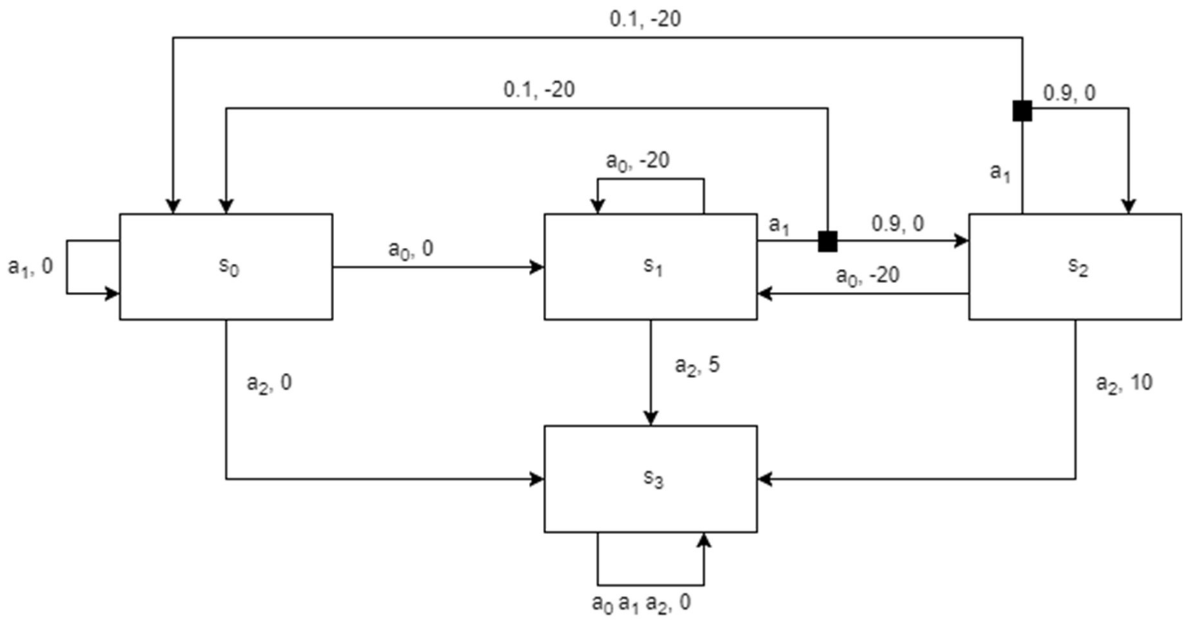
Dish Stacking with Reinforcement Learning

Solution Sheet

1.



a_0 = Grab s_0 = No Plate Held

a_1 = Dry s_1 = Wet Plate Held

a_2 = Store s_2 = Dry Plate Held

s_3 = Finished

Reward Function

	a_0	a_1	a_2
s_0	0	0	0
s_1	-20	$P_f(-20) + \sim p_f(0) = -2$	5
s_2	-20	$P_f(-20) + \sim p_f(0) = -2$	10
s_3	0	0	0

Transition Function

a_0	s_0	s_1	s_2	s_3
s_0	0	1	0	0
s_1	0	1	0	0
s_2	0	1	0	0
s_3	0	0	0	1

a_1	s_0	s_1	s_2	s_3
s_0	1	0	0	0
s_1	0	$P_f = 0.1$	$\sim p_f = 0.9$	0
s_2	0	$P_f = 0.1$	$\sim p_f = 0.9$	0
s_3	0	0	0	1

a_2	s_0	s_1	s_2	s_3
s_0	0	0	0	1
s_1	0	0	0	1
s_2	0	0	0	1
s_3	0	0	0	1

2. A)Example Solution: This MDP is episodic and has a terminating state “finished”. Therefore we can set γ=1.

Set a deterministic policy, I chose to always grab. As finished is the terminal state its action does not matter.

Policy

s_0	a_0
s_1	a_0
s_2	a_0

The probability of doing a in s acording to policy

The value of s' multiplied by the discount factor

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s') \right]$$

New Value of s

$$Q(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma\, V^{\pi}(s') \right]$$

Q value of a in s

The probability of moving to s' by performing a in s

The reward for moving to s' by performing a in s

First I do a policy Evaluation to find V^{policy}

s_0	0
s_1	-20
s_2	-20

Now for a policy improvement step. First I need to determine Q(s,a)

s_0	a_0	-20
	a_1	0
	a_2	0

s_1	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(-20)$ = -2 -18 = -20
	a_2	5

s_2	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(-20)$ = -2 -18 = -20
	a_2	10

I now set the policy to greedily choose the highest value action (random if there’s a tie)

Policy

s_0	a_1
s_1	a_2
s_2	a_2

The policy has changes so I must do the loop again. Policy evaluation 2:

V^{policy}

s_0	0
s_1	5
s_2	10

Policy Improvement 2:

Q(s,a)

s_0	a_0	5
	a_1	0
	a_2	0

s_1	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(10)$ = -2 + 9 = 7
	a_2	5

s_2	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(10)$ = -2 + 9 = 7
	a_2	10

Policy

s_0	a_0
s_1	a_1
s_2	a_2

Policy Evaluation 2:

V^{policy}

s_0	5
s_1	7
s_2	10

Policy Improvement 2:

Q(s,a)

s_0	a_0	7
	a_1	0
	a_2	0

s_1	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(10) =$ $-2 + 9 = 7$
	a_2	5

s_2	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(10) =$ $-2 + 9 = 7$
	a_2	10

Policy

s_0	a_0
s_1	a_1
s_2	a_2

The policy didn’t change so the algorithm is complete.

2B) To do a value iteration, I don't choose an original policy and set the initial values to the highest possible reward from possible actions

Choosing a to maximise V

The value of s' multiplied by the discount factor

$$V_{k+1}(s) = max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s') \right]$$

New Value of s

The probability of moving to s' by performing a in s

The reward for moving to s' by performing a in s

Value

s_0	0
s_1	5
s_2	10

Recalculate Q

s_0	a_0	5
	a_1	0
	a_2	0

s_1	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(10) = -2 + 9 = 7$
	a_2	5

s_2	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(10) = -2 + 9 = 7$
	a_2	10

Recalculate V

s_0	5
s_1	7
s_2	10

Recalculate Q

s_0	a_0	7
	a_1	0
	a_2	0

s_1	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(10) = -2 + 9 = 7$
	a_2	5

s_2	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(10) = -2 + 9 = 7$
	a_2	10

Recalculate V

s_0	7
s_1	7
s_2	10

Recalculate Q

s_0	a_0	7
	a_1	0
	a_2	0

s_1	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(10) =$ $-2 + 9 = 7$
	a_2	5

s_2	a_0	-20
	a_1	$P_f(-20) + {\sim}p_f(10) =$ $-2 + 9 = 7$
	a_2	10

Recalculate V

s_0	7
s_1	7
s_2	10

It hasn't changed so the iteration is complete and now we choose the policy based on the maximum Q values

Policy

s_0	a_0
s_1	a_1
s_2	a_2

2C) I will use ε-greedy exploration, so a policy will not always choose the maximum Q. I also need to define a limit to the number actions before the episode terminates, I'll choose 3. I will set $\gamma=1$.

First-visit MC (I only consider the reward of the first time I explore a state-action pair each episode):

1st "random" policy.

s_0	a_1
s_1	a_2
s_2	a_1

1st Episode (random start)

s_2 –(0)-> s_2 –(-20)-> s_1-(0)-> s_1

Estimate Q

s_0	a_0	0
	a_1	0
	a_2	0

s_1	a_0	0
	a_1	0
	a_2	0

s_2	a_0	0
	a_1	0
	a_2	0

2nd policy chosen based on Q with some error.

s_0	a_0
s_1	a_2
s_2	a_0

2nd Episode

s_1 –(5)-> s_3

Estimate Q

s_0	a_0	0
	a_1	0
	a_2	0

s_1	a_0	0
	a_1	0
	a_2	5

s_2	a_0	0
	a_1	0
	a_2	0

3rd Policy.

s_0	a_0
s_1	a_2
s_2	a_1

3rd Episode

s_0 –(0)-> s_1 –(5)-> s_3

Estimate Q

s_0	a_0	0
	a_1	0
	a_2	0

s_1	a_0	0
	a_1	0
	a_2	5

s_2	a_0	0
	a_1	0
	a_2	0

Etc…

2D) Every-visit MC (I average the rewards that I receive from multiple explorations of a state-action pair each episode):

1st "random" policy.

s_0	a_1
s_1	a_2
s_2	a_1

1st Episode (random start)

s_2 –(0)-> s_2 –(-20)-> s_1-(0)-> s_1

Estimate Q

s_0	a_0	0
	a_1	0
	a_2	0

s_1	a_0	0
	a_1	0
	a_2	0

s_2	a_0	0
	a_1	(0-20)/2 = -10
	a_2	0

2nd policy chosen based on Q with some error.

s_0	a_0
s_1	a_2
s_2	a_1

2nd Episode

s_1 –(5)-> s_3

Estimate Q

s_0	a_0	0
	a_1	0
	a_2	0

s_1	a_0	0
	a_1	0
	a_2	5

s_2	a_0	0
	a_1	-10
	a_2	0

3rd Policy.

s_0	a_0
s_1	a_2
s_2	a_1

3rd Episode

s_0 –(0)-> s_1 –(5)-> s_3

Estimate Q

s_0	a_0	0
	a_1	0
	a_2	0

s_1	a_0	0
	a_1	0
	a_2	5

s_2	a_0	0
	a_1	-10
	a_2	0

Etc…

2E) I will use the same set up as in Monte Carlo but without a maximum length for each episode and with α=0.7.

Update Q values after each step and choose action based on Q each step (with error). For SARSA the update to a Q value uses the formula:

$Q(s,a) = Q(s,a) + \alpha(r+ \gamma Q(s',a')-Q(s,a))$

Where s' is the next state and a' is the action you will take in that state based on your policy.

Start episode at s_2

s_2-(10)-> s_3

Update Q(s_2, a_2)

s_2	a_0	0
	a_1	0
	a_2	O+0.7(10+ (0.5*0)-0) = 7

Learning rate (less than 1, more than 0)

The Q value of your next action, a' (determined by your ε-greedy algorithm) in s'

$Q(s,a) = Q(s,a) + \alpha(\ r+\gamma Q(s',a')-Q(s,a))$

Q value of a in s

The reward for moving to s' by performing a in s

Start new episode at s_1

s_1-(0)-> s_2

a_2 selected as a'. Update Q(s_1, a_1) using value for Q(s_2, a_2)

s_1	a_0	0
	a_1	0 + 0.7(0+ (0.5*7)-0) = 2.45
	a_2	0

s_2-(10)-> s_3

Update Q(s_2, a_2)

s_2	a_0	0
	a_1	0
	a_2	7+0.7(10+ (0.5*0)-7) = 9.1

Start new episode at s_2

At this point the chooses a_1 for a' rather than the action with maximum Q. The plate then breaks while trying to dry it.

s_2-(-20)-> s_1

At this point the ε-greedy algorithm chooses a_2 for a' rather than the action with maximum Q.

Update Q(s_2, a_1) using value for Q(s_1, a_2)

s_2	a_0	0
	a_1	0 + 0.7(-20+(0.5*0)-0) = -15
	a_2	9.1

2F) I will use the same set up as in Monte Carlo but without a maximum length for each episode and with α=0.7.

For Q-Learning the update to Q(s,a) uses the formula:

$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$

Where s' is the next state and a' is chosen to maximise Q(s',a').

Learning rate (less than 1, more than 0)

The maximum Q value for s'

$$Q(s,a) = Q(s,a) + \alpha(r+\gamma \max_{a'} Q(s',a')-Q(s,a))$$

Q value of a in s

The reward for moving to s' by performing a in s

Start episode at s_2

s_2-(10)-> s_3

Update Q(s_2, a_2)

s_2	a_0	0
	a_1	0
	a_2	O+0.7(10+ (0.5*0)-0) = 7

Start new episode at s_1

s_1-(0)-> s_2

a_2 selected as a'. Update Q(s_1, a_1) using value for Q(s_2, a_2) as this is $\max_{a'}$Q(s',a').

s_1	a_0	0
	a_1	0 + 0.7(0+ (0.5*7)-0) = 2.45
	a_2	0

s_2-(10)-> s_3

Update Q(s_2, a_2)

s_2	a_0	0
	a_1	0
	a_2	7+0.7(10+ (0.5*0)-7) = 9.1

Start new episode at s_2

At this point the ε-greedy algorithm chooses a_1 for a' rather than the action with maximum Q. The plate then breaks while trying to dry it.

s_2-(-20)-> s_1

At this point the ε-greedy algorithm chooses a_2 for a' rather than the action with maximum Q.

Update Q(s_2, a_1) using value for Q(s_1, a_1) as this is $\max_{a'}$Q(s',a').

s_2	a_0	0
	a_1	0 + 0.7(-20+(0.5*2.45)-0) = -13.14
	a_2	9.1

5. C, D, E and F are suitable. A and B rely on a complete model and so are unsuitable.

6. Increasing the ε value in an ε-greedy algorithm causes it to become more explorative rather than exploitative. This means the learning algorithm will examine more possible actions and state pairs while looking for an optimal policy. Setting this value too small can hinder finding the optimal policy as the algorithm may not explore enough and never find it. On the other hand, a higher ε value can also increase the time taken to find an optimal policy due to the increased time exploring the rest of the state space.