods collectively known as heuristic search.

20.2 heuristic search can be viewed as an extension of the idea of a greedy policy beyond a single step. (L)

20.3 The great focusing of memory and computational resources on the current decision is presumably the reason why heuristic search can be so effective.

21 Monte Carlo Tree Search (MCTS)

21.2 MCTS is one of the simplest examples of planning as part of the policy. It is

most often used when the model of the world is completely known and cheap to compute.

21.3 MCTS typically involves no approximate value functions or policies that are retained from one time step to the next; these are computed on each step and then discarded.

21.4 MCTS incrementally builds a partial game tree to select each of a game-playing program's moves.

21.5 each iteration of MCTS proceeds in four stages.

① selection ② expansion ③ simulation ④ backpropagation

22 It is straightforward to integrate learning and planning processes simply by allowing both to update the same estimated value function.

23 The smaller the backups, the more incremental the planning methods can be. Among the smallest backups are one-step sample backups, as in Dyna.

24 backward focusing : prioritized sweeping

forward focusing : when planning is done exclusively as part of action selection, as in classical heuristic search and MCTS.

Chapter 9

On-policy Prediction with Approximation

1. The kind of generalization we require is often called function approximation because it takes examples from a desired function and attempts to generalize from them to construct an approximation of the entire function.

1.2 function approximation is an instance of supervised learning.

2. we write $\hat{v}(s,\theta) \approx v_\pi(s)$ for the approximated value of state s given weight vector θ.

3. All of the prediction methods covered in this book have been described as backups, that is, as updates to an estimated value function that shifts its value at particular states toward a "backed-up value" function for that state.

3.2 we refer to an individual backup by the notation $s \to g$, where s is the state backed up and g is the backed-up value, or target, that s's estimated value is shifted toward.

3.3 MC backup: $S_t \to G_t$

TD(0) backup: $S_t \to R_{t+1} + \gamma\hat{v}(S_{t+1},\theta_t)$

n-step TD backup: $S_t \to G_t^{(n)}$

DP policy-evaluation backup: $s \to E_\pi(R_{t+1} + \gamma\hat{v}(S_{t+1},\theta_t) \mid S_t = s)$

here in DP, an arbitrary state s is backed up, whereas in other cases the state encountered in actual experience, S_t, is backed up.

4 We use function approximation methods for value prediction simply by passing to them the $s \to g$ of each backup as a training example. We then interpret the approximate function they produce as an estimated value function.

5. In the tabular case a continuous measure of prediction quality was not necessary because the learned value function

could come to equal the true value function exactly.

6. MSVE : mean squared value error

$$MSVE(\theta) \doteq \sum_{s \in S} d(s)\left(v_\pi(s) - \hat{v}(s,\theta)\right)^2$$

6.2 $d(s)$ is the fraction of time spent in s under the target policy π. This is called the on-policy distribution.

6.3 In continuing tasks, the on-policy distribution is the stationary distribution under π.

6.4 In an episodic task, the on-policy distribution is a little different in that it depends on how the initial states of episodes are chosen.

6.5 It is not completely clear that MSVE is the right performance objective for rl.

7. SGD methods are among the most widely used of all function approximation methods and are particularly well suited to online reinforcement learning.

7.2 the approximate value function $\hat{v}(s,\theta)$ is a differentiable function of θ for all $s \in S$.

8. SGD: $$\theta_{t+1} \doteq \theta_t - \frac{1}{2}\alpha \nabla \left(v_\pi(S_t) - \hat{v}(S_t,\theta_t)\right)^2$$

$$= \theta_t + \alpha\left(v_\pi(S_t) - \hat{v}(S_t,\theta_t)\right)\nabla \hat{v}(S_t,\theta_t)$$

8.2 the convergence results for SGD methods assume that α decreases over time.

9. We do not seek or expect to find a value function that has zero error for all states, but only an approximation that balances the errors in different states.

10. SGD method for stat-value prediction:

$\theta_{t+1} \doteq \theta_t + \alpha(U_t - \hat{v}(S_t,\theta_t))\nabla\hat{v}(S_t,\theta_t)$

here U_t is target value, not the true value $v_\pi(S_t)$. $S_t \to U_t$.

10.2 If U_t is an unbiased estimate, that is, if $E(U_t) = v_\pi(S_t)$, for each t, then θ_t is

11. condition: $\sum_{n=1}^{\infty}\alpha_n = \infty$, $\sum_{n=1}^{\infty}\alpha_n^2 < \infty$.

12. If U_t is an unbiased estimate, that is, if $E(U_t) = v_\pi(S_t)$, for each t, then θ_t is guaranteed to converge to a local optimum under the usual stochastic approximation conditions 11 for decreasing α.

13. ∵ $E(U_t) = E(G_t) = v_\pi(S_t)$

∴ the gradient-descent version of MC state-value prediction is guaranteed to find a locally optimal solution.

here G_t is MC target.

14. ∵ bootstrapping target isn't independent of θ_t.

∴ bootstrapping methods will not produce a true gradient-descent method. ∴ bootstrapping methods do not converge as robustly as gradient methods.

(one does not obtain the same guarantee).

14.2 bootstrapping methods take into account the effect of changing the weight vector θ_t on the estimate, but ignore its effect on the target. They include only a part of the gradient and, accordingly, we call them semi-gradient methods.

14.3 In some cases such as the linear case semi-gradient methods do converge reliably.

14.4 advantages

① significantly faster to learn.

② They enable learning to be continual and online.

15. (A)(B) (S)(P) At time t, t $t+1$

we can choose state A or state B, but we may never have this choice to choose B after time t.

16. semi-gradient TD(0)

$U_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, \theta)$.

17. $\nabla(y - f(x,\theta))^2 = -2(y - f(x,\theta)) f'(x,\theta)$

① for approximate methods, we update θ.

② for tabular methods, we update f directly, i.e. $f'(x,\theta) = 1$.

18. If $v(s,\theta)$ is not continuous about s in some sense, then $\hat{v}(s,\theta)$ can not be learnt. Same for all AI problem.

19. $\hat{v}(s,\theta) \doteq \theta^T \phi(s) \doteq \sum_i^n \theta_i \phi_i(s)$

~~ϕ_i~~ $\phi_i : S \to R$ are called basic functions.

19.2 $\hat{v}(\cdot,\theta)$ is a linear function of θ.

19.3 s may not be continual, but $\phi(s)$ can still be meaningful if $\phi(s)$ is continuous in some sense.

19.4 $\nabla \hat{v}(s,\theta) = \phi(s)$.

19.5 the linear SGD case is one of the most favorable for mathematical analysis.

19.6 In linear case there is only one optimum or one set of equally good optima.

19.7. The gradient MC converges to the global optimum of the MSVE under linear function approximation if α is reduced usual conditions.

19.8 The semi-gradient TD(0) converges to a point near the local optimum under linear function approximation.

19.8.2

$\theta_{t+1} \doteq \theta_t + \alpha(R_{t+1} + \gamma \theta_t^T \phi_{t+1} - \theta_t^T \phi_t)\phi_t$

Here $\phi_t \doteq \phi(S_t)$.

$\therefore \theta_{t+1} = \theta_t + \alpha(R_{t+1}\phi_t - \phi_t(\phi_t - \gamma\phi_{t+1})^T\theta_t)$.

$\therefore E(\theta_{t+1} | \theta_t) = \theta_t + \alpha(b - A\theta_t)$

$b \doteq E(R_{t+1}\phi_t)$

$A \doteq E(\phi_t(\phi_t - \gamma\phi_{t+1})^T)$

∴ if the system converges,

$\Rightarrow b - A\theta_{TD} = 0$

$\Rightarrow \theta_{TD} = A^{-1}b$.

θ_{TD} is called TD fixpoint.

19.8.3 proof of convergence of linear TD(0)

19.8.4 In the continuing case:

$MSVE(\theta_{TD}) \leq \frac{1}{1-\gamma} \min_{\theta} MSVE(\theta)$.

19.9 n-step semi-gradient TD

20. One needs to introduce features for combinations of feature values when using linear function approximation methods.

21. for multi-dimensional continuous state spaces, function approximation for rl has much in common with the tasks of interpolation and regression.

22. represent highly-complex interactions among the state variables.

$\Leftrightarrow$ allow for more accurate approximations of more complicated functions.

23. $s \in \mathbb{R}^d$, $\phi_i(s) = \prod_{j=1}^{d} s_j^{c_{i,j}}$

ϕ_i is polynomial basis, $c_{ij} \in \{0, 1, \dots, N\}$, $N \geq 0$.

$\Rightarrow$ define $(N+1)^d$ distinct functions for dimension d. They make up the order-N polynomial basis.

24. Fourier series expess periodic functions as a weighted sum of sine and cosine basis functions of different frequencies.

24.2 f is periodic $\Leftrightarrow f(x) = f(x+T)$ for all x and some period T.

24.3 With enough basis functions essentially any function can be approximated as accurately as desired.

24.4 even function $\Leftrightarrow$ function that is symmetric about origin.

odd function $\Leftrightarrow$ function that is anti-symmetric about the origin.

24.5 we can represent any even function with just the cosine basis functions. (here the function has to

be mathematically well-behaved.).

24.6 The one-dimensional order-N Fourier cosine basis consists of the $N+1$ functions

$$\phi_i(s) = \cos(i\pi s), \quad s \in [0,1]$$
$$i = 0, 1, \dots, N.$$

24.7 Unlike polynomial basis functions, Fourier basis functions are always bounded and do not require exponentiation.

24.8 For a state space that is the d-dimensional unit hypercube with the origin in one corner, $s \in \mathbb{R}^d$, $s_i \in [0,1]$.

$$\phi_i(s) = \cos(\pi c^i \cdot s).$$

ϕ_i is order-N Fourier cosine basis, $c^i = (c_1^i, \dots, c_d^i)^T$,

$$c_j^i \in \{0, \dots, N\}, \forall j = 1, \dots, d,$$
$$i = 0, \dots, (N+1)^d$$

This defines a function for each of the $(N+1)^d$ possible integer vectors c^i.

24.9 Not surprisingly, Fourier basis functions have trouble with discontinuities.

24.10 As is true for polynomial approximation, the number of basis functions in the order-N Fourier cosine basis grows exponentially with the state space dimension.

24.11 Advantages of Fourier basis functions are that it is easy to select functions by setting the c^i vectors to account for suspected interactions among the state variables, and by limiting the values in the c^i vectors so that the approximation can filter out high frequency components considered to be noise.

24.12 We do not recommend using the polynomial basis for online learning.

25. Given a state, indicate within which

circles the state lies, and thus coarsely code for its location. Representing a state with features that overlap in this way is known as coarse coding.

25.2 the shape of the features will determine the nature of the generalization.

25.3 Features with large receptive fields give broad generalization, and are able to make discriminations much finer than the width of the receptive fields, because ultimately the finest discrimination is controlled more by the total number of features, though initial generalization is indeed controlled by the size and shape of the receptive fields.

26. tile coding is a form of coarse coding the state space, but that for multi-dimensional continuous spaces still form an exhaustive partition. that is flexible and computationally effecient.

26.2 It may be the most practical feature representation for modern sequential digital computers. In the sense that

26.3 ∵ tile coding works with partitions,
∴ the overall number of features that are active at one time is the same for any state.

26.4 tile coding also gains computational advantages from its use of binary feature vectors.

26.5 generalization occurs to states other than the one trained if these states fall within any of the same tiles, proportional to the number of tiles in common.

26.6. Hashing produces tiles consisting of noncontiguous, disjoint regions randomly spread throughout

26.7 hashing frees us from the curse of dimensionality

memory requirements need not be exponential in the number of dimensions, but need merely match the real demands of the task.

27. RBF

radial basis functions (RBFs) are the natural generalization of coarse coding to continuous-valued features.

27.2 a typical RBF feature:

$$\phi_i(s) \doteq \exp\left(-\frac{\|s-c_i\|^2}{2\sigma_i^2}\right)$$

27.3 The primary advantage of RBFs over binary features is that they produce approximate functions that vary smoothly and are differentiable. Although this is appealing, in most cases it has no practical significance.

27.4 An RBF network is a linear function approximator using RBFs for its features.

27.5 some learning methods for RBF networks change the centers and widths of the features as well, bringing them into the realm of nonlinear function approximators. Nonlinear methods may be able to fit target functions much more precisely.

28. ANNs

28.2 ANNs are widely used for nonlinear function approximation.

28.3 both feedforward and recurrent ANNs have been used in rl.

28.4 an ANN with a single hidden layer having a large enough finite number of sigmoid units can approximate any continuous function on a compact region of the networks input space to any degree of accuracy. Here nonlinearity is essential.

28.5 creating the hierarchical representations without relying exclusively on hand-crafted features has been an enduring

challenge for AI. This is why learning algorithms for ANNs with hidden layers have received so much attention over the years.

28.6 backpropagation algorithm consists of alternating forward and backward passes through the network. Each forward pass computes the activation of each unit given the current activations of the network's input units. After each forward pass, a backward pass efficiently computes a partial derivative for each weight.

28.7 We can train ANNs using rl principles instead of backpropagation. These methods are less efficient than bp algorithm, but they may be closer to how real neural networks learn.

28.8 bp algorithm does not work well for deeper ANNs.

28.9 A particularly effective method for reducing overfitting by deep ANNs is the dropout method.

28.10 Because of its special architecture, a deep convolutional network can be trained by bp algorithm.

28.11 units in the same feature map share the same weights. In CNN. This means that a feature map detects the same feature no matter where it is located in the input array.

28.12. The subsampling layers reduce the network's sensitivity to the spatial locations of the features detected, that is, they help make the CNN's responses spatially invariant.

29. LSTD

29.2 Least-Squares TD algorithm is the most data efficient form of linear TD(0), but it is also much more expensive computationally. It requires computation proportional to the square of the number of weights, whereas all other methods are of complexity linear in the number of weights.

Chapter 10

有的人先联明，有的人后联明.

On-policy control with approximation.

1. $S_t, A_t \to U_t$

The target U_t can be any approximation of $q_\pi(S_t, A_t)$

1.2 the general gradient-descent update for action-value prediction is

$$\theta_{t+1} \doteq \theta_t + \alpha(U_t - \hat{q}(S_t, A_t, \theta_t)) \nabla \hat{q}(S_t, A_t, \theta_t)$$

1.3 e.x. one-step Sarsa

$$\theta_{t+1} \doteq \theta_t + \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \theta_t) - \hat{q}(S_t, A_t, \theta_t)) \nabla \hat{q}(S_t, A_t, \theta_t)$$

we call this method episodic semi-gradient one-step Sarsa.

1.4 To form control methods, we need to couple such action-value prediction methods with techniques for policy improvement and action selection. Suitable techniques applicable to continuous actions, or to actions from large discrete sets, are a topic of ongoing research with as yet no clear resolution.

2. n-step semi-gradient Sarsa

2.2 n-step return.

$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, \theta_{t+n-1})$$

$n \geq 1, \ 0 \leq t < T-n$

$G_t^{(n)} \doteq G_t$, if $t+n \geq T$.

2.3 n-step update equation:

$$\theta_{t+n} \doteq \theta_{t+n-1} + \alpha(G_t^{(n)} - \hat{q}(S_t, A_t, \theta_{t+n-1})) \cdot \nabla \hat{q}(S_t, A_t, \theta_{t+n-1}), \ 0 \leq t < T.$$

3. three classical settings for formulating the goal in MDPs.

① episodic setting

② discounted setting

③ average reward setting.

4. In the average-reward setting, the quality of a policy π is defined as the average rate of reward while following that policy.

4.2 $$\eta(\pi) \doteq \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} E(R_t | A_{0:t-1} \sim \pi)$$
$$= \lim_{t\to\infty} E(R_t | A_{0:t-1} \sim \pi)$$
$$= \sum_s d_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) r$$

4.3 $d_\pi(\pi)$ is the steady-state distribution,

$$d_\pi(s) \doteq \lim_{t\to\infty} \Pr(S_t = s | A_{0:t-1} \sim \pi),$$

今日，初次见面，人比画美。

which is assumed to exist and to be independent of S_0. This property is known as ergodicity. It means that where the MDP starts or any early decision made by the agent can have only a temporary effect; in the long run your expectation of being in a state depends only on the policy and the MDP transition probabilities.

4.4 Ergodicity is sufficient to guarantee the existence of the limits in the equations in 4.2.

4.5 We consider all policies that attain the maximal value of $\eta(\pi)$ to be optimal.

4.6 steady state distribution is the special distribution under which, if you select actions according to π, you remain in the same distribution.

$$\sum_s d_\pi(s) \sum_a \pi(a|s,\theta)\, p(s'|s,a) = d_\pi(s')$$

4.7 differential return:

$$G_t \doteq R_{t+1} - \eta(\pi) + R_{t+2} - \eta(\pi) + R_{t+3} - \eta(\pi) + \cdots$$

4.8 differential value functions:

$$v_\pi(s) \doteq E_\pi(G_t | S_t = s)$$

$$q_\pi(s,a) \doteq E_\pi(G_t | S_t = s, A_t = a)$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(s',r|s,a)(r - \eta(\pi) + v_\pi(s'))$$

$$q_\pi(s,a) = \sum_{r,s'} p(s',r|s,a)(r - \eta(\pi) + \sum_{a'} \pi(a'|s')\, q_\pi(s',a'))$$

4.9 Bellman equations

$$v_*(s) = \max_a \sum_{r,s'} p(s',r|s,a)(r - \eta(\pi) + v_*(s'))$$

$$q_*(s,a) = \sum_{r,s'} p(s',r|s,a)(r - \eta(\pi) + \max_{a'} q_*(s',a'))$$

4.10. differential form of TD errors

$$\delta_t \doteq R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1},\theta) - \hat{v}(S_t,\theta)$$

$$\delta_t \doteq R_{t+1} - \bar{R}_t + \hat{q}(S_{t+1}, A_{t+1}, \theta) - \hat{q}(S_t, A_t, \theta)$$

where $\bar{R}$ is an estimate of the average reward $\eta(\pi)$.

4.11 the average reward version of semi-gradient Sarsa:

$$\theta_{t+1} \doteq \theta_t + \alpha \delta_t \nabla \hat{q}(S_t, A_t, \theta_t)$$

where δ_t is given by 4.10.

4.12 differential semi-gradient Sarsa for control:

Learn how to play and ~~speak~~ talk. Improve myself. ⇒ after 3 days, still keep in touch.

$\delta \leftarrow R - \bar{R} + \hat{q}(S', A', \theta) - \hat{q}(S, A, \theta)$

$\bar{R} \leftarrow \bar{R} + \beta\delta$,

$\theta \leftarrow \theta + \alpha\delta\nabla\hat{q}(S, A, \theta)$

here ~~$\bar{R} \leftarrow \bar{R} + \beta\delta$~~ may be wrong, ②

it should be $\bar{R} \leftarrow \bar{R} + \beta(R - \bar{R})$.

5. deprecating the discounted setting

5.2 $J(\pi) = \sum_s d_\pi(s) v_\pi^\gamma(s)$

$= \sum_s d_\pi(s) \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)(r + \gamma v_\pi^\gamma(s'))$

$= \eta(\pi) + \gamma \sum_{s'} v_\pi^\gamma(s') \sum_s d_\pi(s) \sum_a \pi(a|s) p(s'|s, a)$

$= \eta(\pi) + \gamma \sum_{s'} v_\pi^\gamma(s') d_\pi(s')$

$= \eta(\pi) + \gamma J(\pi)$

$\therefore J(\pi) = \frac{1}{1-\gamma}\eta(\pi)$

$J(\pi)$ orders policies identically to the undiscounted objective.

6. n-step differential semi-gradient Sarsa

6.2 $G_t^{(n)} \doteq R_{t+1} - \bar{R} + R_{t+2} - \bar{R} + \cdots + R_{t+n} - \bar{R} + \hat{q}(S_{t+n}, A_{t+n}, \theta)$, $t+n < T$, $n \geq 1$

$G_t^{(n)} \doteq G_t$, $t+n \geq T$.

6.3 n-step TD error

$\delta_t \doteq G_t^{(n)} - \hat{q}(S_t, A_t, \theta)$.

6.4 $\theta_{t+1} \doteq \theta_t + \alpha\delta_t\nabla\hat{q}(S_t, A_t, \theta_t)$

8.3.7 In the approximate case most policies cannot be represented by a value function.

8.

8.2 tabular case { episodic case / continuing case

approximate case { episodic case / continuing case

8.3.1 for the continuing case ~~we have to introduce a~~ the discounted formulation cannot be carried over to control in the presence of approximations.

8.3.3 The arbitrary policies that remain need to be ranked, and the $\eta(\pi)$ provides an effective way to do this.

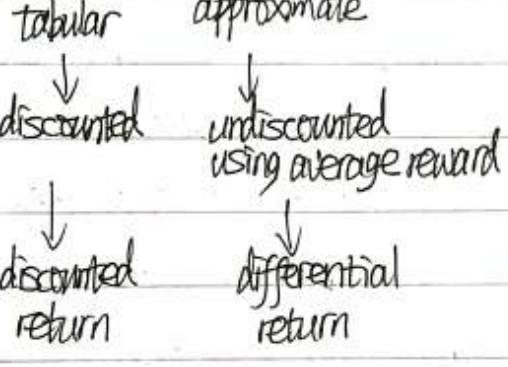
7. The ordering of all policies in the average discounted return setting would be exactly the same as in the average-reward setting.

Chapter 11

今日份和tt联系

off-policy methods with approximation

1. explore the convergence problem.
2. tabular case corresponds to a special case of function approximation.
3. episodic / continuing



4. semi-gradient off-policy TD(0)

$$\theta_{t+1} \doteq \theta_t + \alpha \rho_t \delta_t \nabla \hat{v}(S_t, \theta_t)$$

where $\rho_t \doteq \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$

$$\delta_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, \theta_t) - \hat{v}(S_t, \theta_t)$$

when episodic and potentially discounted

$$\delta_t \doteq R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1}, \theta_t) - \hat{v}(S_t, \theta_t)$$

when continuing and undiscounted using average reward.

5. semi-gradient Expected Sarsa

$$\theta_{t+1} \doteq \theta_t + \alpha \delta_t \nabla \hat{q}(S_t, A_t, \theta_t)$$

where $\delta_t \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a, \theta_t) - \hat{q}(S_t, A_t, \theta_t)$ (episodic)

or

$$\delta_t \doteq R_{t+1} - \bar{R}_t + \sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a, \theta_t) - \hat{q}(S_t, A_t, \theta_t)$$ (continuing)

5.2 This algorithm does not use importance sampling. In the tabular case it is clear that this is appropriate, but with function approximation it is a judgement call. (?)

6. n-step version of semi-gradient Expected Sarsa

$$\theta_{t+n} \doteq \theta_{t+n-1} + \alpha \rho_{t+1} \cdots \rho_{t+n-1} \cdot (G_t^{(n)} - \hat{q}(S_t, A_t, \theta_{t+n-1})) \nabla \hat{q}(S_t, A_t, \theta_{t+n-1})$$

where $G_t^{(n)} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a|S_{t+n}) \hat{q}(S_{t+n}, a, \theta_{t+n-1})$ (episodic)

$$G_t^{(n)} \doteq R_{t+1} - \bar{R}_t + \cdots + R_{t+n} - \bar{R}_{t+n-1} + \sum_a \pi(a|S_{t+n}) \hat{q}(S_{t+n}, a, \theta_{t+n-1})$$ (continuing)

6.2 for episodes, $\rho_t \doteq 1$, if $t \geq T$,
$G_t^{(n)} \doteq G_t$, if $t+n \geq T$.

7. n-step semi-gradient tree-backup algorithm
$\theta_{t+n} \doteq \theta_{t+n-1} + \alpha(G_t^{(n)} - \hat{q}(S_t, A_t, \theta_{t+n-1})) \cdot$
$\nabla \hat{q}(S_t, A_t, \theta_{t+n-1})$,
where $G_t^{(n)} \doteq \hat{q}(S_t, A_t, \theta_{t-1}) + \sum_{k=t}^{t+n-1} \delta_k \prod_{i=t+1}^{k} \gamma \pi(A_i|S_i)$
with δ_t as defined for Expected Sarsa (5), $G_t^{(n)} \doteq G_t$ if $t+n \geq T$, and $\gamma = 1$ in the continuing case.

8. n-step semi-gradient $Q(\sigma)$

9. episode terminates on all transitions with 1% probability, much like a $\gamma = 0.99$ discount rate. ↳

10. Baird's counterexample

10.2 Even the simplest combination of bootstrapping and function approximation can be unstable if the backups are not done according to the on-policy distribution.

10.3 It may be possible to guarantee convergence of Q-learning as long as the behavior policy is sufficiently close to the estimation policy, for example, when it is the ε-greedy policy. To the best of our knowledge, Q-learning has never been found to diverge in this case, but there has been no theoretical analysis.

10.4 stability is guaranteed for function approximation methods that do not extrapolate from the observed targets. These methods, called averagers, include nearest neighbor methods and local weighted regression, but not popular methods such as tile coding and backpropagation.

11. The deadly triad.

11.2 ① training on a distribution of transitions other than that naturally generated by the process whose expectation is being estimated. (e.g., off-policy learning)
② scalable function approximation. (e.g, linear semi-gradient).
③ bootstrapping (e.g., DP, TD learning)

11.3 Any two of these three is fine; the danger arises only in the presence of all three.

Chapter 12

eligibility traces

1. TD(λ), λ refers to the use of an eligibility trace.
2. eligibility traces unify and generalize TD and MC methods, producing a family of methods spanning a spectrum with MC at one end ($\lambda=1$) and one-step TD at the other ($\lambda=0$).

2.2 eligibility traces also provide a way of implementing MC online and on continuing problems without episodes.

3. ① short-term memory vector, the eligibility trace $e_t \in R^n$.
   ② long-term weight vector $\theta_t \in R^n$
   ③ trace-decay parameter $\lambda \in [0,1]$

4. The primary computational advantage of eligibility traces over n-step methods is that only a single trace vector is required rather than a store of the last n feature vectors. Learning also occurs continually and uniformly in time rather than being delayed and then catching up at the end of the episode. In addition learning can occur and affect behavior immediately after a state is encountered rather than being delayed n steps.

5. forward views: looking forward from the updated state.
   backward views: looking backward to recently visited states.
   5.2 forward views are always somewhat complex to implement.

6. tabular and state aggregation cases are special cases of linear function approximation.

7.

7.2 n-step return

$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, \theta_{t+n-1}),$$

$0 \le t \le T-n$

$n \ge 1$.

7.3 average of n-step returns possesses an error reduction

property similar to that of individual n-step returns and thus can be used to construct backups with guaranteed convergence properties.

7.4 averaging produces a substantial new range of algorithms. For example, one could average one-step and infinite-step returns to obtain another way of interrelating TD and MC. In principle, one could even average experience-based backups with DP backups to get a simple combination of experience-based and model-based methods.

7.5 ensemble methods ⟷ averaging ε.

7.6 A backup that averages simpler component backups is called a compound backup.

8. TD(λ) algorithm can be understood as one particular way of averaging n-step backups.

8.2 λ-return

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

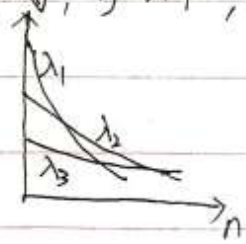
8.3 If $\lambda=0$, then the overall backup reduces to its first component, the one-step TD backup, whereas if $\lambda=1$, then the overall backup reduces to its last component, the MC backup.

8.4 $f(\lambda,n)=(1-\lambda)\lambda^n$

$\lambda\in[0,1]$, $n\geq 1$, $n\in Z$.

① if ↑↘, if λ↑, n fixed.

② if ↓, if n↑, λ fixed.



$\lambda_1<\lambda_2<\lambda_3$

8.5 If we want, we can separate these post-termination terms from the main sum, yielding

$$G_t^\lambda=(1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_t^{(n)}+\lambda^{T-t-1}G_t$$

9. a rate factor such as λ is sometimes an awkward way of characterizing the speed of the decay.

Lily 唱的生日歌 너무 좋다.

For some purposes it is better to specify a time constant, or half-life, $T_{\frac{1}{2}}$, the time by which the weighting sequence will have fallen to half of its initial value.

10. off-line λ-return algorithm.

10.2 As an off-line algorithm, it makes no changes to the weight vector during the episode. Then, at the end of the episode, a whole sequence of off-line updates are made according to our usual semi-gradient rule, using the λ-return as the target:

$$\theta_{t+1} \doteq \theta_t + \alpha\left(G_t^\lambda - \hat{v}(S_t, \theta_t)\right)\nabla \hat{v}(S_t, \theta_t), \quad t = 0, \cdots, T-1.$$

10.3 The λ-return gives us an alternative way of moving smoothly between MC and one-step TD methods that can be compared with the n-step TD way. ~~Chapter 7.~~ 10.4 the overall performance of the off-line λ-return algorithms is comparable to that of the n-step algorithms.

10.5

n-step TD method → off-line λ-return alg

discrete → continuous

course coding → RBFs

10.6 The approach is forward view of a learning algorithm. We decide how to update each state by looking forward to future rewards and states.

11. TD(λ) is one of the oldest and most widely used algorithms in rl. It was the first algorithm for which a formal relationship was shown between a more theoretical forward view and a more computational congenial backward view using eligibility traces.

11.2 TD(λ) improves over the off-line λ-return algorithm in three ways. First it updates the weight vector on every

step of an episode rather than only at the end, ~~this~~ Second, its computations are equally distributed in time rather than all at the end of the episode. And third, it can be applied to continuing problems rather than just episodic problems.

11.3 $e_0 \doteq 0$, $e_t \doteq \nabla \hat{v}(S_t, \theta_t) + \gamma\lambda e_{t-1}$ where ~~γ~~ is the discount rate.

11.4 the trace is said to indicate the eligibility of each component of the weight vector for undergoing learning changes should a reinforcing event occur. The reinforcing events are ~~TD~~ one-step errors,

$$\delta_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, \theta_t) - \hat{v}(S_t, \theta_t)$$

11.5 $\theta_{t+1} \doteq \theta_t + \alpha \delta_t e_t$

11.6 eligibility traces $\Leftrightarrow$ momentum. in form
It has the same idea ~~with~~ as momentum, but it has further explaination.

11.7 TD(λ) is oriented backward in time. At each moment we look at the current TD error and assign it backward to each prior state according to how much that state contributed to the current eligibility trace at that time.

11.8 if $\lambda = 0$, $\Rightarrow e_t = \nabla \hat{v}(S_t, \theta_t)$
$\therefore$ 11.5 reduces to the ~~one-step~~ semi-gradient TD update.
$\therefore$ we call ~~that~~ TD(0).

11.9 if $\lambda = 1$, then the credit given to earlier states falls only by γ per step. This turns out to be just the right thing to do to achieve MC behavior.

11.10 If $\lambda = 1$ and $\gamma = 1$, then the eligibility traces do not decay at all with time. In this case, the method behaves like MC for undiscounted, episodic tasks.

11.11 TD(1) is a way of implementing MC that is more general and that significantly increases their range of applicability. Whereas the earlier MC were limited to episodic tasks, TD(1) can be applied to discounted continuing tasks as well. Moreover, TD(1) can be performed incrementally and on-line.

11.12 If α is chosen larger, λ-return alg is only a little worse whereas TD(λ) is much worse and may even be unstable. This is a weakness of TD(λ).

12. online λ-return algorithm.

12.2 h-truncated λ-return

$$G_t^{\lambda|h} \doteq (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_t^{(n)} + \lambda^{h-t-1}G_t^{(h-t)}$$

$$0 \le t < h \le T,$$

12.3 $$\theta_{t+1}^h \doteq \theta_t^h + \alpha\left(G_t^{\lambda|h} - \hat{v}(S_t,\theta_t^h)\right)\nabla\hat{v}(S_t,\theta_t^h),$$

$$\forall\ 0 \le t < h \le T$$

12.4 $\theta_0^h \doteq \theta_0$, $\theta_h \doteq \theta_h^h$, $\theta_T \doteq \theta_T^T$, $\theta_t \doteq \theta_t^t$

12.5 h is horizon.

12.6 the online λ-return alg is fully online, determining a new weight vector θ_t at each step t during an episode, using only info available at time t. It's main drawback is that it is computationally complex, passing over the entire episode so far on every step.

Note that it is strictly more complex than the off-line λ-return alg, which passes through all the steps at the time of termination but does not make any updates during the episode. The online alg can be expected to perform better than the off-line one.

B. true online TD(λ) alg

13.2 It is "truer" to the idea of the online TD(λ) alg,

tuner even than the TD(λ) alg itself.

13.3 The sequence of weight vectors produced by the on-line λ-return alg can be arranged in a triangle:

$$\begin{array}{ccccc} \theta_0^0 & & & & \\ \theta_0^1 & \theta_1^1 & & & \\ \theta_0^2 & \theta_1^2 & \theta_2^2 & & \\ \vdots & & & & \\ \theta_0^T & \theta_1^T & \theta_2^T & \cdots & \theta_T^T \end{array}$$

Really only the weight vectors on the diagonal, the θ_t^t, need to be produced by the alg. The strategy is to find a compact, ~~efficient~~ way of computing each θ_t^t from the one before.

13.4 for linear case in which $\hat{v}(s,\theta)=\theta^T\phi(s)$, the true ~~online~~ online TD(λ) alg is:

$$\theta_{t+1} \doteq \theta_t + \alpha\delta_t e_t + \alpha(\theta_t^T\phi_t - \theta_{t-1}^T\phi_t)(e_t - \phi_t)$$

where $\phi_t \doteq \phi(S_t)$,

$$e_t \doteq \gamma\lambda e_{t-1} + (1-\alpha\gamma\lambda e_{t-1}^T\phi_t)\phi_t$$

13.5 a_t and e_t are updated on each time step $t<T$ and then, at time T when G is observed, they are used to compute θ_T.

13.5 This alg has been proven to produce exactly the same sequence of weight vectors, θ_t, $\forall\, 0\le t\le T$, as the on-line λ-return alg. (ε)

14. eligibility trace in ~~ES~~ true online TD(λ) is called a dutch trace.

The trace used in TD(λ) is called an accumulating trace.

15. Dutch traces in MC learning

15.2 $\theta_{t+1} \doteq \theta_t + \alpha(G-\theta_t^T\phi_t)\phi_t$ $\quad 0\le t<T$

We assume return G is a single reward received at the end of the episode and that there is no discounting.

15.3 $\theta_T = \theta_{T-1} + \alpha(G-\theta_{T-1}^T\phi_{T-1})\phi_{T-1} = F_{T-1}\phi_{T-1} + \alpha G\phi_{T-1}$

where $F_t \doteq I-\alpha\phi_t\phi_t^T$ is a ~~forgetting~~, or fading, matrix

15.4 $\therefore \theta_T = a_{T-1} + \alpha G e_{T-1}$

where $a_{T-1} = F_{T-1}F_{T-2}\cdots F_0\theta_0$

$e_{T-1} = \sum_{k=0}^{T-1} F_{T-1}F_{T-2}\cdots F_{k+1}\phi_k = e_{t-1} + (1-\alpha e_{t-1}^T\phi_t)\phi_t$

Chapter 13
Policy Gradient Methods

1.① methods that learn the values of actions and then select actions based on their estimated action values.

② methods that learn a parameterized policy that can select actions without consulting a value function.

1.3 $\theta \in \mathbb{R}^n$ policy weight vector

w : value-function's weight, as in $\hat{v}(s,w)$

2. policy gradient methods.

$\theta_{t+1} \doteq \theta_t + \alpha \widehat{\nabla \eta(\theta_t)}$

where $\widehat{\nabla \eta(\theta_t)}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to its argument θ_t.

2.2 $\eta(\theta)$ is performance measure with respect to the policy weights.

2.3 methods that learn approximations to both policy and value functions are often called actor-critic methods, where 'actor' is a reference to the learned policy, and 'critic' refers to the learned value function, usually a state-value function.

2.4 episodic case: $\eta(\theta) = v_{\pi_\theta}(s_0)$

continuing case: $\eta(\theta) = r(\theta)$

where $v_{\pi_\theta}(s_0)$ is the value of the start state under π_θ.

$r(\theta)$ is average reward rate.

3. policy approximation.

3.2 to ensure exploration we generally require that $\pi(a|s,\theta) \in (0,1)$, $\forall a, s, \theta$.

3.3 policy-based methods offer useful ways of dealing with continuous action spaces.

3.4 if the action space is discrete and not too large, then a natural kind of parameterization is to form parameterized numerical preferences $h(s,a,\theta) \in \mathbb{R}$ for each state-action pair.

3.5 $\pi(a|s,\theta) \doteq \frac{\exp(h(s,a,\theta))}{\sum_b \exp(h(s,b,\theta))}$

3.6 ① An immediate advantage of selecting actions according to the softmax in action preferences (3.5) is that the approximate policy can approach determinism, whereas with ε-greedy action selection over action values there is always an ε probability of selecting a random action.

② Perhaps the simplest advantage that policy parameterization may have over action-value parameterization is that the policy may be a simpler function to approximate.

③ Action-value methods have no natural way of finding stochastic optimal policies, whereas policy approximating methods can. This is a third significant advantage of policy-based methods.

④ the choice of policy parameterization is sometimes a good way of injecting prior knowledge about the desired form of the policy into the rl system.

4. Policy Gradient Theorem

4.2 assume that every episode starts in some particular (non-random) state s_0. Then, in the episodic case we define performance as $\eta(\theta) \doteq v_{\pi_\theta}(s_0)$

4.3 policy gradient theorem:

$$\nabla \eta(\theta) = \sum_s d_\pi(s) \sum_a q_\pi(s,a) \nabla_\theta \pi(a|s,\theta).$$

4.4 proof:

$$\begin{aligned}
\nabla v_\pi(s) &= \nabla\left(\sum_a \pi(a|s) q_\pi(s,a)\right) \\
&= \sum_a \left(\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a)\right) \\
&= \sum_a \left(\nabla \pi(a|s) q_\pi(s,a) + \right. \\
&\qquad \left. \pi(a|s) \nabla \sum_{s',r} p(s',r|s,a)(r + \gamma v_\pi(s'))\right) \\
&= \sum_a \nabla \pi(a|s) q_\pi(s,a) + \\
&\qquad \sum_a \pi(a|s) \sum_{s'} \gamma p(s'|s,a) \nabla v_\pi(s') \\
&= \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&\qquad + \gamma \sum_{s'} P(s \to s') \sum_{a'} \nabla \pi(a'|s') q_\pi(s',a') \\
&\qquad + \gamma^2 \sum_{s''} P(s \to s'') \sum_{a''} \nabla \pi(a''|s'') q_\pi(s'',a'') \\
&\qquad + \cdots \\
&= \sum_{x \in S} \sum_{k=0}^{\infty} \gamma^k P(s \to x | k) \sum_a \nabla \pi(a|x) q_\pi(x,a) \\
&= \sum_{x \in S} d_\pi(x) \sum_a \nabla \pi(a|x) q_\pi(x,a). \quad \text{OK.}
\end{aligned}$$

5. REINFORCE

5.2 $\nabla\eta(\theta)=\sum_s d_\pi(s)\sum_a q_\pi(s,a)\nabla_\theta\pi(a|s,\theta)$

$\therefore \nabla\eta(\theta)=E_\pi(\gamma^t\sum_a q_\pi(S_t,a)\nabla_\theta\pi(a|S_t,\theta))$ (2)

$\therefore \nabla\eta(\theta)=E_\pi(\gamma^t\sum_a\pi(a|S_t,\theta)q_\pi(S_t,a)\frac{\nabla_\theta\pi(a|S_t,\theta)}{\pi(a|S_t,\theta)})$

$=E_\pi(\gamma^t q_\pi(S_t,A_t)\frac{\nabla_\theta\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)})$

$=E_\pi(\gamma^t G_t\frac{\nabla_\theta\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)})$

$(\because E_\pi(G_t|S_t,A_t)=q_\pi(S_t,A_t))$

5.3 REINFORCE:

$$\theta_{t+1}\doteq\theta_t+\alpha\gamma^t G_t\frac{\nabla_\theta\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$$

5.4 REINFORCE uses the complete return from time t, In this sense REINFORCE is a MC algorithm and is well defined only for the episodic case.

5.5 $\frac{\nabla_\theta\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$ has been given several names and notations in the literature. We will refer to it simply as eligibility vector.

5.6. the expected update over an episode is in the same direction as the performance gradient. (Not proved).

5.7 As a MC method it may be of high variance and thus slow to learn.

6. REINFORCE with Baseline.

6.2 $\because \sum_a b(s)\nabla_\theta\pi(a|s,\theta)=b(s)\nabla_\theta 1=0,$

$\forall s\in S.$

$\therefore \nabla\eta(\theta)=\sum_s d_\pi(s)\sum_a(q_\pi(s,a)-b(s))\nabla_\theta\pi(a|s,\theta)$

6.3 $\because \theta_{t+1}\doteq\theta_t+\alpha(G_t-b(S_t))\frac{\nabla_\theta\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$

6.4 In general, the baseline leaves (no γ^t) the expected value of the update unchanged, but it can have a large effect on its variance.

6.5 One natural choice for the baseline is an estimate of the state value, $\hat{v}(S_t,w)$, where $w\in R^m$ is a second learned weight vector.

7. Although the REINFORCE-with-baseline method learns both a policy and a state-value function, we do not consider it to be an actor-critic method because its state-value function is

used only as a baseline, not as a critic. That is, it is not used for bootstrapping, but only as a baseline for the state being updated. This is a useful distinction, for only through bootstrapping do we introduce bias and an asymptotic dependence on the quality of the function approximation.

8. Actor-Critic Methods

8.2 One-step actor-critic (episodic)

$$\theta_{t+1} \doteq \theta_t + \alpha\left(G_t^{(1)} - \hat{v}(S_t,w)\right)\frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$$

$$= \theta_t + \alpha\left(R_{t+1} + \gamma\hat{v}(S_{t+1},w) - \hat{v}(S_t,w)\right)\frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \quad (\text{no } \gamma^t \text{ yet})$$

The natural state-value function learning method to pair with this is semi-gradient TD(0).

8.3 actor-critic with eligibility traces (episodic)

9. for continuing problems without episode boundaries,

$$\eta(\theta) \doteq r(\theta) = \lim_{n\to\infty}\frac{1}{n}\sum E(R_t|\theta_0=\theta_1=\cdots=\theta_{t-1}=\theta)$$

$$= \lim_{t\to\infty} E(R_t|\theta_0=\theta_1=\cdots=\theta_{t-1}=\theta)$$

$$= \sum_s d_{\pi_\theta}(s)\sum_a \pi_\theta(a|s,\theta)\sum_{s',r} p(s',r|s,a)\,r$$

9.3 d_π is steady-state distribution under π.

$$d_\pi(s) \doteq \lim_{t\to\infty} P\{S_t=s|A_{0:t}\sim\pi\}.$$

9.4 $v_\pi(s) \doteq \sum_{k=1}^{\infty} E_\pi(R_{t+k} - r(\pi)|S_t=s)$

9.5 proof of policy gradient theorem (continuing case).

$\because \nabla v_\pi(s) = \nabla\left(\sum_a \pi(a|s) q_\pi(s,a)\right), \forall s\in S.$

$$= \sum_a\left(\nabla\pi(a|s) q_\pi(s,a) + \pi(a|s)\nabla\sum_{s',r} p(s',r|s,a)(r - r(\theta) + v_\pi(s'))\right)$$

$\therefore \nabla r(\theta)$

$$= \sum_a \nabla\pi(a|s) q_\pi(s,a) + \sum_a \pi(a|s)\sum_{s'} p(s'|s,a)\nabla v_\pi(s') - \nabla v_\pi(s), \quad \forall s\in S.$$

$\because \sum_s d_\pi(s) = 1$

$\therefore \nabla\eta(\theta) = \nabla r(\theta)$

$$= \sum_s d_\pi(s)\sum_a \nabla\pi(a|s) q_\pi(s,a) + \sum_{s'}\sum_s d_\pi(s)\sum_a \pi(a|s) p(s'|s,a)\cdot\nabla v_\pi(s') - \sum_s d_\pi(s)\nabla v_\pi(s)$$

$$= \sum_s d_\pi(s)\sum_a \nabla\pi(a|s) q_\pi(s,a) + \sum_{s'} d_\pi(s')\nabla v_\pi(s') - \sum_s d_\pi(s)\nabla v_\pi(s)$$

$$= \sum_s d_\pi(s)\sum_a \nabla\pi(a|s) q_\pi(s,a).$$

10. policy-based methods offer practical ways of dealing with large actions spaces, even continuous spaces with an infinite number of actions.