CS 4700: Foundations of Artificial Intelligence **(Due: 4/28/20 11:59pm EDT)**

Homework #4: Learning Approaches SOLUTIONS

50 Points Total

Instructor: Haym Hirsh *Name:* Student name, *Netid:* NetId

Course Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission.

- Please include your name and NetIDs on the first page. We recommend typesetting your submission in LaTeX, and an Overleaf template is linked here.
- Homeworks must be submitted via Gradescope by the due date and time.
- Late homeworks are accepted until **4/30/20 at 11:59pm EDT** for a 50% penalty per course policy.
- All sources of material outside the course must be cited. The University Academic Code of Conduct will be strictly enforced.

Problem 1: Policy Iteration (10 points)

A student robot is a bot mimic the behaviour of a student.

As a simple example, assume that the robot can only distinguish between two knowledge levels; the set of states is $S = \{high, low\}$. knowledge level indicates how well the student bot understand the course material.

In each state, the agent can decide whether to

- study hard consistently ("study")
- take exam at any time ("exam")
- watch Netflix for fun ("Netflix")

Suppose this system can be represented as an MDP with the following transition probabilities and rewards. And the Reward is defined as the satisfaction of the study bot.

s	a	s'	$P(s'\|s,a)$	$R(s,a,s')$	Intuition
high	study	high	1.0	+0	If the student robot is at the high knowledge level, keep studying will let it stay in high knowledge level. However, the study action only will not increase its satisfaction
		low	0.0	+0	
high	exam	high	0.9	+10	If the robot is at the high knowledge level, it will have a 90% to get a high score when taking the exam, which will give it a positive utility of 10. In addition it will still stay in the high knowledge state. However, it will have 10% of chance to get a low score, and it will know that it didn't really understand all the knowledge, so it will switch back to the low knowledge level.
		low	0.1	−10	
high	Netflix	high	1.0	+1	Watching Netflix will let the student feel happier at the moment but not increase the knowledge level
		low	0.0	+1	
low	study	high	0.3	+0	If the bot study in low knowledge state, it will have 30% chance to go to high knowledge state
		low	0.7	+0	
low	exam	high	0.05	+10	taking exams at a low knowledge level may not be a good choice, but there are 5% chance that it smash the exam, and get a high score and get into high knowledge state during the exam
		low	0.95	−10	
low	Netflix	high	0.0	+1	Watching Netflix will let the student feel happier at the moment but not increase the knowledge level
		low	1.0	+1	

A (3 points): How many unique policies are there in this example?
Solution: For each of the 2 states, there are 3 possible actions. So you have $3^2 = 9$ policies.

B (7 points): Manually run policy iteration for this MDP, following the pseudocode in the lecture slides. Set the initial policy to be the policy of always studying. Initialize the utility of both states to be 0. Let $\gamma = 0.5$. For each iteration, calculate the updated utilities and updated policy for each state, and show your work.

You only need to show 2 iterations. Box your policy choice of each iteration. And did it converge?

Solution:

Iteration 1:

Policy evaluation (calculate utilities):

$$\begin{aligned}\hat{U}_1(high) \leftarrow & P(high|high, study) \cdot (R(high, study, high) + \gamma \hat{U}_0(high)) \\ & + P(low|high, study) \cdot (R(high, study, low) + \gamma \hat{U}_0(low)) \\ & = 1 \cdot (0 + \gamma \cdot 0) + 0 \cdot (0 + \gamma \cdot 0) \\ & = 0\end{aligned}$$

$$\begin{aligned}\hat{U}_1(low) \leftarrow & P(high|low, study) \cdot (R(low, study, high) + \gamma \hat{U}_0(high)) \\ & + P(low|low, study) \cdot (R(low, study, low) + \gamma \hat{U}_0(low)) \\ & = 0.3 \cdot (0 + \gamma \cdot 0) + 0.7 \cdot (0 + \gamma \cdot 0) \\ & = 0\end{aligned}$$

Policy improvement:

For state "high":

- Utility of choosing "study": $1 \cdot (0 + 0.5 \cdot 0) + 0 \cdot (0 + 0.5 \cdot 0) = 0$

- Utility of choosing "exam": $0.9 \cdot (10 + 0.5 \cdot 0) + 0.1 \cdot (-10 + 0.5 \cdot 0) = 8$
- Utility of choosing "Netflix": $1.0 \cdot (1 + 0.5 \cdot 0) = 1$

So set $\boxed{\hat{\pi}_1(high) \leftarrow exam}$.
For state "low":

- Utility of choosing "study": $0.3 \cdot (0 + 0.5 \cdot 0) + 0.7 \cdot (0 + 0.5 \cdot 0) = 0$
- Utility of choosing "exam": $0.05 \cdot (10 + 0.5 \cdot 0) + 0.95 \cdot (-10 + 0.5 \cdot 0) = -9$
- Utility of choosing "Netflix": $1.0 \cdot (1 + 0.5 \cdot 0) = 1$

So set $\boxed{\hat{\pi}_1(low) \leftarrow Netflix}$.
Iteration 2:
Policy evaluation:

$$\begin{aligned}\hat{U}_2(high) \leftarrow & P(high|high, exam) \cdot (R(high, exam, high) + \gamma \hat{U}_1(high)) \\ & + P(low|high, exam) \cdot (R(high, exam, low) + \gamma \hat{U}_1(low)) \\ & = 0.9 \cdot (10 + \gamma \cdot 0) + 0.1 \cdot (-10 + \gamma \cdot 1) \\ & = 8\end{aligned}$$

$$\begin{aligned}\hat{U}_2(low) \leftarrow & P(high|low, Netflix) \cdot (R(low, Netflix, high) + \gamma \hat{U}_1(high)) \\ & + P(low|low, Netflix) \cdot (R(low, Netflix, low) + \gamma \hat{U}_1(low)) \\ & = 1 \cdot (1 + \gamma \cdot 0) \\ & = 1\end{aligned}$$

Policy improvement:
For state "high":

- Utility of choosing "study": $1 \cdot (0 + 0.5 \cdot 8) + 0 \cdot (0 + 0.5 \cdot 1) = 4$
- Utility of choosing "exam": $0.9 \cdot (10 + 0.5 \cdot 8) + 0.1 \cdot (-10 + 0.5 \cdot 1) = 11.65$
- Utility of choosing "Netflix": $1.0 \cdot (1 + 0.5 \cdot 8) = 5$

So set $\boxed{\hat{\pi}_2(high) \leftarrow exam}$
For state "low":

- Utility of choosing "study": $0.3 \cdot (0 + 0.5 \cdot 8) + 0.7 \cdot (0 + 0.5 \cdot 1) = 1.55$
- Utility of choosing "exam": $0.05 \cdot (10 + 0.5 \cdot 8) + 0.95 \cdot (-10 + 0.5 \cdot 1) = -8.325$
- Utility of choosing "Netflix": $1.0 \cdot (1 + 0.5 \cdot 8) = 1.5$

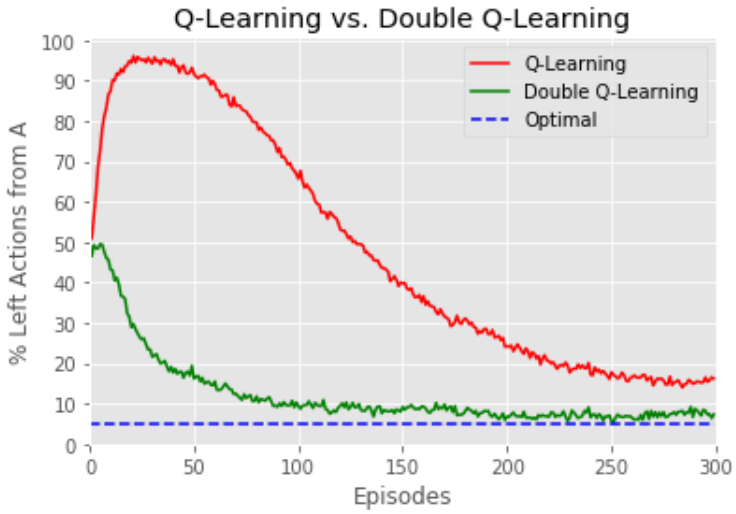
So set $\boxed{\hat{\pi}_2(low) \leftarrow study}$
not converged

Problem 2: Double Q-Learning (15 points)

Please go to the Problem 2 Appendix (at the end of the document) to read the motivations and tutorials needed for this problem.

A (6 points): Please provide pseudocode for the Double Q-learning algorithm. Use Q1 and Q2 to indicate the two q-value tables and update each table with uniform probability: P(update Q1) = P(update Q2) = 0.5. Finally assume that the policy used to get the actions is an ϵ-greedy policy.
Solution:

```
Double Q Learning:
  Get initial s
  for each episode:
    Choose an ε-greedy action to take, a, using the q-value function Q1+Q2
    Take action a and observe next state, s' and reward, R(s, a, s')
    With probability 0.5:
      Q1(s, a) = (1-α)Q1(s, a) + α[R(s, a, s') + γQ2(s', argmax Q1(s', a'))]
    else:
      Q2(s, a) = (1-α)Q2(s, a) + α[R(s, a, s') + γQ1(s', argmax Q2(s', a'))]
    s = s'
  end when s is terminal
```



When running Q-Learning and Double Q-Learning on the MDP defined in the Problem 2 Appendix, we get the graph above. Use this graph to answer Part B and C.

B (3 points): Why is the optimal policy choosing the left action 5% of the time? (Hint: what is the ϵ in the ϵ-greedy policy)
Solution: The optimal solution still has a small chance of error because we are choosing ϵ-greedy actions. If our $\epsilon = 0.1$, then we have a 10% chance of choosing a random action. Given that we have 2 actions to choose from, that would mean we have a 5% chance of choosing the wrong action.

C (3 points): How does the graph above demonstrate the maximization bias for Q-learning?
Solution: For the first couple hundred episodes, we see that the Q-learning agent chooses to pick the left action with much greater frequency than the double q-learning agent. That means that the estimated q-value for the left action from state A has a strong positive bias relative to its true expected reward of -0.1. Also note that at the tail, the Q-learning agent still struggles to find the optimal solution.

D (3 points): If two Q functions helped the policy converge to the optimal more quickly, what about three Q functions? What about k Q functions? What part of the implementation makes it difficult to create such an algorithm?
Solution: Within the update of a k Q-learning algorithm, it is unclear how to interweave the multiple Q-value estimates. For example for $k = 3$, if we want to update the estimate of $Q1$, do we use the values from $Q2$ or $Q3$? Is this chosen at random? Another valid reason lies in the sample efficiency. If we keep splitting up the data into more and more parts, then each estimate is based upon less and less samples. Given these facts, it is unclear that multiple Q-value estimates beyond 2 are necessary and empirically there are minimal performance increases if any.

Problem 3: Multi-armed bandits (15 points)

A bandit has n arms where the i'th arm produces a reward of $0, 1, 2$ with probability $p_{i_0}, p_{i_1}, p_{i_2}$. For this question suppose we have 2 arms: M_a with $p_{a_0} = 0.2, p_{a_1} = 0.4, p_{a_2} = 0.4$, and M_b with $p_{b_0} = 0.6, p_{b_1} = 0.2, p_{b_2} = 0.2$.

A (6 points): A very simple optimal policy to the multi armed bandits problem is to always pull the arm with the largest *Gittens Index*. The Gittens Index for an arm M is defined as:

$$\lambda(M) = \max_{T>0} \frac{\mathbb{E}\left(\sum_{t=0}^{T-1} \gamma^t R_t\right)}{\sum_{t=0}^{T-1} \gamma^t}$$

where R_t is the random variable denoting the reward at time t and γ is our usual discount factor. What is the Gittens Index for each of the arms M_a, M_b? What is the optimum policy?

Solution: We use linearity of expectation. No matter what the maximizer T^* ends up being, we have for arm M with probability p_0, p_1, p_2 getting reward of $0, 1, 2$

$$\begin{aligned}\lambda(M) &= \frac{\sum_{t=0}^{T^*-1} \gamma^t E(R_t)}{\sum_{t=0}^{T^*-1} \gamma^t} \\ &= \frac{(p_1 + 2p_2)\sum_{t=0}^{T^*-1} \gamma^t}{\sum_{t=0}^{T^*-1} \gamma^t} \\ &= p_1 + 2p_2\end{aligned}$$

So the Gittens Index for M_a is 1.2, and M_b is 0.6. The optimum policy is to pull arm M_a all the time.

B (6 points): For more complex problems, calculating the Gittens Index may prove to be difficult. In class we've seen the Upper Confidence Bound (UCB) heuristic, which is an approximately optimal policy with provable bounds on the expected regret. Recall that the UCB heuristic pulls the arm with the largest $UCB(M_i)$:

$$UCB(M_i) = \hat{R}_i + \frac{g(N)}{\sqrt{N_i}}$$

where $\hat{R}_i$ is the average reward for M_i so far, N is the total number of pulls made so far, and N_i is the total number of pulls of M_i so far. For this question, let us use

$$g(N) = \sqrt{2 \ln N}$$

Suppose the two arms, when pulled, will produce the following sequence of rewards:

M_a	0_1	1_3	2_4	2_5	$1 \cdots$
M_b	1_2	0	2	0	$0 \cdots$

For example, if a gambler chooses to pull arms M_a, M_b, M_a, M_a, M_a, we'd get the sequence 0_1,1_2,1_3,2_4, 2_5. (The subscripts are not part of the reward; they are only there to clarify the explanation.)

Simulate the UCB algorithm on this sequence of rewards for 5 timesteps (every time an arm is pulled counts as a timestep; this includes the initialization steps). If there is a tie between two arms's UCB heuristics, choose the arm with the smaller index e.g. choose M_a over M_b if $UCB(M_a) = UCB(M_b)$. After the simulation, please calculate the average reward $\mu_N^{UCB(g(N))}$ and the regret$_N^{UCB(g(N))}$ for the policy when $N = 5$.

Solution: Notice that in the definition of UCB, we pull each arm one time. On timestep 3, 4 and 5, M_b has better score, so we pull M_b, for: M_a, M_b, M_b, M_b, M_b.

Timestep	3	4	5
UCB(M_a)	1.177	1.482	1.665
UCB(M_b)	2.177	1.548	1.961
pulled arm	M_b	M_b	M_b

Table 1: UCB for two arms at timestep 3,4 and 5.

The average reward is 0.6, and the regret for the policy is 3.

C (3 points): In this next question, we'll get some intuition on why this heuristic is called the Upper Confidence Bound, and why we chose $g(N)$ the way we did.

First, we introduce *Hoeffding's Inequality* which states that for independent and identically distributed (iid) random variables $R_1 \dots R_n$ that are either zero or one we have:

$$Pr(\hat{R} + a > \mu) \geq 1 - e^{-2na^2}$$

where $\hat{R} = \frac{1}{n}\sum_{i=1}^{n} R_i$ and μ is the expected value of the random variables. Notice that since they are iid all the random variables have the same expected value.

Suppose we want to guarantee with high probability that the true expected payoff μ is less than $\hat{R} + a$. This makes $\hat{R} + a$ an upper bound. In other words, what is a value for a such that:

$$Pr(\hat{R} + a > \mu) \geq 1 - N^{-4}$$

where N is the number of rounds played in total? a can, and should, depend on n and N.

Solution: Set $1 - e^{-2na^2} = 1 - N^{-4}$ to get $a = \frac{\sqrt{2 \ln N}}{\sqrt{n}}$.

Problem 4: AlphaZero and MCTS (10 points)

In 2016, AlphaGo defeated former Go World Chamption Lee Sedol in a historic game. Follow-up work simplified the AlphaGo algorithm AlphaGo Zero and AlphaZero (basically the same as AlphaGo Zero, but removing Go-specific tweaks). Because we will be looking at games other than Go, we will talk about AlphaZero.

In AlphaZero, a neural network is trained to predict what would be the most promising moves in next rounds. We should note that AlphaZero does not require supervision; it learns a game by itself. More specifically, it uses Monte Carlo Tree Search (MCTS) to generate knowledge about the game by simulating many playouts/rollouts and receiving corresponding evaluations about the playouts/rollouts. Therefore, MCTS is key to AlphaZero's success. By working through this problem, we hope you to have a better understanding of MCTS. In the next homework assignment, we will see more how MCTS is interacting with the neural networks.

A (5 points): If we're playing tic-tac-toe, and the neural network always outputs a uniform probability distribution over all remaining legal moves, what is the probability of MCTS sampling the following game when starting from an empty board?

X	O	X
O	X	O
O	X	O

Solution: Sampling uniformly from legal moves means that when there are m moves available each move has a probability $1/m$ of being sampled. That means that this game has the following probability of being sampled for a fixed order.

$$\frac{1}{9}\frac{1}{8}\frac{1}{7}\frac{1}{6}\frac{1}{5}\frac{1}{4}\frac{1}{3}\frac{1}{2}\frac{1}{1} = \frac{1}{9!} = \frac{1}{362880} \approx 0.000003 = \text{3e-6}$$

However, you can arrive at this same board in many different ways. In particular, you have 5! choices for how to place the Os, and 4! choices for how to place the Xs. That means our final probability is...

$$\frac{5! * 4!}{9!} = \frac{2880}{362880} \approx 0.0079 = \text{7.9e-3}$$

Optionally, you can multiply the above by $\frac{1}{2}$ to account for the random choice of O going first.

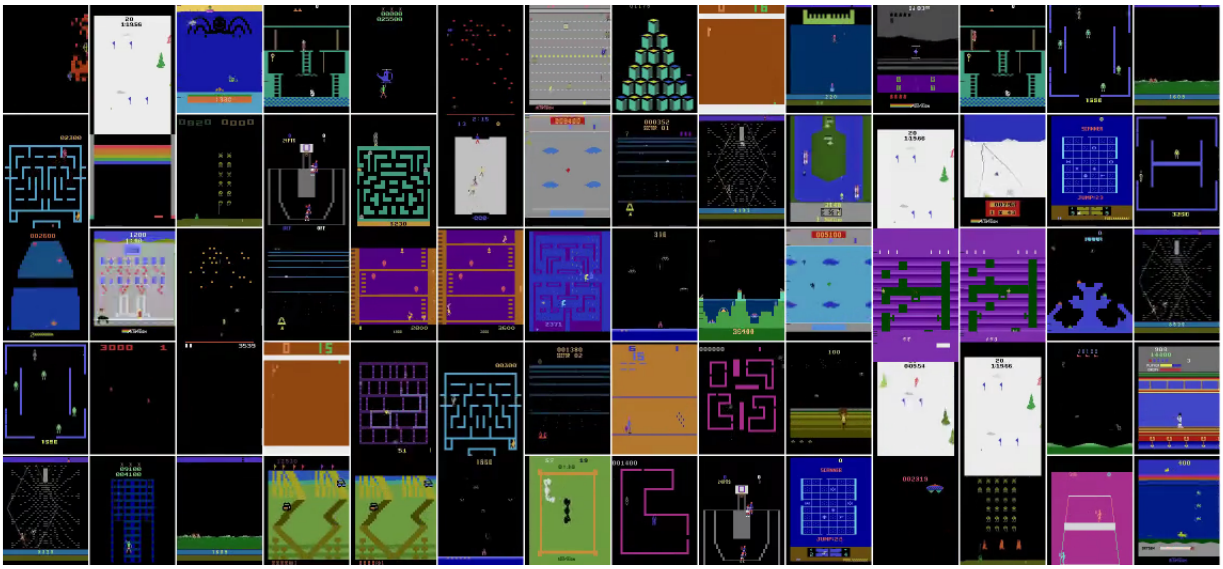
B (5 points): MCTS simulations can often suffer from high variance for being a randomized search. We consider the following scenario: node selection in MCTS is based on rate wins/visits; node a has 78 wins and 100 visits, and node b has 786 wins and 1000 visits. Though node b has a slightly higher winning rate, it is more interesting to select node a and see what happen. Let's define the current node selection function f to be

$$f(w, v) = (w + \epsilon)/(v + 2\epsilon)$$

where w is the number of times the move wins, v is the number of visits, and ϵ is an additive factor subject to tuning. How can you modify f so that node a can be chosen.

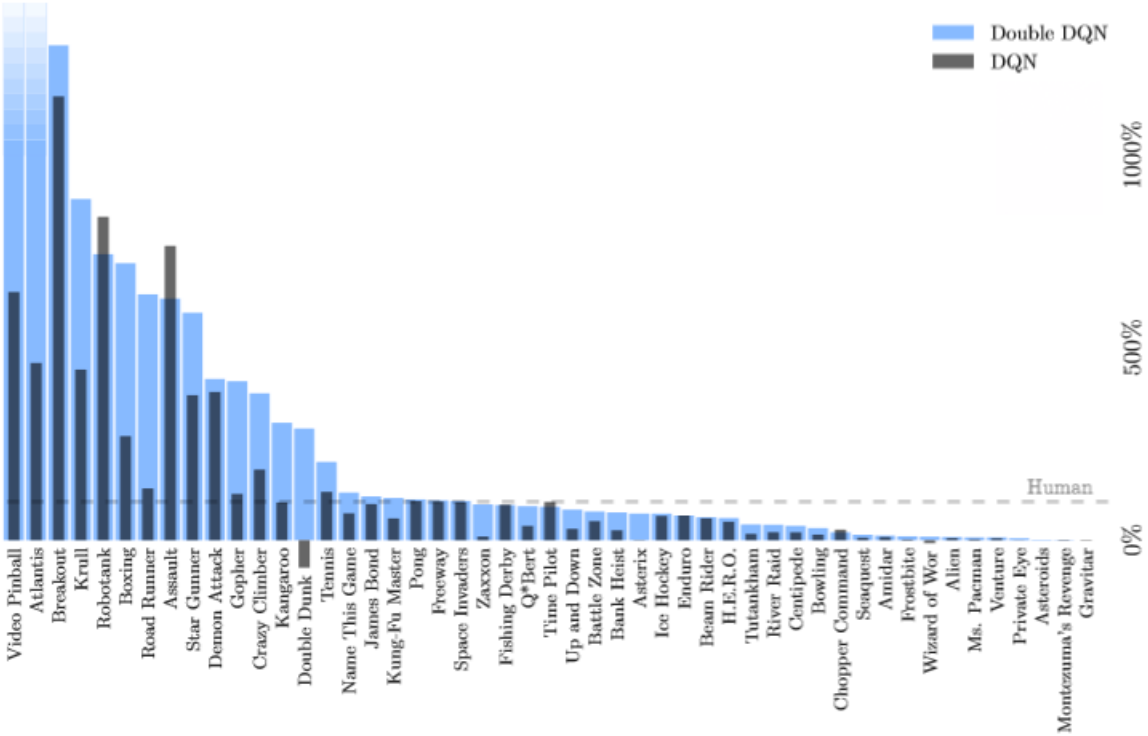
Solution: We account for this variance by balancing exploitation (pick child with best win rate) with exploration (pick lesser visited child). One popular way to do this is to use UCB1 as the tree policy. That is, $UCB1 = f(w, v) + C\sqrt{\frac{lnV}{v}}$, where V is the number of times its parent being visited. Therefore, by increasing C, we can have MCTS to select node a.

Problem 2 Appendix



This problem considers an improvement to the base Q-Learning algorithm called Double Q-Learning. In particular, it uses a simple Markov Decision Process (MDP) to show how double Q-learning is better than the base Q-learning, but the ideas and intuitions behind this improvement also work for more complex tasks such as Atari Games or robot manipulation learning.

Here is a graph to show the difference in performance between the Q-Learning (DQN) and the Double Q-Learning (Double DQN) agents on the Atari benchmark environments shown above.



Notice how the general trend is that the Double Q-Learning agent outperforms the Q-Learning agent.

Tutorial: ϵ-Greedy Policy

Before we delve into this improvement, let us first define a type of behavior policy called an ϵ-greedy policy. Recall that a greedy policy chooses the action that immediately maximizes the expected reward. An ϵ-greedy policy chooses the greedy action with probability $1-\epsilon$ and chooses a random action with probility ϵ making this a near-greedy action selection rule. Another way to think about this is that the policy chooses to *explore* with probability ϵ and chooses to *exploit* with probability $1-\epsilon$. Formally, the ϵ-greedy next action a from state s given a q-value function Q is

$$a = \begin{cases} \max_{a \in A} Q(s,a), & \text{with probability } 1-\epsilon \\ \text{random } a \in A, & \text{with probability } \epsilon \end{cases}$$

Tutorial: Maximization Bias and Double Learning

So what contributes to this difference in performance? Recall the Q-Learning update *step*? for a state s and action a

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(R(s,a,s') + \gamma \max_{a' \in A} Q(s',a')\right)$$

where s' is the next state and r is the reward for taking action a from state s and arriving at the next state s'. Notice how we are estimating what $Q(s,a)$ is by taking the maximum, $\max_{a' \in A} Q(s',a')$, over previously estimated values. This bootstrapping of estimates could lead to a significant bias towards positive estimates. For example, let's say for a given state s, the true actions values, $Q^*(s,a)$, are 0 and the estimated values, $Q(s,a)$, are a mix of positive and negative values. The true max is $\max_{a \in A} Q^*(s,a) = 0$ while the estimated max is $\max_{a \in A} Q(s,a) > 0$. This positive bias is often called the **maximization bias**.

In this problem, we will explore how maximization bias can harm the performance of Q-Learning. But first, how can we design an algorithm to avoid this bias? One way to view this problem is that it is caused by using the same samples to both estimate the best action and estimate its value. To solve this, we would want the samples to only affect either the choosing of the action, $\arg\max_{a' \in A} Q(s,a')$, or the update of the value, $Q(s,a)$. A way to do this would be to divide the samples into two sets and learn two independent estimates, $Q_1(s,a)$ and $Q_2(s,a)$, for the true action values for all $a \in A$. Then we could use one estimate, lets say $Q_1(s,a)$, to get the best action, $a^* = \arg\max_{a' \in A} Q_1(s,a')$, and use the other estimate to provide the estimate of the true value, $Q_2(s,a^*) = Q_2(s, \arg\max_{a' \in A} Q_1(s,a'))$. This process can be repeated with the Q_1 and Q_2 swapped. This is the main idea behind **Double Q-Learning**. We would then have the following update rule

$$\begin{aligned} a' &= \arg\max Q1(s',a) \\ Q1(s,a) &= (1-\alpha)Q1(s,a) + \alpha\left(R(s,a,s') + \gamma Q2(s',a')\right) \\ &\textbf{or} \\ a' &= \arg\max Q2(s',a) \\ Q2(s,a) &= (1-\alpha)Q2(s,a) + \alpha\left(R(s,a,s') + \gamma Q1(s',a')\right) \end{aligned}$$

At each update step, we would choose either of these update steps randomly with equal probability. When actually choosing the action to take with the ϵ-greedy policy, we use the sum of the two q-value tables: $Q1+Q2$.

Performance Experiment using Example MDP

The MDP below will illustrate how maximization bias can harm the performance of control algorithms like Q-learning. The MDP below has two non-terminal states **A** and **B** and two terminal states **C** and **D**. Every single episode will start in state **A** with the choice to either go **left** or **right** both with a reward of 0. From state **B**, there are 15 possible actions all with a reward drawn from a normal distribution with mean -0.1 and variance 1. Thus, we have that the expected reward from state **A** when going **right** is 0 and when going **left** is -0.1, making **left** always the wrong choice.