tic tac toe

$nr \leftarrow 3$, $nc \leftarrow 4$, $size \leftarrow nr \cdot nc$.

1. state

1.2. $M \leftarrow M(nr, nc) \leftarrow 0$

M is data, state, $M_{ij} \in \{-1, 0, 1\}$

$f(M) = h$, f is bijective. h is hashvalue.

1.2.1 f: $x \leftarrow 0$

$\forall i \in M$, $x \leftarrow 3x + (i+1)$

$\therefore h \leftarrow x$

1.2.2 $\therefore$ h is ternary number:

$(M_{11}+1, M_{12}+1, \cdots, M_{nr,nc}+1)$.

1.3 is end

if $end \neq -1$, return end.

// $end \in \{-1, 0, 1\}$

sequence $re \leftarrow \phi$.

$\forall i \in [1, nr]$,

re add $sum(M_{i,:})$

$\forall i \in [1, nc]$,

re add $sum(M_{:,i})$

if $nr = nc$

$t \leftarrow trace(M)$

$rt \leftarrow$ reverse $trace(M)$

re add t.

re add rt.

$\forall i \in [1, |re|]$

$x \leftarrow re(i)$

$x_0 \leftarrow \begin{cases} nc & \text{if } i \leq nr \\ nr, & \text{else} \end{cases}$

if $x = x_0$

$w \leftarrow 1$, $end \leftarrow 1$, stop

if $x = -x_0$

$w \leftarrow -1$, $end \leftarrow 1$, stop

if $\|M\|_1 = size$

$w \leftarrow 0$, $end \leftarrow 1$, stop

$end \leftarrow 0$.

1.4. next state $(i, j, sb: symbol)$

$s' \leftarrow state()$

$s'.M \leftarrow M$

$s'.M_{ij} \leftarrow sb$.

1.5 print state

1.6 init

$M \leftarrow 0, \in \{-1, 0, 1\}^{nr \times nc}$

$h \leftarrow -1, \in \{-1\} \cup \{0, 1, \cdots, 3^{size} - 1\}$

w: winner $\leftarrow 0 \in \{-1, 0, 1\}$

$end \leftarrow -1 \in \{-1, 0, 1\}$

2. all states S.

2.3 get all states:

csb: current symbol ← 1.

// 1 moves first, -1 second.

s ← state()

S: all states ← dict

S add (s.hash, s)

get all states impl (s, cs, S)

2.4 get all states impl (s, cs, S)

∀ i ∈ [1, nr],

∀ j ∈ [1, nc],

if s.M_ij = 0

then s' ← s.next state (i, j, cs)

h' ← s'.hash

if h' ∉ S

end' ← s' is end

S add (h', s')

if end' = 0

get all states impl (s', -cs, S)

3. Judge (Player p1, p2).

3.2 init:

p1 ← p1, p2 ← p2.

cp: current player ← 0 ∈ {-1, 0, 1}

~~p1.sb~~: p1 symbol ← 1

~~p2.sb~~: ← -1

p1 set symbol (~~p1.sb~~)

p2 set symbol (~~p2.sb~~)

3.3 reset:

p1 reset

p2 reset

3.4 alternate:

loop:

yield p1

yield p2.

3.5 play (is print state ← 0)

alt ← alternate

reset

s ← State

p1 set state s

p2 set state s

if is print state

s print

loop:

player p ← next (alt)

```
i, j, sb ← p act
s' ← s.next state(i, j, sb)
h' ← s' hash
s ← S(h') // get s from S.
  // not use s', ';' s has some
 // values computed already.
p1 set state s.
p2 set state s.
if is print state
    s print
if s is end
    return s.winner.
```

~~end loop~~.

4. AI player

4.2 init(α: step size ← 0.1,
ε ← 0.1)

V: estimations ← dict

α ← α, ε ← ε

Sequence St ← ϕ

Seq greedy g ← ϕ

symbol sb ← 0

4.4 reset

St ← ϕ

g ← ϕ

4.5 set state(s)

St add s,

g add 1.

4.6 set symbol (sb)

// compute V(s) initially.

sb ← sb.

∀ h ← S

s ← S(h)

if s.end

if s.winner = sb

V(h) ← 1

else if s.winner = 0

V(h) ← 0.5

else

V(h) ← 0

else

V(h) ← 0.5

4.7 backup:

states hash hs ← {s.hash | s∈St}

// h⟺s

∀ i ∈ {|hs|-1, ..., 2, 1}

s ← hs(i), s' ← hs(i+1)

td error δ ← g(i)(V(s')-V(s))

V(s) ← V(s) + αδ

4.8 act

// choose a based on s.

s ← St(-1)

as : next action set ← ϕ

ss : next state set ← ϕ

∀ î ∈ [1, nr]

∀ j ∈ [1, nc]

if s.M(i,j) = 0

as add (i,j)

s' ← s.next state(i,j,sb)

ss add (s', hash).

ran ← rand()

if ran < ε

choose i from [1, |as|] randomly.

a ← as(i), ~~add~~ g(-1) ← 0

return (a, sb)

else

~~Vt = {V(s) | s∈ss}~~

i ← argmax Vt(i) randomly

// if ∃ multiple i, choose

// randomly.

a ← as(i)

return (a, sb)

4.9 save policy.

4.10 load policy

5. human player

5.2 init

sb ← 0

keys ← {'q', 'w', 'e', ...}

state s ← state

5.3 reset

pass

5.4 set state (s)

s ← s

5.5 set symbol (sb)

sb ← sb

~~5.6 act~~ :

~~s.play~~

5. to act

key ← input()

n ← keys.index(key)

i ← (n-1)/nc + 1.

j ← (n-1) % nc + 1.

return (i, j, sb).

6. train (epochs, print every n ← 500)

p1 ← AI player (ε ← 0.01)

p2 ← AI player (ε ← 0.01)

j ← judger (p1, p2)

p1 win ← 0.0

p2 win ← 0.0

∀ i ← [1, epochs]

winner w ← j play (i3 print ← 0)

if w = 1

p1 win ++

if w = -1

p2 win ++

if i % print every n = 0

print (i, p1win/i, p2win/i)

p1 backup, p2 backup, j reset.

end ∀

p1 save policy, p2 save policy.

7. compete (turns)

p1 ← AI player (ε ← 0)

p2 ← AI player (ε ← 0)

j ← judger (p1, p2)

p1 load policy, p2 load policy.

p1 win ← 0, p2 win ← 0

∀ i ← [1, turns]

w ← j play

if w = 1, p1 win++.

if w = -1, p2 win++.

j reset.

end ∀.

print (turns, p1 win/turns, p2 win/turns)

8 play (human symbol ← 1)

∈ {1, -1}

loop:

if hsb = 1, p1 ← human player

p2 ← AI player

else p1 ← AI player, p2 ← human player.

p1 load policy, p2 load policy.

j ← judger (p1, p2)

w ← j play.

if w = hsb print (you win)

else if w = -hsb print (You lose)

else print (tie).

9. all process:

S ← get all states

train (10^5)

compete (10^3)

play

1. Bandit

1.3 init($k \leftarrow 10$, $\varepsilon \leftarrow 0$, initial = 0,
$\alpha = 0.1$, is_using_sample_average $\leftarrow 0$,
$c \leftarrow 0$, is_using_gradient $\leftarrow 0$,
is_using_gradient_baseline $\leftarrow 0$,
true_reward $\leftarrow 0$)
As $\leftarrow$ As.letters $\in N^k$
time $t \leftarrow 0$
$\bar{R} \leftarrow 0$.

1.4 reset
Vec$R$ $q^* \sim N(tr, 1) \in R^k$
$Q \leftarrow$ initial, $\in R^k$
$N \leftarrow 0$, $\in N^k$
$A^* \leftarrow \underset{a}{\text{argmax}}\, q^*(a)$
$t \leftarrow 0$

1.5 act
① if rand() < ε
return $A \in$ As randomly.
// (As is vector, but here is seen as a set)
② if $c \neq 0$, (i.e., $c > 0$)
if $t < k$, if $N(a) = 0$. return $A \leftarrow a$, $\forall a$.
else $Q\_UCB(a) \leftarrow Q(a) + c\sqrt{\frac{\log t}{N(a)}}$, $\forall a$
return $A \leftarrow \underset{a}{\text{argmax}}\, Q\_UCB(a)$ randomly.
③ if is_using_gradient
$H \leftarrow Q$
$e^H \leftarrow e^H$
$\pi \leftarrow \frac{e^H}{\sum e^H} \in R^k$
return $A \leftarrow$ draw from As by π.
④ $A \leftarrow \underset{a}{\text{argmax}}\, Q(a)$ randomly.
return A.

1.6. step(action: A)
$R \leftarrow N(q^*(A), 1)$,
$t \leftarrow t+1$, $N(A) \leftarrow N(A)+1$.
$\bar{R} \leftarrow \bar{R} + \frac{1}{t}(R - \bar{R})$
if is_sample_average
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}(R - Q(A))$
else if is_using_gradient
onehot $\leftarrow 0$, $\in N^k$
onehot$(A) \leftarrow 1$.
if is_u.g.-b., baseline $\leftarrow \bar{R}$
else baseline $\leftarrow 0$.
$H \leftarrow H + \alpha(R - \text{baseline})(\text{onehot} - \pi)$
$Q \leftarrow H$
else $Q(A) \leftarrow Q(A) + \alpha(R - Q(A))$
return R.

2. simulate(runs←2000, time←1000, bandits)

rewards rs ← 0 ∈ $R^{|bandits| \times runs \times time}$

best action counts bac ← 0 , same shape with rs.

∀ i, bandit ∈ bandits

∀ run ∈ [1, runs],

bandit.reset

∀ t ∈ [1, time]

a ← bandit.act

r ← bandit.step(a)

rs(i, run, t) ← r

if a = bandit.best_action A*

bac(i, run, t) ← 1

end ∀ t.

end ∀ i, run.

mean best action counts mbac ← bac.mean(axis←2)

mean rewards mrs ← rs.mean(axis←2)

return mbac, mrs, ∈ $R^{|bandits| \times time}$.

3. bandit ← Bandit (ε=0, UCB para c←2,

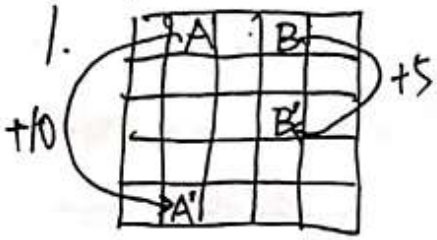
is using_sample_average←1)

average rewards rs ← simulate(2000, 1000, bandit)

plot(rs) : UCB:

Gridworld

1.



1.3 reward $\in \{-1, 0, 5, 10\}$

action $\in$ {left, up, right, down}

1.4 suppose the agent selects all four actions with equal probability in all states.

v_π is the value function for this random policy, for the discounted reward case with $\gamma = 0.9$.

v_π can be computed by solving the system of linear equations,

i.e., Bellman equation for v_π.

$$v_\pi(s) \doteq \mathbb{E}_\pi(G_t \mid S_t = s)$$
$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_\pi(s')), \quad \forall s \in S$$

1.5 Let v_* be optimal value function.

We can solve the Bellman optimality equation for v_*.

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

2. $X_m \leftarrow 5$, $Y_m \leftarrow 5$

Pos_A $\leftarrow (2,5)$

Pos_A' $\leftarrow (2,1)$

Pos_B $\leftarrow (4,5)$

Pos_B' $= (4,3)$

$R_A = 10$, $R_B = 5$

$\gamma = 0.9$

$A = \{(-1,0), (0,1), (1,0), (0,-1)\}$

Action_Pr $\leftarrow 0.25$

2.3 step(states, action a)

if s = Pos_A

return Pos_A', R_A.

if s = Pos_B

return Pos_B', R_B

$s' \leftarrow s + a$

$x, y \leftarrow s'$

if $x < 1$ or $x > X_m$ or $y < 1$ or $y > Y_m$

reward $r \leftarrow -1$

$s' \leftarrow s$

else $r \leftarrow 0$

return s', r.

3. figure 3_2()

Mat $v \leftarrow 0$, $\in R^{X_m \times Y_m}$

Mat $v' \leftarrow v$

loop:

$v' \leftarrow 0$

$\forall (i,j) \in [1, X_m] \times [1, Y_m]$,

$s \leftarrow (i,j)$

$\forall a \in A$

$s', r \leftarrow step(s, a)$

$\pi_{a|s} \leftarrow Action\_Pr$

$v'(s) += \pi_{a|s} \cdot (r + \gamma v(s'))$

// Bellman equation for v_π

end $\forall a$

end $\forall i, j$.

if $\|v - v'\|_1 < 10^{-4}$

draw v.

stop

$v \leftarrow v'$

4. figure 3_5()

Mat $v \leftarrow 0(X_m, Y_m)$

Mat $v' \leftarrow v$

$vs \leftarrow 0 \in R^{|A|}$

loop:

$v' \leftarrow 0$

$\forall s = (i,j) \in [1, X_m] \times [1, Y_m]$

$vs \leftarrow 0$

$\forall a \in A$

$s', r \leftarrow step(s, a)$

$vs(a) \leftarrow r + \gamma v(s')$

end $\forall a$

$v'(s) \leftarrow \max_a (vs(a))$

// value iteration.

end $\forall s$.

if $\|v - v'\|_1 < 10^{-4}$

draw v

stop

$v \leftarrow v'$

5. draw v:

$v\_draw \leftarrow v$.

v_draw. rotate 90° clockwise (axis)

v_draw. show.

Gambler's problem

1.

The problem can be formulated as an undiscounted, episodic, finite MDP.

$s \in S = \{1, 2, \cdots, 99\} \cup \{0, 100\}$

$s=0$ and $s=100$ are two dummy states corresponding to termination, giving them values of 0 and 1 respectively.

$a \in A(s) = \{0, 1, \cdots, \min(s, 100-s)\}$

$r \in \{0, 1\}$.

$S^+ = \{1, 2, \cdots, 99\}$,

$S^+ \cup \{0, 100\} = S$.

1.3 the state-value function $V(s), s \in S$ gives the probability of winning from each state.

The optimal policy maximizes the probability of reaching the goal.

1.4.

Let p_h denote the probability of the coin coming up heads. If p_h is known, the the entire problem is known and it can be solved, for instance, by value iteration.

1.5 for the case of $p_h = 0.4$, there is a whole family of optimal policies, all corresponding to ties for the argmax action selection with respect to the optimal value function.

2. Goal=100,

$S = \{0, 1, \cdots, Goal\}$,

$S^+ = \{1, 2, \cdots, Goal-1\}$

$p_h = 0.4$

3. figure_4_3()

Map $V: S \to \mathbb{R}$,

$V(S) \leftarrow 0$, $V(Goal) = 1$

sweeps_history $\leftarrow \emptyset$

loop: // value iteration

sweeps_history.add(V)

$\Delta \leftarrow 0$

$\forall s \in S^+$

$v \leftarrow V(s)$

Vector A $\leftarrow A(s) = \{0, \cdots, \min(s, Goal-s)\}$

Vector vs $\leftarrow 0, \in \mathbb{R}^{|A|}$

$\forall i_a \in |A|$, (here we mean $i_a \in \{1, \cdots, |A|\}$)

$a \leftarrow A(i_a)$

$vs(i_a) \leftarrow p_h \cdot V(s+a) + (1-p_h) V(s-a)$

end $\forall i_a$

$V(s) \leftarrow \max_{i_a}(vs)$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

end $\forall s$.

if $\Delta < 10^{-9}$: sweepshistory add V. break loop.

end loop.

figure:

$\forall V \in$ sweeps_history

plot V.

xlabel (Capital)

ylabel (value estimations)

plot (π)

xlabel (Capital)

ylabel (final policy)

// compute optimal policy.

Map $\pi^* : S \to N$,

$\pi(S) \leftarrow 0$.

$\forall s \in S^+$

Vec $A \leftarrow A(s) = \{0, 1, \dots, \min(s, Goal-s)\}$

Vec $vs \leftarrow 0_, \in R^{|A|}$

$\forall i_a \in |A|$

$a \leftarrow A(i_a)$

$vs(i_a) \leftarrow p_h \cdot V(s+a) + (1-p_h) \cdot V(s-a)$

end $\forall i_a$

$\pi(s) \leftarrow A(\operatorname{argmax}_{i_a}(vs(2:)) + 1)$

// avoid choosing $a=0$ which doesn't change state.

end $\forall s$.

Chapter 4

1. Value iteration

$V = 0 \in \mathbb{R}^{|S|}$

loop:

$\Delta \leftarrow 0$

$\forall s \in S$

$v \leftarrow V(s)$.

$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma V(s'))$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

end $\forall s$.

if $\Delta < \theta$ (a small positive number)

stop loop

end loop

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

2. correction: Gambler's problem

$S = \{1, 2, \cdots, 99\}$

$S^+ = S \cup \{0, 100\}$

i.e. $S \leftrightarrow S^+$

3. Policy iteration (using iterative policy evaluation).

1) $V(s) \in \mathbb{R}$, $\pi(s) \in A(s)$, $\forall s \in S$. go to 2).

initialize V, π arbitrarily.

2) policy evaluation

loop:

$\Delta \leftarrow 0$

$\forall s \in S$

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))(r + \gamma V(s'))$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

end $\forall s$

if $\Delta < \theta$, stop loop

3) policy improvement

policy stable $\leftarrow$ 1

$\forall s \in S$

old $a \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)(r + \gamma V(s'))$

if old $a \neq \pi(s)$: policy stable $\leftarrow 0$

end $\forall s$.

if policy stable

stop and return $V \approx v_*$, $\pi \approx \pi_*$.

Car rental

1. max cars at 1: $M_1=20$
   max cars at 2: $M_2=20$.
   max move: $M_m=5$
   expected rental request at i: λ_{i1}
   expected ~~rental~~ returns at i: λ_{i2}
   $i=1,2$.
   $\therefore \begin{pmatrix}\lambda_{11} & \lambda_{12}\\ \lambda_{21} & \lambda_{22}\end{pmatrix}=\begin{pmatrix}3 & 3\\ 4 & 2\end{pmatrix}$
   discount $\gamma=0.9$
   rent reward $R_r=R_{rent}=10$,
   move cost $c_m=2$.
   $A=\{-M_m,\cdots,M_m\}$
   Poisson upper bound $M_p=11$
   poisson cache: dict

2. poisson_pr(n,λ) // $\lambda<10$
   key$=n\cdot 10+\lambda+1$
   if key $\notin$ poisson cache pc
   pc(key) $\leftarrow$ poisson.pmf(n,λ)
   return pc(key)

2.1 poisson.pmf:
   $f(n,\lambda)=e^{-\lambda}\cdot\lambda^n/n!$
   // $e^{\lambda}=\sum_{n=0}^{\infty}\frac{\lambda^n}{n!}$

3. Bellman equation for q_π
   Bellman_for_q-s,a:
   input: s, a, V, is constant returned cars
   output:
   $q(s,a)\leftarrow E_a\leftarrow\sum_{s',r}p(s',r|s,a)(r+\gamma V(s'))$
   process:
   expected return: $E_a\leftarrow 0$
   $E_a\leftarrow E_a-|a|\cdot c_m$
   number of cars at i: n_i
   $n_1\leftarrow\min(s_1-a,M_1)$
   $n_2\leftarrow\min(s_2+a,M_2)$
   $\forall$ rental request $req_1\in\{0,\cdots,M_p\}$
   $\forall req_2\in\{0,\cdots,M_p\}$
   $p\leftarrow$ poisson_pr(req_1,λ_{11})
   $\cdot$ poisson_pr(req_2,λ_{21})
   valid rental at i: n_{iv}
   $n_1'\leftarrow n_1$, $n_2'\leftarrow n_2$
   $\therefore n_{1v}\leftarrow\min(n_1',req_1)$
   $n_{2v}\leftarrow\min(n_2',req_2)$
   $r\leftarrow(n_{1v}+n_{2v})R_r$
   $n_1'\leftarrow n_1'-n_{1v}$, $n_2'\leftarrow n_2'-n_{2v}$

if is constant returned cars

returned cars $ret_1 \leftarrow \lambda_{12}$

$ret_2 \leftarrow \lambda_{22}$

$n_1' \leftarrow \min(n_1' + ret_1, M_1)$

$n_2' \leftarrow \min(n_2' + ret_2, M_2)$

$s' \leftarrow (n_1', n_2')$

$E_{a_1} \leftarrow E_{a_1} + p(r + \gamma V(s'))$

else

// $E_{a_1} \leftarrow E_{a_1} + p(r + \gamma \sum_{s'} p' V(s'))$

$E_{a_1}' \leftarrow 0$

$\forall ret_1, ret_2 \in \{0, \cdots, M_p\}$

$p' \leftarrow poisson\_pr(ret_1, \lambda_{12}) \cdot poisson\_pr(ret_2, \lambda_{22})$

$n_1' \leftarrow \min(n_1' + ret_1, M_1)$

$n_2' \leftarrow \min(n_2' + ret_2, M_2)$

$s' \leftarrow (n_1', n_2')$

$E_{a_1}' \leftarrow E_{a_1}' + p' V(s')$

end $\forall ret_1, ret_2$.

$\therefore E_{a_1} \leftarrow E_{a_1} + p(r + \gamma E_{a_1}')$

end $\forall req_1, req_2$.

return E_{a_1}.

Map $V: S \to R$, $V(S) \leftarrow 0$

Map $\pi: S \to A \subseteq Z$, $\pi(S) \leftarrow 0$

iteration $k \leftarrow 0$

loop:

draw policy π

loop: (policy evaluation (in-place)

$\Delta \leftarrow 0$

$\forall s \in S$, $v \leftarrow V(s)$,

$V(s) \leftarrow$ Bellman_for_q-sa

$(s, \pi(s), V,$ is constant rn)

// $V(s) \leftarrow q(s, \pi(s))$

$\Delta \leftarrow \Delta \vee |v - V(s)|$

end $\forall s$.

if $\Delta < 10^{-4}$: stop loop.

end loop. // inner loop.

is policy stable $\leftarrow 1$.

$\forall s \in S$., $a\_old \leftarrow \pi(s)$

$qsa \in R^{|A|}$

$\forall ia \in \{1, \cdots, |A|\}$

$a \leftarrow A_{ia}$

4. figure 4.2 (is constant returned cars $\leftarrow 1$)

if $a \in [-s_2, 0] \cup [0, s_1]$

qsa(i_a) ← Bellman_for_q-s-a(s, a, V, is constant...)

else

qsa(i_a) ← $-\infty$

end $\forall i_a$.

$\pi(s) \leftarrow A(\text{argmax}(qsa))$

if is policy stable and a_old $\neq \pi(s)$

is policy stable ← 0

end $\forall s$.

if is policy stable

draw V, stop

k ← k+1

5. We formulate this as a continuing finite MDP.

Car rental (synchronous)

1. $M_1=20$, $M_2=20$, $c_m=2$.
additional park cost $c_{park}=4$
$R_r=10$, rent reward.
$\begin{pmatrix}\lambda_{11} & \lambda_{12}\\ \lambda_{21} & \lambda_{22}\end{pmatrix}=\begin{pmatrix}3,3\\ 4,2\end{pmatrix}$
$S \leftarrow \{0,\cdots,M_1\}\times\{0,\cdots,M_2\}$
poisson cache pc: dict

2. poisson (n,λ)
key $\leftarrow i\cdot 10+\lambda$
if key $\notin$ pc
$pc(key) \leftarrow e^{-\lambda}\cdot\lambda^n/n!$
return pc(key)

3. class: policy iteration

3.2 init (is solve 4-5 $\leftarrow$ 0)
$M_p \leftarrow 8$
$\delta_M \leftarrow 10^{-1}$
$\gamma \leftarrow 0.9$
$A \leftarrow \{-M_2,\cdots,0\}\cup\{0,\cdots,M_1\}$
Map: $V: S\to R$, $V(S)\leftarrow 0$
Map $\pi: S\to A\subset Z$
is solve 4-5 $\leftarrow$ is solve 4-5

3.3 solve:
iterations $k\leftarrow 0$
loop:
$V \leftarrow$ policy evaluation (V,π)
is policy change, $\pi \leftarrow$
policy improvement (A,V,π)
if is policy change $=0$
stop
$k \leftarrow k+1$.

3.4 (out-place)
policy evaluation (V,π)
process:
$V_{new}: S\to R$
loop:
$V_{new}(s) \leftarrow v_{\pi}\_s\_pe(\pi,V,s), \forall s\in S$.
if $\|V_{new}-V\|_1<\delta_M$
$V\leftarrow V_{new}$, return V.
$V \leftarrow V_{new}$

3.5. policy improvement (A,V,π)
$\pi_{new}: S\to A$.
expected action returns $qsa: S\times A\to R$

$qsa(s,a) \leftarrow v \leftarrow q\_\pi\_s\_a\_pi(V,s,a)$

$\forall a \in A, s \in S.$

$\pi\_new(s) \leftarrow \underset{a}{\text{argmax}}(qsa(s,:)), \forall s \in S.$

if $\pi\_new \neq \pi$

$\pi \leftarrow \pi\_new$, return 1

return 0 // is stable, is changed = 0.

3.6. // $O(n^4)$ for all req, ret.

Bellman(V, s, a) for q(s,a)

process:

$E_a \leftarrow 0$

if is solve 4.5

if $a > 0$

// free shuttle to loc 2.

$E_a \leftarrow E_a - (a-1)c_m$

else

$E_a \leftarrow E_a - |a| c_m$

else

$E_a \leftarrow E_a - |a| c_m$

$n_1 \leftarrow (s_1 - a) \wedge M_1$

$n_2 \leftarrow (s_2 + a) \wedge M_2$

$\forall req_1, req_2 \in \{0, \cdots, M_p\}$

$p \leftarrow poisson(req_1, r_1) \cdot poisson(req_2, r_2)$

$n_1' \leftarrow n_1$, $n_2' \leftarrow n_2$

$n_{1v} \leftarrow n_1' \wedge req_1$, $n_{2v} \leftarrow n_2' \wedge req_2$

$r \leftarrow (n_{1v} + n_{2v}) R_r$

if is solve 4.5

if $n_1' \geq 10$

$r \leftarrow r - c_{park}$

if $n_2' \geq 10$

$r \leftarrow r - c_{park}$

$n_1' \leftarrow n_1' - n_{1v}$, $n_2' \leftarrow n_2' - n_{2v}$

$E_a' \leftarrow 0$

$\forall ret_1, ret_2 \in \{0, \cdots, M_p\}$

$p' \leftarrow poisson(ret_1, \lambda_{r2}) \cdot poisson(ret_2, \lambda_{r2})$

$n_1' \leftarrow (n_1' + ret_1) \wedge M_1$,

$n_2' \leftarrow (n_2' + ret_2) \wedge M_2$

// classical Bellman equation

// for V(s)

$E_a' \leftarrow E_a' + p' V(s')$,

here $s' = (n_1', n_2')$

end $\forall ret_1, ret_2$.

$\therefore E_a \leftarrow E_a + p(r + \gamma E_a')$

end $\forall req_1, req_2$.

return E_a.

3.7 v_π_sp(π, V, s) // $v_\pi(s)$

$a \leftarrow \pi(s)$

$E_a \leftarrow$ Bellman(V, s, a)

return E_a.

3.8 q_π_sapi(V, s, a) // $q_\pi(s,a)$

if $a \in [-s_2, 0] \cup [0, s_1]$

$E_a \leftarrow$ Bellman(V, s, a)

else

$E_a \leftarrow -\infty$

return E_a.

4. test:

solver ← Policy iteration (is solve 4-5 ← 1)

solver. solve

plot π.

grid world

1. $X_m \leftarrow 4$, $Y_m \leftarrow 4$

$A = \{(0,-1), (-1,0), (0,1), (1,0)\}$

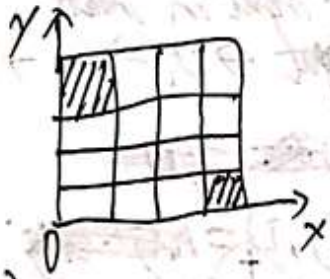
2. is terminal(s)

if $s = (1, Y_m)$ or $s = (X_m, 1)$

return 1

else

return 0.



3. step(s, a)

if is terminal(s)

return s, 0.

$s' \leftarrow s + a$

if $s'_1 < 1$ or $s'_1 > X_m$ or $s'_2 < 1$ or $s'_2 > Y_m$

$s' \leftarrow s$

$r \leftarrow -1$

return s', r

4. compute V (in place $\leftarrow$ 1, $\gamma \leftarrow$ 1)

Map $V: S \rightarrow R$

$k \leftarrow 0$.

if in place

$V_{old} \leftarrow V$

denote V as V_{new}

else

$V_{new} \leftarrow V$

denote V as V_{old}

loop:

if in place

$V_{old} \leftarrow V$

$\forall s \in S$,

$v \leftarrow 0$

$\forall a \in A$,

$s', r \leftarrow$ step(s, a)

$v \leftarrow v + \underbrace{\pi(s)}_{=0.25}(r + \gamma \cdot V(s'))$

end $\forall a$.

$V_{new}(s) \leftarrow v$

end $\forall s$.

$\Delta \leftarrow \max(|V_{old} - V_{new}|)$

if not in place

$V \leftarrow V_{new}$

if $\Delta < 10^{-4}$: stop loop

$k \leftarrow k + 1$, end loop.

return V, k.

Blackjack

1. $A = \{hit, stick\}$, $\{a_h, a_s\}$,

$a_h \leftarrow 1$, $a_s = 2$

2. $\pi_P : N \to A$

$VecN(2)$, $\pi_P : \{1, \cdots, 11\} \to a_h$.

$\{12, \cdots, 19\} \to a_h$

$\{20, 21\} \to a_s$

3. $\pi_p : S \to A$

usable_ace_player ua_p,

player sum s_p

dealer card c_d

$\leftarrow s$

return $\pi_P(s_p)$

4. $S \to A$

if binomial$(1, 0.5) = 1$

return a_s

else return a_h.

5. $\pi_d : N \to A$

$VecN(2)$ $\pi_d : \{1, \cdots, 11\} \leftarrow a_h$

$\pi_d : \{12, \cdots, 16\} \to a_h$

$\{17, \cdots, 21\} \to a_s$.

6. get_card( )

card ~ uniform (1, 13)

card = card ∧ 10

return card.

7. card_value (card)

if card = 1 return 11

else return card.

8. play (π, initial_s, initial_a)

$s_0 \leftarrow$ none $a_0 \leftarrow$ none

// π is policy for player.

sum of player $sum_p = 0$

trajectory of player $traj_p \leftarrow$ none

$ua_p \leftarrow 0$ or 1

$c_{1d} = 0$, $c_{2d} = 0$

$ua_d = 0$

if s_0 is none:

while $sum_p < 12$:

card ← get card

sum_p ← sum_p + card_value(card)

if card = 1

if $sum_p > 21$

$sum_p = sum_p - 10$

else

$ua_p = 1$

end while $s_p < 12$

c_{1d} ← get card

c_{2d} ← get card

else // s_0 is not none.

ua_p, sum_p, c_{1d} ← s_0

c_{2d} ← get card

assert $sum_p \leq 21$

sum_d ← card_value(c_{1d}) + card_value(c_{2d})

if $sum_d > 21$

assert $sum_d = 22$. (∵ dealer only has 2 cards)

$sum_d = sum_d - 10$.

// now we have $sum_d \in \{1, 11\}$

$sum_p \in \{12, 11, 21\}$

// game starts!

// player's turn.

loop: $s \leftarrow (ua_p, sum_p, c_{1d})$

if inital action $a_0 \neq$ none

initial action $a_0 \leftarrow$ none

else

$a \leftarrow \pi(ua_p, sum_p, c_{1d})$

$traj_p$.add(s, a)

if $a = a_s$: break loop

card ← get card

if card = 1

count_ace ← count_ace + 1

sum_p ← sum_p + card_value(card)

$sum_p \leftarrow sum_p + card\ value(card)$

if card = 1

if $ua_p = 1$

$sum_p = sum_p - 10$ (we can't have ~~two usable aces~~ more than one.)

else

$ua_p = 1$

if $sum_p > 21$

if $ua_p = 1$

$sum_p = sum_p - 10$

$ua_p = 0$

if $sum_p > 21$

return $(s_0, -1, traj_p)$ return s_0

end loop (player)

// dealer's turn.

loop:

$a \leftarrow \pi_d(sum_d)$

if $a = a_s$: break loop

card $\leftarrow$ get card(), $sum_d \leftarrow sum_d + card\ value(card)$

if card = 1

if $ua_d = 1$: $sum_d \leftarrow sum_d - 10$

else: $ua_d = 1$

if $sum_d > 21$

if $ua_d = 1$

$sum_d \leftarrow sum_d - 10$

$ua_d = 0$

if $sum_d > 21$

return $(s_0, 1, traj_p)$

end loop (dealer)

// compare sum_p and sum_d.

(here we must have $sum_p \leq 21$, $sum_d \leq 21$)

if $sum_p > sum_d$

$R \leftarrow 1$

return $s_0, 1, traj_p$.

else if $sum_p = sum_d$

$R \leftarrow 0$

return $s_0, 0, traj_p$

else $R \leftarrow -1$

return $s_0, R, traj_p$.

~~on-policy~~ ~~off-policy~~

breaking ties 2)

random (see book)

$\pi(s) \leftarrow \underset{a}{\text{argmax}}\, Q(s,a)$

end $\forall (s,a)$.

end $\forall i$.

return Q, π.

9. ~~MC sample with on-policy~~

MC_policy_evaluation ~~(...)~~ (episodes)

~~$V(s) \leftarrow 0$, $n(s) \leftarrow 0$, $\forall s$~~

$\forall i \in \{1, \dots, \text{episodes}\}$

$\_, \_, traj_p \leftarrow play(\pi_p, none, none)$

$\forall (s,a) \in traj_p$

~~$sum_p, sum_p, C_d \leftarrow 0$~~

$n(s) \leftarrow n(s)+1$

$V(s) \leftarrow V(s) + \frac{1}{n(s)}(G - V(s))$

return V

10. // MC with exploring starts.

MC_es(episodes)

$Q(s,a) \leftarrow 0$.

$\pi \leftarrow \pi_p$

$n(s,a) \leftarrow 0$

$\forall i \in \{1, \dots, \text{episodes}\}$

$s_0 \leftarrow$ random_choice(S)

$a_0 \leftarrow$ random_choice(A)

$G, traj_p \leftarrow play(\pi, s_0, a_0)$

$\forall (s,a) \in traj_p$

$n(s,a) \leftarrow n(s,a)+1$

$Q(s,a) \leftarrow Q(s,a) + \frac{1}{n(s,a)}(G - Q(s,a))$

return Q, π

~~control~~

~~MC_off_policy (episodes)~~

MC_estimate_state_value

_off_policy (episodes)

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

// $V_{n+1}(s) = V_n(s) + \frac{1}{n}(W_n G_n - V_n(s))$

$s_0 = (sum_p = 1, sum_p = 13, C_d = 2)$

~~$\forall n \leftarrow 0$~~, $V \leftarrow 0$, $V_W \leftarrow 0$, $C \leftarrow 0$, OS, WS.

~~$\forall i$~~ $\forall i \in \{1, \dots, \text{episodes}\}$

$\_, G, traj_p \leftarrow play(\mu_p, s_0, none)$

$W \leftarrow 1$

$\forall (s,a) \in traj_p$.

if $a = \pi_p(s)$

$W \leftarrow W \cdot \frac{1}{\mu_p(a|s)}$, here $= 0.5$

else

$W \leftarrow 0$, break $\forall (s,a)$

end $\forall (s,a)$.

~~$V_0 \leftarrow V_0$~~ $n \leftarrow n+1$, $C \leftarrow C + W$

$V_0 \leftarrow V_0 + \frac{1}{n}(WG - V_0)$

$V_W \leftarrow V_W + \frac{W}{C}(G - V_W)$

$OS(i) \leftarrow V_0$, $WS(i) \leftarrow V_W$, end $\forall i$

return OS, WS

4.

12. figure5_1()

$V_1 \leftarrow$ MC_policy_evaluation_on_policy(10^4)

$V_2 \leftarrow$ MC_policy_evaluation_on_policy(5×10^5)

$V_1(:,:,ua_p=1).log()$

$V_1(:,:,ua_p=0).log()$

$V_2(:,:,ua_p=1).log()$

$V_2(:,:,ua_p=0).log()$

13. figure5_2()

$Q_* \leftarrow$ MC_es(5×10^5)

$V_*(s) = \max_a Q_*(s,a)$, $\forall s$

$g(x_1,x_2) = \max_{x_3} f(x_1,x_2,x_3)$

$g_1(x_1) = \max_{x_3} f(x_1,1,x_3)$

$g_2(x_1) = \max_{x_3} f(x_1,2,x_3)$

$V_*(:,:,ua_p=1).log("optimal value")$

$V_*(:,:,ua_p=0).log("optimal value")$ without usable Ace.

$\pi_*(:,:,ua_p=1).log("optimal policy")$

$\pi_*(:,:,ua_p=0).log("optimal policy")$ without usable Ace.

14. figure5_3()

true_value = -0.27726

episodes = 10^4

runs = 100

$e_0, e_w \leftarrow 0$ (episodes), $\forall i \leftarrow \{1,\cdots,runs\}$

$OS, WS \leftarrow$ MC_estimate_state_value_off_policy(episodes)

$e_0 \leftarrow e_0 + (OS - true\_value)^2$ // vector

$e_w \leftarrow e_w + (WS - true\_value)^2$

end $\forall i$

$e_0 \leftarrow e_0/runs$, $e_w \leftarrow e_w/runs$

plot(e_0), plot(e_w)

title(ordinary importance sampling)

title(weight importance sampling)

π is deterministic policy

$v_\pi(s)$

v_π, q_π, π, q_*, $q_\pi(s,a)$

1. First-visit MC policy evaluation

π, V, returns(s), ∀s ∈ S.

Repeat:

generate an episode using π

∀s in the episode

G ← return following the first occurrence of s

n(s) ← n(s)+1

V(s) ← V(s) + $\frac{1}{n(s)}$(G − V(s))

sum G(s) ← sum G(s) + G

n(s) ← n(s) + 1

$V(s) \leftarrow \frac{\text{sum } G(s)}{n(s)}$

should use incremental implementation

E G ←

3.2 the backup diagram for MC estimation of v_π.

4. Without a model, state values alone are not sufficient to determine a policy.

2. Blackjack: 2.2 playing blackjack is naturally formulated as an episodic finite MDP

2.3 consider the policy that sticks if the player's sum is 20 or 21, and otherwise hits.

2.4 to find v for this policy by a MC approach.

2.5 there is no difference between first-visit and every-visit MC methods.

2.6 the terminal reward is also the return.

3. entire trajectory of transitions along a particular single episode.

5 $\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \rightarrow \cdots \xrightarrow{I} \pi_* \xrightarrow{E} q_*$

6. ~~MC control~~:

MC ES (MC control)

Initialize $Q(s,a) \leftarrow$ arbitrary

$\pi(s) \leftarrow$ arbitrary

~~sum(s,a)←0~~, $n(s,a) \leftarrow 0$

~~returns(s,a) ← empty list~~

$\forall s \in S$, $a \in A(s)$

repeat:

① choose $s_0 \in S$, $a_0 \in A(s_0)$, s.t. all pairs have probability >0

② generate an episode starting from s_0, a_0, following π.

③ ~~∀ s,a~~ pairs s,a in the episode

$G \leftarrow$ return following the first occurrence of s, a.

$n(s,a) \leftarrow n(s,a)+1$

$Q(s,a) \leftarrow Q(s,a) +$ ~~sum(s,a) ← sum(s,a) + G~~ (use incremental implementation) ~~No usable ace:~~

$\frac{1}{n(s,a)}(G - Q(s,a))$ ~~n(s,a) ← n(s,a)+1~~

$Q(s,a) \leftarrow \frac{sum(s,a)}{n(s,a)}$

④ $\pi(s) \leftarrow \underset{a}{\text{argmax}}\, Q(s,a)$, $\forall s$ in the episode

7. solving Blackjack

7.2 apple MC ES to blackjack.

7.3 "the episodes are all simulated games, ~~;;~~ it is easy to arrange for exploring starts that include all possibilities

7.4 pick s_0 at random with equal probability.

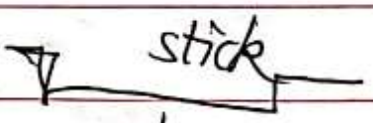
7.5 initial policy is the one in 2.

the initial $Q(s,a) = 0$ $\forall s, a$.

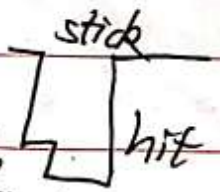
7.6 V is computed from Q found by one ES.

7.7 π_*:

usable ace:



No usable ace:



7.8 The policy is the same as the basic strategy of Thorp with the exception of the leftmost notch

$n(s,a) \leftarrow n(s,a)+1$

$Q(s,a) \leftarrow Q(s,a) + \frac{1}{n(s,a)}(G - Q(s,a))$

in the policy for a usable ace, which is not present in Thorp's strategy. We are uncertain of the reason for this discrepancy.

8. among ε-soft policies, ε-greedy policies are in some sense those that are closest to greedy.

8.3 ; next page I do not require that the policy be taken all the way to a greedy policy, only that it be moved toward a greedy 8.3 next page policy.

10. on-policy first-visit MC control (for ε-soft policies)

$\pi(a|s) \leftarrow$ arbitrary.

~~Returns(s,a) $\leftarrow$ empty list~~ $n(s,a) \leftarrow 0$

$\pi(a|s) \leftarrow$ an arbitrary ~~soft~~ ε-soft policy

Repeat:

① generate an episode using π

② ∀ pair s,a in the episode

$G \leftarrow$ return following the first occurrence of s,a

$n(s,a) \leftarrow n(s,a)+1$

$Q(s,a) \leftarrow Q(s,a) + \frac{1}{n(s,a)}(G - Q(s,a))$

Sum $G(s,a) \leftarrow$ Sum $G(s,a) + G$

$n(s,a) \leftarrow n(s,a) + 1$ ← else incremental implementation

$Q(s,a) \leftarrow \frac{\text{Sum } G(s,a)}{n(s,a)}$

③ ∀s in the episode

$A^* \leftarrow \arg\max_a Q(s,a)$

∀ $a \in A(s)$

$\pi(a|s) \leftarrow \begin{cases} 1-\varepsilon+\frac{\varepsilon}{|A(s)|}, & \text{if } a = A^* \\ \frac{\varepsilon}{|A(s)|}, & \text{if } a \neq A^* \end{cases}$

8.3

ε-soft policies π

$\Leftrightarrow \pi(a|s) \geq \frac{\varepsilon}{|A(s)|}$, $\forall s, a$.

for some $\varepsilon > 0$

8.4 any ε-soft policy π,

any ε-greedy policy with respect to q_π is guaranteed to be better than or equal to π.

3. on-policy first-visit MC control (for ε-soft policies)

init: $\pi(a|s)$, an arbitrary ε-soft policy.

Repeat:

$\pi(a|s) \leftarrow \begin{cases} 1-\varepsilon+\frac{\varepsilon}{|A(s)|} \\ \frac{\varepsilon}{|A(s)|} \end{cases}$

10.3 here $\pi(a|s)$ is ε-soft policy, (not ε-greedy policy.)

If π' is partial ε-greedy pol·cy,

$\Rightarrow q_\pi(s, \pi'(s)) \begin{cases} = v_\pi(s) & \text{if } \pi'(a|s) = \pi(a|s) \\ \geq v_\pi(s) & \text{if } \pi'(a|s) \neq \pi(a|s) \end{cases}$

10.4 In the on-policy method we move it only to an ε-greedy policy.

10.5 off-policy prediction via importance sampling

11.3 In practice, the weighted importance-sampling estimator usually has dramatically lower variance and is strongly preferred.

11.4 $V_t(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{|\mathcal{T}(s)|}$ ordinary importance sampling.

11.5 weighted importance sampling:

$V_t(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)}}$, $V_t(s) = 0$ if $\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} = 0$

11.6. $\rho_t^T \doteq \cdots = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}$

11.7

12. off-policy estimation of a blackjack state value.

12.2. $Q(s,a) \doteq \frac{\sum_{t \in \mathcal{T}(s,a)} \rho_{t+1}^{T(t)} G_t}{\sum_{t \in \mathcal{T}(s,a)} \rho_{t+1}^{T(t)}}$

12.2. We evaluate the state in which the dealer is showing a deuce, the sum of the player's cards is 13, and the player has a usable ace.

12.3 the data is generated by starting in this state then choosing to hit or stick at random with equal probability (the behavior policy). The target policy is to stick only on a sum of 20 or 21.

13 $V_{n+1} = V_n + \frac{1}{n}(G_n - V_n)$

$nV_{n+1} = nV_n + G_n - V_n$

$V_{n+1} = \frac{G_n + (n-1)V_n}{n}$

$V_{n+1} = \frac{\text{sum}_{-1} G_i + G}{n}$

B: $V_2 = \frac{W_1 G_1}{W_1} = G_1$

$V_3 = \frac{W_1 G_1 + W_2 G_2}{W_1 + W_2}$

$V_3 = \frac{G_1 + G_2}{2}$

$\therefore V_{n+1} = \frac{G_1 + \cdots + G_n}{n}$

$\therefore nV_{n+1} = G_1 + \cdots + G_n$

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

$= \frac{nV_n + G_n - V_n}{n}$

$= V_n + \frac{1}{n}(G_n + V_n)$

13.3

$\therefore V_{n+1} = \frac{\sum W_k G_k}{\sum W_k}, \quad n \geq 1$ (1)

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

$= C_{n-1} V_n + W_n G_n, \quad C_0 = 0$

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

$\therefore = V_n + \frac{W_n}{C_n}(G_n - V_n)$

$(n \geq 1)$

here V_1 is arbitrary.

13.4 let $W_n = 1$, $\therefore C_n = n$

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

13.5 $V_{n+1} = \frac{\sum_{k=1}^{n} W_k G_k}{n} \Leftrightarrow V_{n+1} = V_n + \frac{1}{n}(W_n G_n - V_n)$

$Q \to q_\pi$

14 incremental off-policy every-visit MC policy evaluation.

process: $Q(s,a) \leftarrow$ arbitrary, $C(s,a) \leftarrow 0$

$\mu(a|s) \leftarrow$ an arbitrary soft behavior policy

$\pi(a|s) \leftarrow$ an arbitrary target policy

$\forall s \in S, a \in A(s)$

Repeat:

1) generate an episode using μ:

$S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T, S_T$.

2) $G \leftarrow 0$

3) $W \leftarrow 1$

4) $\forall t = T-1, T-2, \cdots, 0$

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

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}(G - Q(S_t, A_t))$

$W \leftarrow W \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$

If $W = 0$: exit for loop.

15. off-policy MC control

15.3 π converges to optimal at all encountered states.

15.4 : off-policy MC control method learns only from the tails of episodes.

∴ If nongreedy actions are common, then learning will be slow, particularly for states appearing in the early portions of long episodes.

15. off-policy every-visit MC control (returns $\pi \approx \pi_*$)

process:

$Q(s,a) \leftarrow$ arbitrary, $C(s,a) \leftarrow 0$.

$\pi(s) \leftarrow$ a deterministic policy that is greedy w.r.t. Q.

loop:

① generate an episode using any soft policy μ:

$S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T, S_T$.

② $G \leftarrow 0$

③ $W \leftarrow 1$

④ $\forall t = T-1, T-2, \cdots, 0$

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

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}(G - Q(S_t, A_t))$

$\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$ (with ties broken consistently)

if $A_t \neq \pi(S_t)$ then exit for loop.

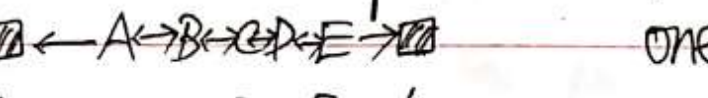
$W \leftarrow W \frac{1}{\mu(A_t|S_t)}$

random walk

true value v_π.

1. $v_\pi(C) = 0.5$

$v_\pi(i) = \frac{i}{6}$, $i = 1, \dots, 5$.



1.3 $V(s) = 0.5$, $\forall s \in S$

1.4 $S = S \cup \{0, 6\}$, $V(0) = 0$, $V(6) = 0$

1.5 All episodes start in C, and proceed either left or right by one state on each step, with equal probability.

2. $v_\pi : S \to R$,

true value

$v_\pi(s) = \frac{s}{6}$, $s = 1, \dots, 5$, $v_\pi(0) = 0$, $v_\pi(6) = 0$

2.3 action left $a_l = 0$

action right $a_r = 1$

3. TD(0)

(V, $\alpha = 0.1$, is batch $= 0$)

state $s = 3$.

trajectory traj add s.

loop:

$x \sim$ binomial(1, 0.5)

if $x = a_l$

$s' = s - 1$

else $s' = s + 1$

if $s' = 6$, $r = 1$

else $r = 0$

$V(s) \leftarrow V(s) + \alpha \cdot (r + V(s') - V(s))$

$s \leftarrow s'$

traj.add(s), rs.add(r)

if $s = 6$ or $s = 0$

break loop

end loop

return traj, rewards.

4. constant-α MC

(V, $\alpha = 0.1$, is batch $= 0$)

$s = 3$

traj add s.

loop:

$x \sim$ binomial(1, 0.5)

if $x = a_1$

$s' = s - 1$

else

$s' = s + 1$

traj add s'

if $s' = 6$

return traj, $G_i = 1$

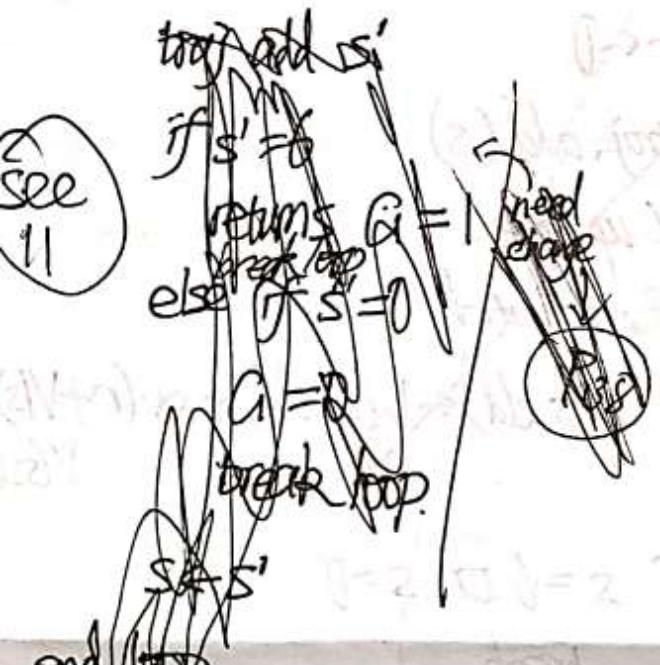
else if $s' = 0$

$G_i = 0$

break loop

$s \leftarrow s'$

end loop.



if is batch = 0

$V(s) \leftarrow V(s) + \alpha \cdot (G_i - V(s))$, $\forall s \in$ traj

return traj, rs.setSize(|traj|)

rs ← G_i.

5. example 6.2 left:

compute state value:

episodes = {0, 1, 10, 100}

$V \leftarrow V_0$

$\forall i \in \{0, \dots, 100\}$

if $i \in$ episodes

plot V, with title: i + "episodes".

TD(0): (V)

end $\forall i$

plot v_π.

6 example 6.2 right

rms_error():

TD(0)_αs = {0.15, 0.1, 0.05}

MC_αs = {0.01, 0.02, 0.03, 0.04}

episodes = 0/

runs = 100, total errors E = 0(episo

$\forall$ i ∈ {0, ..., |αs=TD(0)_αs MC_αs|}

$\alpha \leftarrow$ αs(i)

if i < |TD(0)_αs|

method ← TD

else

method ← MC

$\forall r \in |runs| = \{1, \dots, runs\}$

errors ← R

$V \leftarrow V_0$

$\forall j \in \{0, \dots, episodes\}$

err ← $\sqrt{\sum (V - v_\pi)^2}$ / #S.size(), here=5

errors.add(err)

if method = "TD"

TD(0)(V, α)

else

constMC(V, α)

end $\forall$ i

total_errors ← total_errors + errors.
end ∀ r
total_errors ← total_errors / runs
plot(total_errors, name: method + α)

7. // figure 6.3
batch_updating(method, episodes, α=10^-3)
runs = 100
total_errors E ← 0(episodes)
∀ r ∈ {1, ..., runs}
V ← V0
errors ← R^episodes
trajs ← ∅ rewards rss ← ∅
∀ ep ∈ {1, ..., episodes}
if method = 'TD'
traj, rs ← TD(0)(V, is_batch←1)
else
traj, rs ← constant-α MC(V, is_batch←1)
trajs ← add(traj)
rss ← add(rs)
loop
sum_s ← 0(|S|.size)
∀(traj, rs) in (trajs, rss)
∀ i ∈ {1, ..., |traj| - 1}
if method = 'TD'
V' ← V' + (rs(i) + V(traj(i+1)) - V(traj(i)))
else
V' ← V' + (rs(i) - V(traj(i)))
end ∀ i end ∀(traj, rs)
if ||sum_s||_1 < 10^-3
break loop.
V ← V + α·sum_s
end loop.
err ← sqrt(Σ(V - V*)^2 / |S.size()), here S.
errors.add(err)
end ∀ ep.
E ← E + errors
end ∀ r.
E ← E / runs
return E.

8 example_6_2()

~~rms_errors()~~

compute_state_value()

rms_error()

9. figure_6_3()

episodes = 100

TD_errors ← batch_updating('TD', episodes)

MC_errors ← batch_updating('MC', episodes)

plot(TD_errors)

plot(MC_errors)

11. if s'=6

G←1

else if s'=0

G=0

~~else G=0~~

s←s'

traj.add(s), ~~reward.add~~

if s=0 or s=6

break loop

end loop

~~if is batch~~

10. s←traj(i), s'←traj(i+1), ~~r←reward~~

~~δ←r+V(s')-V(s)~~

~~sum_δ(s)←sum_δ(s)+δ~~

if method="TD"

~~r←rs(i)~~

δ←r+V(s')-V(s)

else

~~δ=0~~

G←rs(i)

δ←G-V(s)

sum_δ(s)←sum_δ(s)+δ

TD(0)

1. $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$

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

2. Tabular TD(0) for estimating v_π

Input: π

initialize $V(s)$ arbitrarily, $\forall s \in S^+$

eg. $V(s)=0, \forall s \in S^+$

Process:

loop for each episode

initialize $S \in S$

repeat for each step of episode

$A \sim \pi(A|S)$

take A, observe R, S'

$V(S) \leftarrow V(S) + \alpha(R + \gamma V(S') - V(S))$

$S \leftarrow S'$

until S is terminal.

end loop.

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

② $G_t - V(S_t) = \sum_{k=0}^{T-t-1} \gamma^k \delta_{t+k}$, here

$G_T = 0, \quad V(S_T) = 0.$

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

5. Sarsa: an on-policy TD control algorithm.

initialize $Q(s,a)$ arbitrarily, $\forall s \in S, a \in A(s)$. $Q(s, \cdot) = 0$, $\forall s$ is terminal state.

// ∴ $Q(s,a): s \in S^+, a \in A(s)$

Repeat for each episode:

① initialize $S \in S$

② choose A from S using policy derived from Q (e.g., ε-greedy)

③ Repeat (for each step of episode):

take A, observe R, S'.

choose A' from S' using policy derived from Q (e.g., ε-greedy)

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

$S \leftarrow S', \quad A \leftarrow A'.$

④ until S is terminal.

⑤ end repeat for each episode.

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

7. Q-learning: an off-policy TD control algorithm.

Initialize $Q(s,a)$ arbitrarily, $\forall s \in S, a \in A(s)$,

$Q(s,\cdot) = 0, \forall s \in S^+ / S$

Repeat for each episode:

initialize $S \in S$

repeat for each step of episode:

choose A from S using policy derived from Q, (e.g., ε-greedy)

take 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.

end Repeat for each episode.

Initialize S

Repeat for each step of episode

① Choose A from S using policy derived from Q_1, Q_2. (e.g., ε-greedy in Q_1+Q_2)

② Take A, observe R, S'

③ With 0.5 probability:

$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha(R + \gamma Q_2(S', \arg\max_a Q_1(S',a)) - Q_1(S,A))$

else

$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha(R + \gamma Q_1(S', \arg\max_a Q_2(S',a)) - Q_2(S,A))$

④ $S \leftarrow S'$

⑤ until S is terminal

end repeat for each episode.

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

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

9. $Q_1(S_t,A_t) \leftarrow Q_1(S_t,A_t) + \alpha(R_{t+1} + \gamma Q_2(S_{t+1}, \arg\max_a Q_1(S_{t+1},a)) - Q_1(S_t,A_t))$

10. double Q-learning

Initialize $Q_1(s,a)$, $Q_2(s,a)$ arbitrarily, $\forall s \in S, a \in A(s)$.

$Q_1(s,\cdot) = Q_2(s,\cdot) = 0, \forall s \in S^+ / S$

Repeat for each episode:

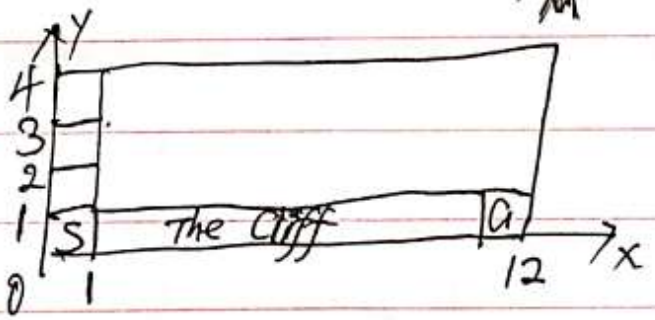
cliff walking

1. $X_M = 12$, $\varepsilon = 0.1$, $\alpha = 0.5$, $\gamma = 1$.

$a_l, a_r, a_u, a_d \leftarrow 1, 2, 3, 4$.

$A = \{a_l, a_r, a_u, a_d\}$

START $S = (1, 1)$, GOAL $= (12, 1)$



2. step(state s, action a):

assert $s \neq$ GOAL.

$x, y \leftarrow s$

if $a = a_l$: $x \leftarrow x-1$

else if $a = a_r$: $x \leftarrow x+1$

else if $a = a_u$: $y \leftarrow y+1$

else $y \leftarrow y-1$

if $x < 1$: $x \leftarrow 1$

else if $x > X_M$: $x \leftarrow X_M$

if $y < 1$: $y \leftarrow 1$

else if $y > Y_M$: $y \leftarrow Y_M$

$s' \leftarrow (x, y)$

$r \leftarrow -1$

if $(x, y) \in [1, X_M) \times [1]$

$s' \leftarrow$ START, $r \leftarrow -100$

return r, s'.

3. $Q(s, a) \in R^{X_M \times Y_M \times A.size}$

$Q(s, a) \leftarrow -1, \forall s, a$

$Q(s, a) \leftarrow -100$, $s \in [1, X_M) \times [1] \times \{a_d\}$

$Q(s, a) \leftarrow -100$, $s =$ START, $a = a_r$.

4. choose action (s, Q)

sample $\leftarrow$ binomial $(1, \varepsilon)$

if sample $= 1$

$i_a \sim$ uniform $(1, 4)$

$a \leftarrow A(i_a)$

else

$a \leftarrow \underset{a}{\text{argmax}}\, Q(s, a)$ randomly

return a.

5. sarsa $(Q$, is_expected $\leftarrow 0$,

$\alpha \leftarrow$ ALPHA)

$s \leftarrow$ START

$a \leftarrow$ choose action (s, Q)

sum_r $\leftarrow 0$

loop:

```
r, s' ← step(s, a)
a' ← choose_action(s', Q)
sum_r ← sum_r + r
if is_expected = 0
    tmp ← Q(s', a')
else
    tmp ← 0
    set_a ← argwhere(Q(s',:) = max_a Q(s',:))
    ∀a ∈ A
        if a ∈ set_a
            p ← (1-ε)/|set_a| + ε/|A|
            tmp ← tmp + p·Q(s', a)
        else
            p ← ε/|A|
            tmp ← tmp + p·Q(s', a)
    end if
tmp ← r + γ·tmp
Q(s, a) ← Q(s, a) + α(tmp - Q(s, a))
s ← s', a ← a'
if s = GOAL : stop loop.
end loop.
return sum_r.
```

6. Q-learning(Q, α=ALPHA)

```
s ← START
sum_r ← 0
loop:
    a ← choose_action(s, Q)
    r, s' ← step(s, a)
    sum_r ← sum_r + r
    Q(s,a) ← Q(s,a) + α·(r + γ max_a Q(s',:) - Q(s,a))
    s ← s'
    if s = GOAL
        stop loop
end loop
return sum_r.
```

7. print_optimal_policy(Q)

7. print_optimal_policy(Q)

$\pi(s) = \arg\max_a Q(s,a), \forall s.$

print π

i.e. ~~if $\pi(s)$~~ $a \leftarrow \pi(s)$

if s = GOAL, print "G"; continue.

if $a = a_l$: print: ←

else if $a = a_r$: print: "→"

else if $a = a_u$: print: "↑"

else: print "↓"

sum_rs_sarsa /= runs

sum_rs_Q_learning /= runs.

plot(sum_rs_sarsa) name="Sarsa"

plot(sum_rs_Q_learning), name="Q_learn

savefigure("sum_r")

print optimal policy(Q_sarsa)

print optimal policy(Q_Q_learning

8 figure_6_4()

episodes = 500

runs = 50

sum_rs_sarsa ← 0^episodes

sum_rs_Q_learning ← 0^episodes

$\forall r_{run} \in \{1, \dots, runs\}$

Q_sarsa ← $R^{x_m \times y_m \times 4}$

Q_Q_learning ← $R^{x_m \times y_m \times 4}$

Q_sarsa ← 0, Q_Q_learning ← 0

$\forall i \in \{1, \dots, episodes\}$

sum_rs_sarsa(i) ← sum_rs_sarsa(i) + sarsa(Q_sarsa)

sum_rs_Q_learning(i) ← sum_rs_Q_learning + Q_learning(Q_Q_learning)

end $\forall$ i, ps

end $\forall r_{run}$

9. figure_6_6()

episodes = 1000, runs = 10

$\alpha s \leftarrow \{0.1, 0.2, \dots, 1\}$

asymptotic_sarsa, asy_sarsa ← 1

asy_expected_sarsa ←

asy_Q_learning ←

interim ~~interim~~ ~~int~~ ~~infinite~~_sarsa ← 4

~~int~~ expected_sarsa ←

int ~~int~~_Q_learning ← 6

performance ← $0^{6 \times |\alpha s|}$

∀ run ∈ {1, ..., runs}

  ∀ i ∈ {1, ..., $|\alpha s|$}

    $\alpha \leftarrow \alpha s(i)$

    Q_sarsa ← $0^{X_m \times Y_m \times 4}$

    Q_expected-sarsa ← $0^{X_m \times Y_m \times 4}$

    Q_Q-learning ← $0^{X_m \times Y_m \times 4}$

    ∀ ep ∈ {1, ..., episodes}

      sum_rs_sarsa ← sarsa(Q_sarsa, ~~0~~ is expected ← 0, α)

      sum_rs_expected_sarsa ← sarsa(Q_expected_sarsa, 1, α)

      sum_rs_Q_learning ← Q_learning (Q_Q_learning, α)

      performance(asy_sarsa, i) ~~+~~ += sum_rs_sarsa

      performance(asy_expected_sarsa, i) += sum_rs_expected_sars

      performance(asy_Q_learning, i) += sum_rs_Q_learning

      if ep ≤ 100

        performance(~~if_intermediat~~ interim_sarsa, i) += sum_rs_sarsa

        ...

        ...

    end ∀ ep, i, run.

    performance(1:3, :) /= episodes · runs

    ·(4:6, :) /= 100 · runs

plot(performance, name='sum_reward per episode')