NAME1: Masoumeh Bakhtiariziabari
STUDENTID1: 11813105
MAIL1: mbakhtiariz@gmail.com

1 Introduction

1.1 Introduction

1.1.1 Explain what is meant by the 'curse of dimensionality'.

"The curse of dimensionality," means that computational requirements grow exponentially with the number of state variables. It happens when number of state in respect to data is too much and it is impossible or hard to solve the problem. The ability of some reinforcement learning methods to learn with parameterized approximators addresses the classical "curse of dimensionality" in operations research and control theory.

1.1.2 Suppose you are trying to design a predator agent that can learn to catch a randomly moving prey on a 5×5 toroidal grid. You have been given the (x,y)-coordinates of the predator and the (x,y)-coordinates of the prey to use as the state.

(a) **How many possible states are there in this naive approach?**
25×25, Because there would be 25 possible position for predator agent and 25 possible position for prey. So the permutation between these states will be total number of possible states.

(b) **There is a way of reducing the state space considerably a priori. Write down how you would adapt the given state representation to reduce the size of the state space. Note that you only care about a representation that you can use to solve the problem.**
We consider to know the position of prey as a priori at each time step. While predator is doing every action selection, only the distance between predator and prey would be important, which is dependant on Δx and Δy. There are 9 possible values for Δx and 9 for Δy because if $x_{predator}, x_{prey} \in \{0, 1, 2, 3, 4\}$ and $y_{predator}, y_{prey} \in \{0, 1, 2, 3, 4\}$ then $\Delta x, \Delta y \in \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$. The permutation of possible values for Δx and Δy will give us number of possible states which is 9×9.

(c) **How many possible states are there now?**
9×9. It has been explained in the answer of previous question.

(d) **What is the advantage of doing this?**
Having lower state space faster convergence and finding solutions easier.

(e) **Consider the Tic-Tac-Toe example in Chapter 1.5 of the book. Here, too, we can exploit certain properties of the problem to reduce the size of the state space. Give an example of a property you can exploit**
For example we can keep $num("X") =$ number of times "X" has happened till now. and $num("O") =$ number of times "O" has happened till now. At each time step we should have the condition of $|num("X") - num("O")| \in \{0, 1\}$. So as a priori here we will first know opponent's action then we will decide our next action. (Figure 1)

1.1.3 Suppose you want to implement a Reinforcement Learning agent that learns to play Tic-Tac-Toe, as outlined in Chapter 1 of the book.

(a) **Which agent do you think would learn a better policy in the end: a greedy agent that always chooses the action it currently believes is best, or a non-greedy agent that sometimes tries new actions? Why?**
A non-greedy agent that sometimes tries new actions. Because exploration is crucial for learning an optimal policy by experiencing states that with an greedy action we might never see.

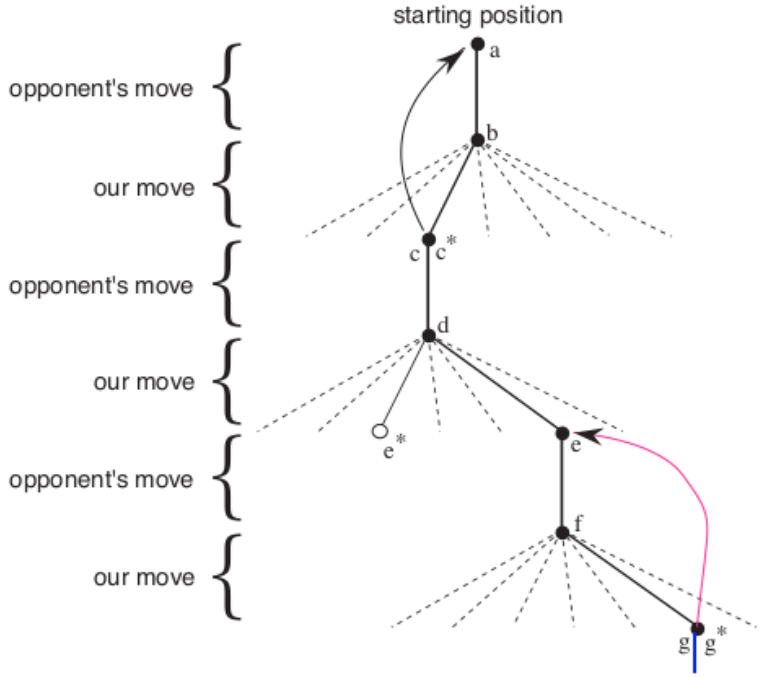

Figure 1: For Introduction Question 1.1.2.(e)- To reduce the size of the state space. as a priori, we will first know opponent's action then we will decide our next action

1.1.4 Assume we start with an exploration rate of ϵ, meaning that whenever the agent chooses an action, it has a probability of ϵ to pick an action at random, and a probability of 1ϵ to pick the greedy action. If we assume the environment has been sufficiently explored,we may want to reduce the amount of exploration after some time.

(a) **Write down how you would do this.**
We can reduce ϵ gradually by passing the time by $\epsilon = \epsilon_0 e^{-\alpha(t)}$ where $0 < \alpha < 1$.

(b) **Does your method work if the opponent changes strategies? Why/why not? If not, provide suggestions on a heuristic that can adapt to changes in the opponent's strategy.**
No it does not, because it does not consider the changing strategy of opponent and we learned the best policy till now based on previous strategy. If optimal policy is rapidly being changed that signifies a likely change in the behaviour of opponent we can take the policy change rate to be reflected in ϵ change equation.

1.2 Exploration

1.2.1 In ϵ-greedy action-selection for the case of n actions, what is the probability of selecting the greedy action?

$\frac{\epsilon}{n} + (1 - \epsilon)$

1.2.2 Consider a 3-armed bandit problem with actions 1,2,3. If we use ϵ-greedy action-selection,initialization at 0, and sample-average action-value estimates, which of the following sequence of actions are certain to be the result of exploration?A1= 1,R1=1,A2=2,R2= 1,A3= 2,R3=2,A4= 2,R4= 2,A5= 3,R5= 1.

A4 ad A5 are the results of exploring, because when those actions where chose there was some other action to have higher Q-value.
$Q(A) = Q(A) + \frac{1}{N(A)}[R - Q(A)]$

Q(A=1)	Q(A=2)	Q(A=3)	greedy action	selected action
0	0	0	1,2,3	$A_1 = 1$
-1	0	0	2,3	$A_2 = 2$
-1	1	0	2	$A_3 = 2$
-1	$1 + \frac{1}{2}(-2 - 1) = -0.5$	0	3	$A_4 = 2$
-1	$-0.5 + \frac{1}{3}(2 + 0.5) = \frac{1}{3}$	0	2	$A_5 = 3$

1.2.3 You are trying to find the optimal policy for a two-armed bandit. You try two approaches:in the pessimistic approach, you initialize all action-values at -5, and in the optimistic approach you initialize all action-values at +5. One arm gives a reward of +1, one arm gives a reward of -1. Using a greedy policy to choose actions, compute the resulting Q-values for both actions after three interactions with the environment. In case of a tie between two Q-values, break the tie at random.

We consider that we have 2 states A and B and 2 type of actions a_1 and a_2.
All Q initial values can be +5 or -5. So at the beginning Q-value for both possible action would be the same and we have a tie between Q-values, so for the beginning we choose first action randomly. In the table shown below we represent scenarios of selection a_1 or a_2 as first action with initial Q-value of +5 or -5.
$Q(A) = Q(A) + \frac{1}{N(A)}[R - Q(A)]$
$Q(B) = Q(B) + \frac{1}{N(B)}[R - Q(B)]$

Q(A)	Q(B)	action	R	$N(A)$	$N(B)$
5	5	a_1	1	1	0
5+(1-5)=1	5	a_2	-1	1	1
1	5+(-1-5)=-1	a_1	1	2	2
1+0.5(1-1)=1	-0.5	-	-	-	-
			$\sum R = 1$		
5	5	a_2	-1	0	1
5	-1	a_1	1	1	1
5+1-5=1	-1	a_1	1	2	1
1+0.5(1-1)=1	-1	-	-	-	-
			$\sum R = 1$		
-5	-5	a_1	1	1	0
-5+(1+5)=1	-5	a_1	1	2	0
1+0.5(1-1)=1	-5	a_1	1	3	0
1	-5	-	-	-	-
			$\sum R = 3$		
-5	-5	a_2	-1	0	1
-5	-5+(-1+5)=-1	a_2	-1	0	2
-5	-1+0.5(-1+1)=-1	a_2	-1	0	3
-5	-1	-	-	-	-
			$\sum R = -3$		

1.2.4 Which initialization leads to a higher (un-discounted) return? What if you had broken the tie differently?

pessimistic mode can result in higher Q-value(+3) or lower(-3) depending on the random choice for tie breaking. With the unlucky tie breaking the return of pessimistic mode can be worse.

Random tie breaking in optimistic initialization does not make a change (for both choice the result was 1) however state-action value function in pessimistic mode is influenced by random tie breaking.

1.2.5 Which initialization leads to a better estimation of the Q-values?

Because of lack of exploration the pessimistic mode can not estimate the true Q-value as good as optimistic one.

1.2.6 Explain why one of the two initialization methods is better for exploration.

For optimistic mode we have better exploration because the action which are not explored and their Q-value are not updated, will have higher Q-value than explored ones and this causes to have more exploration.

2 MDPs and dynamic programming

2.1 Markov Decision Processes

2.1.1

(a) **For the first four examples outlined in Section 1.2 of the book, describe the state space, action space and reward signal.**

- Chess:
  - State space: 8×8 grid and the each unique piece information. Whether the agent plays black or white.
  - Action space: A permissible movement of each chess piece.
  - Reward signal: Winning will have the maximum positive reward , while loosing will have the maximum negative reward , drawing reward will be based on some heuristics . While each time the agent removes opponent chess piece and based on the level of piece the agent gets positive reward while each time agent chess piece gets remove it gets a negative reward .
- Adaptive controller
  - State space: All the possible different levels of each parameters of interest in petroleum.
  - Action space: Increasing/decreasing the level of each parameter.
  - Reward signal: A signal that combines yield/cost/qualities of produced petroleum.
- Calf
  - State space: The status of each of muscles and joints of calf as well as the steepness/slipperiness of the ground.
  - Action space: Contraction/expansion of each muscle and changing the angle of joints.
  - Reward signal: Some negative reward for falling down and positive reward for higher speeds of running.
- Mobile robot
  - State space: 1) Relative position to charging spot, 2) Position/amount of all trash pieces, 3) Amount of remaining battery.
  - Action space: Moving in four directions, grab trash, plug to charge.
  - Reward signal: negative reward for each move while searching , positive reward for every trash picked up, huge negative reward if robot exhausts it battery completely, negative reward if the robot charges itself per time step .

(b) **Come up with an example of your own that you might model with an MDP. State the action space, state space and reward signal.**

Boxing can be modeled with an MDP:

- State space: 1) relative position of agent w.r.t. the opponent, 2) Scores, 3) The guard status of the opponent.
- Action space: Punching and kicking the opponent on each size or at different height, Closing the gaurd.
- Reward signal: +1 for hitting the opponent, -1 for getting hit, +10 for winning, -10 for loosing.

(c) **Come up with an example for a problem that you might have trouble solving with an MDP. Why doesn't this fit the framework?**
Autonomous driving, because environment is partially observable and only the current observable state of the world is not sufficient to make rational decision.

(d) **In mazes, the agent's position is often seen as the state. However, the agent's position alone in not always a sufficient description. Come up with an example where the state consists of the agent's location and one or more other variables.**
For the maze knowing the current location is not enough, for example if we already have tried going to the left at the current position and we encountered a dead-end before, we have to memories it to prevent going to the left again. So we need both current position and history of our movements and its corresponding results.

(e) **Consider the example in exercise 3.3 of the book. Why might you choose to view the actions as handling the accelerator, brake and steering wheel? What is the disadvantage of doing this?**
If the agent lacks the high level skills and the problem itself is to learn the driving then such et of low level action is the right level of abstraction to use. Disadvantage of this is: with such abstraction level solving completed problems would result in sequence of thousands of actions.

(f) **Why might you choose to view the actions as choosing where to drive? What is the disadvantage of doing this?**
If the agent aims to solve a high level problem the defined set of actions should also be high level, for instance, if the agent wants to learn to solve the traveling sales man driving from city A to B, it is the right level of abstraction. Disadvantage of this can be seen in the case where doing such high level actions is not trivial to the agent and due to the complexity of the task we have to ignore it to make the solution sensible.

(g) **Can you think of some way to combine both approaches?**
One way would be identification of useful intermediate skills that abstract a number of low level actions into a useful higher level action. By separately learning /mastering such skills with low level actions we can simply use such skills as building blocks to solve a complex high level problem.

2.1.2

(a) **Eq. 3.8 in the book gives the discounted return for the continuous case. Write down the formula for the discounted return in the episodic case.**
An episodic task can be seen like continuous one which ends after "T" actions so $G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$

(b) **Show that $\sum_{k=0}^{\inf} \gamma^k = \frac{1}{1-\gamma} if 0 =< \gamma < 1$. (Hint: if you're stuck, have a look at the Wikipedia page on geometric series)**
$a = \gamma^0 + \gamma^1 + \gamma^2 + ... + \gamma^{n-1}$ where $n \simeq \inf$
$\gamma a = \gamma^1 + \gamma^2 + \gamma^3 + ... + \gamma^n$

$a - \gamma a = \gamma^0 - \gamma^n = 1 - \gamma^n$

$a = \frac{1-\gamma^n}{1-\gamma} = \frac{1-\gamma^{\inf}}{1-\gamma} \overset{0 \leq \gamma < 1}{\Longrightarrow} a = \frac{1}{1-\gamma}$

(c) **Consider exercise 3.7 in the book. Why is there no improvement in the agent?**
The agent gets no feedback on the length of its solution no matter how much it wastes time in the maze or get out quickly it will get reward if +1.

(d) **How would adding a discount factor of γ i1 help solve this problem?**
Yest, the longer path the solution has with a $\gamma < 1$ the more discounted the reward to the certain action will be therefore the actions that guide the agent to quickly reach the final state will be better rewarded and encouraged.

(e) **How might changing the reward function help solve this problem?**
r=-1 for non terminating actions and a large positive value as reward for the terminating actions.

2.2 Dynamic Programming

2.2.1 Write the value, $v^\pi(s)$, of a states under policy π, in terms of π and $q^\pi(s,a)$. Write down both the stochastic and the deterministic policy case.

Deterministic: $V_\pi(s) = max_{a\in A} q_\pi(s,a)$
Stochastic: $V_\pi(s) = \sum_{a\in A} \pi(a|s) q_\pi(s,a)$

2.2.2 In Policy Iteration, we first evaluate the policy by computing its value function, and then update it using a Policy Improvement step. You will now change Policy Iterationas given on page 80 of the book to compute action-values. First give the new policy evaluation update in terms of $Q^\phi(s,a)$ instead of $V^\pi(s)$. Note that Policy Improvement uses deterministic policies.

1. Initialization
Q(s,a) $\in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
Loop:
$\Delta \leftarrow 0$
Loop for each $s \in \mathcal{S}$:
loop for each a∈A:
q = Q(s,a)
Q(s,a) $= \sum_{s'r} p(s', r|s, a)[r + \gamma max_{a'} Q(s', a')]$
$\Delta \leftarrow \max(\Delta,$ **|q-Q(s,a)|**)
until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

2.2.3 Now change the Policy Improvement update in terms of $Q_\pi(s,a)$ instead of $V^\pi(s)$.

3. Policy Improvement
policy-stable $\leftarrow$ *true*
For each $s \in \mathcal{S}$:
old-action $\leftarrow \pi(s)$
$\pi(s) \leftarrow \arg\max_a$ **q(s,a)**
If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

2.2.4 The Value Iteration update, given in the book in Eq. 4.10 can also be rewritten in terms of Q-values. Give the Q-value Iteration update.

$Q_{k+1}(s,a) = \sum_{s',r} p(s', r|s, a) max_{a'} Q(s', a')$

3 Monte Carlo methods

3.1 Monte Carlo

3.1.1 Consider an MDP with a single states0that has a certain probability of transitioning back onto itself with a reward of 0, and will otherwise terminate with a reward of 5. Your agent has interacted with the environment and has gotten the following three trajectories:[0,0,5],[0,0,0,0,5],[0,0,0,5]. Use= 0.9.

(a) **Estimate the value of s_0 using first-visit MC.**
V(s_0) will be 3.658 by these calculations:

[0,0,5]
$G = 5 + \gamma G = 5$
$G = 0 + (0.9)(5) = 4.5$
$G = 5 + (0.9)(4.5) = 4.05$
$\rightarrow$ Returns(s_0) = [4.05], V(s_0) = 4.05

[0,0,0,0,5]
$G = 5 + \gamma G = 5$
$G = 0 + (0.9)(5) = 4.5$
$G = 5 + (0.9)(4.5) = 4.05$
$G = 5 + (0.9)(4.05) = 3.645$
$G = 5 + (0.9)(4.5) = 3.2805$
$\rightarrow$ Returns(s_0) = [4.05,3.2805], V(s_0) = 3.6652

[0,0,0,5]
$G = 5 + \gamma G = 5$
$G = 0 + (0.9)(5) = 4.5$
$G = 5 + (0.9)(4.5) = 4.05$
$G = 5 + (0.9)(4.05) = 3.645$
$\rightarrow$ Returns(s_0) = [4.05,3.2805, 3.645], V(s_0) = 3.658

(b) **Estimate the value of s_0 using every-visit MC.**
For first visit for every episode we reset the Returns(s_0)=[]. While for every visit we keep all returns of s_0 from all episodes and at the end the average of all those values will be V(s_0).

$$V(s_0) = [(5 + 4.5 + 4.05) + (5 + 4.5 + 4.05 + 3.645 + 3.2805) + (5 + 4.5 + 4.05 + 3.645)\ /12 = 4.268$$

3.1.2 What is a disadvantage of using ordinary importance sampling in off-policy Monte Carlo?

The variance of ordinary importance sampling is in general unbounded because the variance of the ratios can be unbounded.

3.1.3 What is a disadvantage of using weighted importance sampling in off-policy Monte Carlo?

weighted importance sampling is biased (though the bias converges asymptotically to zero).

4 Temporal Difference methods

4.1 Temporal Difference Learning (Application)

4.1.1 What are the state(-action) value estimates V(s) (or Q(s,a)) after observing the sample episode (where we initialize state values to 0 and use a learning rate= 0.1) when applying:

(a) **TD(0)**
$v(s_t) = v(s_t) + \alpha[R_{t+1} + \gamma v(s_{t+1}) - v(s_{t+1})]$

action	reward	updated V(A)	updated V(B)
-	-	0	0
a3	-3	-0.3	0
a4	+4	-0.3	0.37
a5	-4	-0.7	0.37
a6	-3	-0.893	0.37
a7	+1	-0.893	0.433

$A \xrightarrow[R=-3]{a3} B : v(A) = -0.3$
$B \xrightarrow[R=4]{a4} A : v(B) = 0 + (0.1)[4 - 0.3 - 0] = 0.37$
$A \xrightarrow[R=-4]{a5} A : v(A) = -0.3 + (0.1)[-4 - 0.3 + 0.3] = -0.7$
$A \xrightarrow[R=-3]{a6} B : v(A) = -0.7 + (0.1)[-3 + 0.37 + 0.7] = -0.893$
$B \xrightarrow[R=1]{a7} T : v(B) = 0.37 + (0.1)[1 + 0 - 0.37] = 0.433$

(b) **3-step TD**
$v(s_t) = v(s_t) + \alpha[estimation - v(s_{t+1})]$
$estimation = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v(s_{t+3})$

action	reward	updated V(A)	updated V(B)
-	-	0	0
a3,a4,a5	-3,4,-4	-0.3	0
a4,a5,a6	+4,-4,-3	-0.3	-0.3
a5,a6,a7	-4,-3,1	-0.87	-0.3

$A \xrightarrow[R=-3]{a3} \xrightarrow[R=4]{a4} \xrightarrow[R=-4]{a5} A : v(A) = 0 + (0.1)[-3 + 4 - 4 + 0 - 0] = -0.3$
$B \xrightarrow[R=4]{a4} \xrightarrow[R=-4]{a5} \xrightarrow[R=-3]{a6} A : v(B) = 0 + (0.1)[4 - 4 - 3 + 0 - 0] = -0.3$
$A \xrightarrow[R=-4]{a5} \xrightarrow[R=-3]{a6} \xrightarrow[R=1]{a7} A : v(A) = -0.3 + (0.1)[-6 + 0 + 0.3] = -0.87$

(c) **SARSA**
$Q(s,a) = Q(s,a) + \alpha[R + \gamma Q(s',a') - Q(s,a)]$

action	reward	next action	updated Q(A,1)	updated Q(A,2)	updated Q(B,1)	updated Q(B,2)
-	-	-	0	0	0	0
a3	-3	a4	-0.3	0	0	0
a4	+4	a5	-0.3	0	0.4	0
a5	-4	a6	-0.3	-0.43	0.4	0
a6	-3	a7	-0.57	-0.43	0.4	0
a7	1		-0.57	-0.43	0.4	0.1

$A \xrightarrow[R=-3]{a3=1} B \xrightarrow{a4=1}: Q(A,1) = 0 + (0.1)[-3 + 0 - 0] = -0.3$
$B \xrightarrow[R=4]{a4=1} A \xrightarrow{a5=2}: Q(B,1) = 0 + (0.1)[4 - 0 - 0] = 0.4$

$A \xrightarrow[R=-4]{a5=2} A \xrightarrow{a6=1}: Q(A,2) = 0 + (0.1)[-4 - 0.3] = -0.43$
$A \xrightarrow[R=-3]{a6=1} B \xrightarrow{a7=2}: Q(A,1) = -0.3 + (0.1)[-3 + 0.3] = -0.57$
$B \xrightarrow[R=1]{a7=2} T \xrightarrow{\cdot} Q(B,2) = 0 + (0.1)[1 + 0] = 0.1$

(d) **Q-learning**
$Q(s,a) = Q(s,a) + \alpha[R + \gamma max_{a'} Q(s',a') - Q(s,a)]$

action	reward	next action	updated Q(A,1)	updated Q(A,2)	updated Q(B,1)	updated Q(B,2)
-	-	-	0	0	0	0
a3	-3	a4	-0.3	0	0	0
a4	+4	a5	-0.3	0	0.4	0
a5	-4	a6	-0.3	-0.4	0.4	0
a6	-3	a7	-0.53	-0.4	0.4	0
a7	1		-0.53	-0.4	0.4	0.1

$A \xrightarrow[R=-3]{a3=1} B \xrightarrow{a=1}: Q(A,1) = 0 + (0.1)[-3 + 0 - 0] = -0.3$
$B \xrightarrow[R=4]{a4=1} A \xrightarrow{a=2}: Q(B,1) = 0 + (0.1)[4 - 0 - 0] = 0.4$
$A \xrightarrow[R=-4]{a5=2} A \xrightarrow{a=2}: Q(A,2) = 0 + (0.1)[-4 - 0] = -0.4$
$A \xrightarrow[R=-3]{a6=1} B \xrightarrow{a=1}: Q(A,1) = -0.3 + (0.1)[-3 + 0.4 - 0.3] = -0.53$
$B \xrightarrow[R=1]{a7=2} T \xrightarrow{\cdot} Q(B,2) = 0 + (0.1)[1 + 0] = 0.1$

4.1.2 Choose a deterministic policy that you think is better than the random policy given the data. Refer to any of the state(-action) value estimates to explain your reasoning.

I choose greedy policy which is $\pi(A) = 2, \pi(B) = 1$ referring to the estimated Q-learning table in question (7.1.(d)). Given the observed data such a policy is maximum liklihood for getting highest return.

4.1.3 Let π_{random} denote the random policy used so far and $\pi_{student}$ denote the new policy you proposed. Suppose you can draw new sample episodes indefinitely until convergence of the value estimates.

(a) **Discuss how do you expect the final value estimates to differ if you ran Q-Learning with π_{random} compared to $\pi_{student}$.**
π_{random} is guaranteed to converge therefore the final $Q(s,a)$ is more optimal than the one obtained from $\pi_{student}$

(b) **What problems may arise with π_{random} or $\pi_{student}$ respectively?**
With π_{random} convergence is slower. With $\pi_{student}$ there is a possibility that some of state-action pairs do not continue to be visited to guarantee the convergence to the optimal state-value function.

(c) **Do you think using an ϵ-greedy policy as behavior policy would be beneficial? Explain why/why not?**
Yes, it would be beneficial because it converges to optimal ϵ greedy state value . Most of the time it takes greedy action with $1 - \epsilon$ but sometimes given a state it takes action with ϵ which makes sure that all state action value are explored given infinite time and thus optimal states action values will be found

4.2 Contraction Mapping

4.3 Temporal Difference Learning (Theory)

4.3.1 Show that $V_M(S)$ can be written as the update rule $V_M(S) = V_{M-1}(S) + \alpha_M[G_M(S) - V_{M-1}(S)]$ and identify the learning rate α_M.

$V_M(S) = \frac{1}{M}\sum_{n=1}^{M} G_n(S)$
$= \frac{1}{M}(\sum_{n=1}^{M-1} G_n(S) + G_M(S))$
$= \frac{M-1}{M}\left[\frac{1}{M-1}\sum_{n=1}^{M-1} G_n(S)\right] + \frac{1}{M}G_M(S)$
$= \frac{M-1}{M}V_{M-1}(S) + \frac{G_M(S)}{M}$
$= (1 - \frac{1}{M})V_{M-1}(S) + \frac{G_M(S)}{M} = V_{M-1}(S) + \frac{G_M(S) - V_{M-1}(S)}{M}$
$\rightarrow V_M(S) = V_{M-1}(S) + \frac{1}{M}\left[G_M(S) - V_{M-1}(S)\right]$

Therefore, $\alpha_M = \frac{1}{M}$

4.3.2 Consider the TD-error $\delta_t = R_{t+1} + \gamma V(St+1) - V(St)$.

(a) **What is $E[\delta_t|St = s]$ if δ_t uses the true state-value function V_π?**

It will be zero because:
$E[\delta_t|St = s] = E[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)|S_t = s]$
If we for optimal state consider: $V^\pi(S_t) = E[R_{t+1} + \gamma V^\pi(S_{t+1})]$
Then:$E[\delta_t|St = s] = E[R_{t+1} + \gamma V(S_{t+1})R_{t+1} - R_{t+1} - \gamma V^\pi(S_{t+1})|S_t = s] = 0$

(b) **What is $E[\delta_t|S_t = s, A_t = a]$ if δ_t uses the true state-value function V_π**

$$\begin{aligned}
\mathbb{E}[\delta_t|S_t = s, A_t = a] &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)|S_t = s, A_t = a] \\
&= \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) - V_\pi(S_t)|S_t = s, A_t = a] \\
&= \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s, A_t = a] - V_\pi(s) \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] - V_\pi(s) \\
&= Q_\pi(s, a) - V_\pi(s)
\end{aligned}$$

.

4.3.3 The Monte-Carlo error can be written as the sum of TD-errors if the value estimates don't change.

(a) **Show that the n-step error can also be written as a sum TD errors (assuming the value estimates don't change).**

$$\begin{aligned}
V_{t+n}(S_t) &= V_{t+n-1} + \alpha[(G_{t:t+n}) - V_{t+n-1}(S_t)] \\
&= V_{t+n-1} + \alpha[R_{t+1} + \gamma G_{t+2} - V_{t+n-1}(S_t)] \\
&= V_{t+n-1} + \alpha[\underline{R_{t+1} + \gamma G_{t+2} - V_{t+n-1}(S_t) + \gamma V_{t+n-1}S_{t+1} - \gamma V_{t+n-1}S_{t+1}}]_{\text{Similar to eq given in question and the book}} \\
&= V_{t+n-1} + \alpha\sum_{k=t}^{t+n}\gamma^{k-t}\delta_k
\end{aligned}$$

4.4 Maximization Bias

4.4.1 We repeatedly apply Q-learning and SARSA on the observed data until convergence. Give all final state-action values for Q-learning and SARSA respectively.

Algorithm	Q(A,1)	Q(A,2)	Q(B,1)	Q(B,2)	Q(B,3)	Q(B,4)
Q-Learning	2	1.5	$\frac{0+2}{2}=1$	$\frac{2+0}{2}=1$	$\frac{2+2}{2}=2$	$\frac{0+0}{2}=0$
Sarsa	2-ϵ	1.5	$\frac{0+2}{2}=1$	$\frac{2+0}{2}=1$	$\frac{2+2}{2}=2$	$\frac{0+0}{2}=0$

For finding Q(B,a) we average on the achieved values. So $Q(B,a) = E_a(B)$.
Q(A,2) will be 1.5 for both algorithm because there is only one available way from this path to Terminal state.
Best route for Q-learning depends on greedy selection of $A \xrightarrow{goingLeft} B \xrightarrow{action3} T$ which is not obviously the optimal path. Optimal path would be going from A to right to achieve T but result of Q(A,1) is higher than Q(A,2).
For SARSA we use ϵ-greedy to choose next action as well as value update after going from A to B. For obtaining Q(A,1) we have: $Q(A,1) = 0 + [0 + (\frac{\epsilon}{4} + 1 - \epsilon)Q(B,3) + \frac{\epsilon}{4}Q(B,1) + \frac{\epsilon}{4}Q(B,2) + \frac{\epsilon}{4}Q(B,4)] = 0 + [0 + (\frac{\epsilon}{4} + 1 - \epsilon) * 2 + \frac{\epsilon}{4} + \frac{\epsilon}{4} + 0] = 2 - \epsilon$ which is the expected value

if $2 - \epsilon < Q(A,2)$ or $0.5 < \epsilon$ then SARSA will choose action 2 as the best action from A to next state otherwise it will be deceived and will choose action 1 and will go to left path.

4.4.2 This problem suffers from maximization bias. Explain where this can be observed. Do both Q-learning and SARSA suffer from this bias? Why/why not?

While optimal action on state A is going to right but it goes to left using Q learning due to maximization bias, Q-learning has always maximization bias but SARSA depending on value of ϵ (assuming it follows an ϵ-greedy policy) might suffer from the same issue (if $\epsilon < 0.5$)

4.4.3 To circumvent the issue of maximization bias, we can apply Double Q-learning. Use the given example to explain how Double Q-learning alleviates the problem of maximization bias.

Maximization bias error happens because on action from B to next state we have wrong optimistic bias with high reward values due to maximization operation in update rule. To solve this issue DoubleQ-learnig uses 2 separate Q-tables one for getting the index of max and other for the value, given that the maximum on the 2 tables do not necessary correspond together this solution avoid maximization bias.So if we consider $Q_1(a)$ and $Q_2(a)$ as 2 learning estimates . We use $Q_1(a)$ to find the max action and $Q_2(a)$ for the expected values. So for example sake if we give the first set of values to $Q_1(a)$ and the other set to $Q_2(a)$ then we get $Q_1(B,2)$ and $Q_1(B,3)$ as best set of action and now we determine the value using $Q_2(B,a)$ we get $Q_2(A,r) = 1$. So the problem of maximization bias gets solved

4.4.4 What are the true state-action values that we would expect to get (after convergence) if we continued sampling episodes.

Algorithm	Q(A,1)	Q(A,2)	Q(B,1)	Q(B,2)	Q(B,3)	Q(B,4)
Q-Learning and Sarsa	1	1.5	1	1	1	1

After convergence the probability of having both 0 and 2 as observed reward (while going from B to T) will be the same and equal to 0.5. So expected reward going from to B to T will be 1 no matter which action it choose. This will result having Q(A,1)=1 for both SARSA and Q-learning and this value is less than Q(A,2)=1.5 so going to right will be always the winner action from A.