Free-response questions (5 points)

1. (0.5 points) Tic-Tac-Toe

Here we will formulate Tic-Tac-Toe as an environment in which we can train a reinforcement learning agent. You will play as X's, and your opponent will be O's. Two-player games such as Tic-Tac-Toe are often modeled using game theory, in which we try and predict the moves of our opponent as well. For simplicity, we ignore the modeling of the opponent moves and treat our **opponent's actions as a source of randomness within the environment**. Assume you always go first.

a. (0.25 points) What are the states and actions within the Tic-Tac-Toe reinforcement learning environment? How does the current state affect the actions you can take?
b. (0.25 points) Design a reward function for teaching a reinforcement learning agent to play optimally in the Tic-Tac-Toe environment. Your reward function should specify a reward value for each of the 3 possible ways that a game can end (win, loss, or draw) as well as a single reward value for actions that do not result in the end of the game (e.g., your starting move). For actions that do not end the game, should reward be given to your agent before or after your opponent plays?

Answer:

a. **The states are the states of the "board", i.e, occupancy of each cell. At any given state, Actions are drawing a X in one of the unoccupied cells. The current state has a set of cells that are unoccupied, that will be the action space, which determines which actions I can take.**

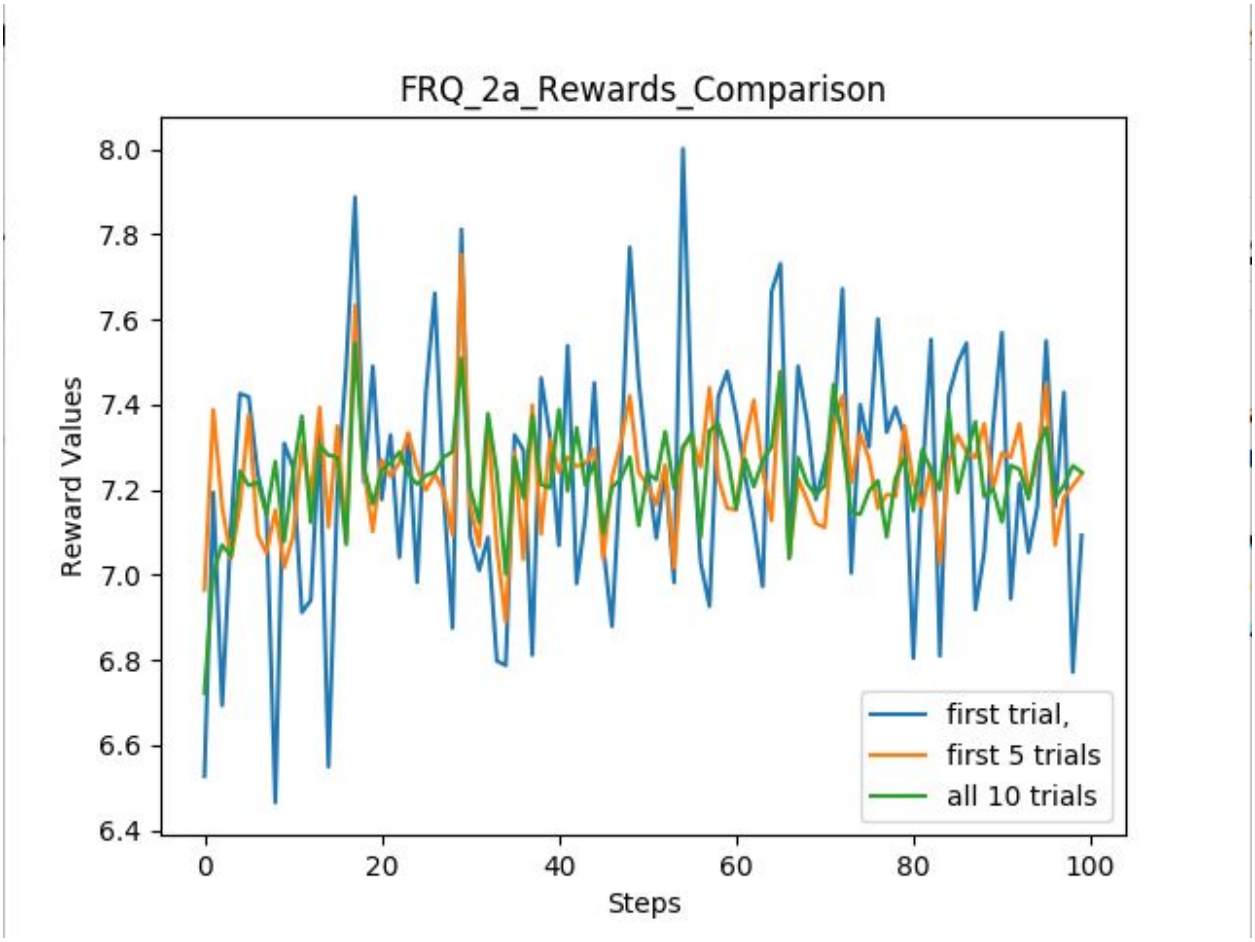
b. **Rewards:**
   i. **For a win: 100**
   ii. **For a loss: -100**
   iii. **For a draw: 10**
   iv. **Actions do not result in the end of the game: 0**

Rationale: we want to have high rewards for wins and losses so the agent is able to draw a clear distinction between them. We also want to slightly reward a draw since if we are not able to win, we want a draw, which is also reasonable. But the reward for a draw should be much lower than a win. We do not want to reward actions that do not end the game, as this will result in the agent moving less directly to the winning or drawing states, and the agent will more likely move in a random manner.

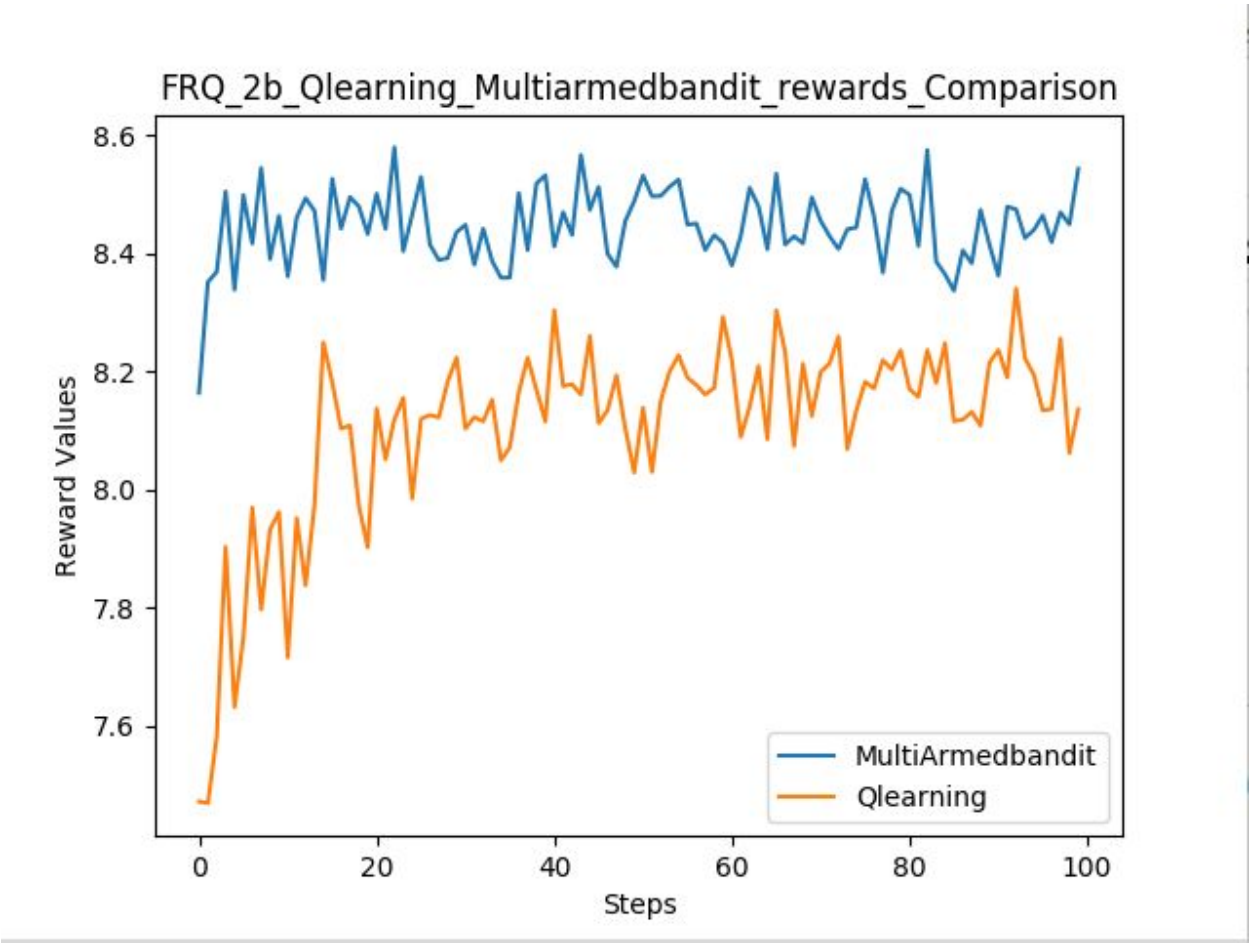
2. (1.5 points) Bandits vs Q-Learning

Here we will setup an experiment to train both the MultiArmedBandit and QLearning models on the SlotMachines-v0 environment. Use the **default values** for both MultiArmedBandit (epsilon and discount) as well as SlotMachines-v0 (n_machines, mean_range, and std_range). You should use the **gym.make** function to create the SlotMachines-v0 environment. See experiments/example.py for an example of this.

a. (0.25 points) Train 10 MultiArmedBandit learners, each for **100,000** steps. We will refer to **each** of the 10 independent training sessions as one trial. Create one plot with 3 lines on it. Plot as the first line the **rewards array** (the second return value from the fit function you implemented in the code; this should be **a 1D numpy array of length 100**) from **the first trial.** For the second line, **average** the rewards arrays learned in the **first 5 independent trials**. The resulting averaged rewards array should be the element-wise average over the first 5 trials of MultiArmedBandit and should also be of length 100. For your third line, repeat what you did to create the second line, but this time use all 10 trials. Label each line on your plot with the number of trials that were averaged over to create the line.



b. (0.25 points) Now train 10 QLearning learners, each for **100,000** steps, on the SlotMachines-v0 environment. Plot the averaged **QLearning** rewards array over **all 10 trials** that used QLearning and the **averaged MultiArmedBandit** rewards array over all 10 trials that used MultiArmedBandit on the same plot. Make sure to label each line in your plot with its associated learner.



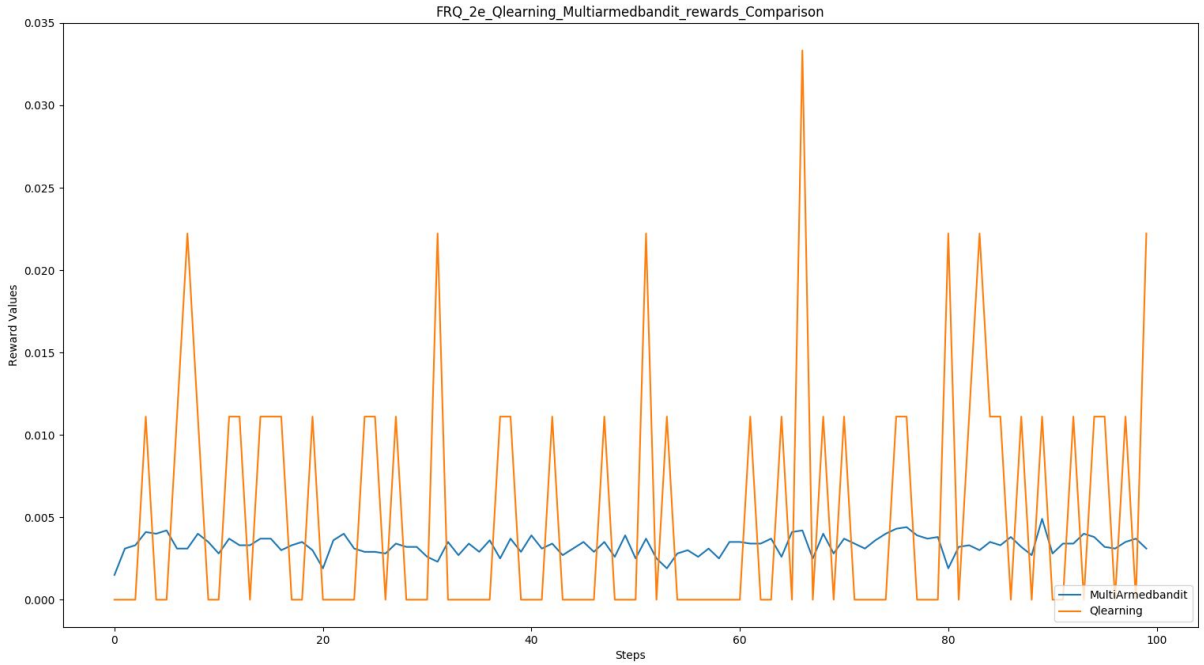
c. (0.25 points) Look at your plot from question 2.a. Why is it important that we average over multiple independent trials for the same learner? What happened to the jaggedness of the line (the variance) as the number of trials increased?

Answer: Averaging over multiple independent trials for the same learner is important as it can reduce the randomness caused by any single independent trial. As the number of trials increases, the jaggedness of the line decreases.

d. (0.25 points) Look at your plot from question 2.b. How does the reward obtained by the two learners differ on the SlotMachines-v0 environment? Does one learner appear to be significantly better (i.e., obtain higher reward) than the other?

A: The average reward at each step obtained by the multiarmedbandit learner is significantly higher than the QLearning one. Also, the multiarmedbandit learner has more "stable" rewards.

e. (0.5 points) Make a plot like your plot from question 2.b, but this time using data from training both learners for 10 trials (100,000 steps per trial) on the FrozenLake-v0 environment. Include your plot and answer the following questions:



(0.25 points) How does the reward obtained by the two learners differ on the FrozenLake-v0 environment? Does one learner appear to be significantly better (i.e., obtain higher reward) than the other?

Answer: the reward from multi-armed bandit is quite stable over the steps, which means there are no significant improvement over time. However, Qlearning's reward, as time goes on, will be come higher and higher, as can be seen by the decrease in 0 reward time. From the plot, we can see that Q learning is better than multi-armed bandit at least at the final steps.

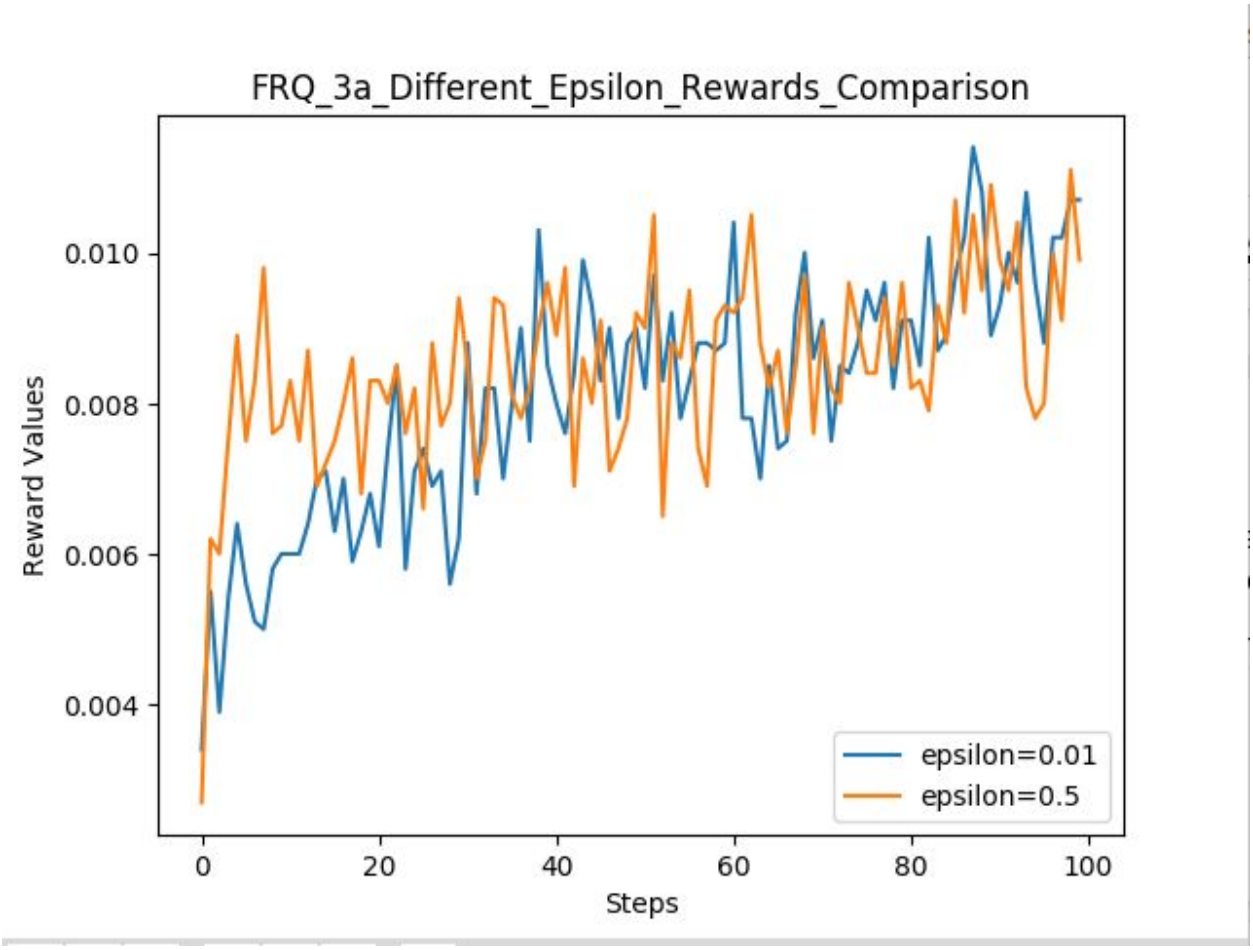
(0.25 points) The best action to take within the FrozenLake-v0 is state-dependent. For example, there are some states in which moving to the right will cause you to fall into the water. In these

states, moving right is a bad choice of action. Look at your plot from question 2.e. You should observe that one of your learners performed poorly on FrozenLake-v0. Identify which learner was less able to learn the environment (i.e., underfitting) and describe why that underfitting occurs due to the limited hypothesis space of that learner.

Answer: The multi-armed bandit learner is less able to learn the environment. The reason is that its value function at a given state and state does not consider the following states' values. In some (state, action) pairs, the action always yields zero reward (i.e, the next state will not be a hole or goal). This is similar to focusing only on the present, and being blind to the future steps. Therefore the learner does not know what happens after taking the action. Then, the agent has limited hypothesis space and is less able to learn the environment.

3. (1.5 points) Exploration vs Exploitation

a. (0.5 points) Setup an experiment to train the QLearning model on the FrozenLake-v0 environment for two values of epsilon: **epsilon = 0.01** and **epsilon = 0.5**. For each value, train **10 QLearning** learners, each for **100,000** steps (i.e., steps=100000), with the default **discount rate of 0.95**. See test_q_learning for an example of how to run your model on an OpenAI Gym Environment. For each value of epsilon, plot the rewards array (averaged over all 10 trials). **All values of epsilon should be plotted on the same axis**. Label each line in your plot with its associated epsilon value. Include your plot and answer the following questions:

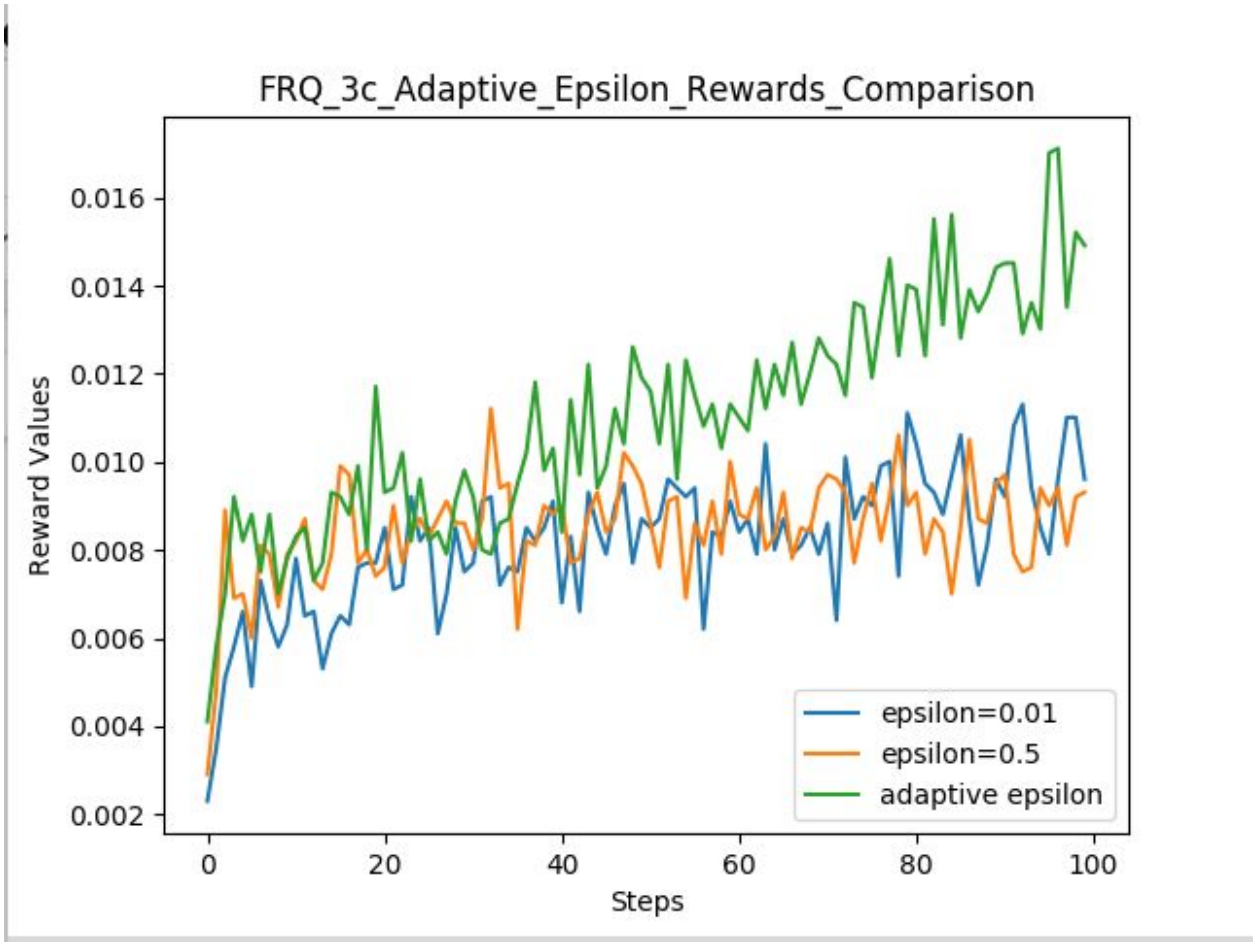


b. (0.25 points) For which value of epsilon is the averaged rewards of the QLearning learner maximized on the FrozenLake-v0 environment? Note: we are asking for the value of epsilon that produces the largest single reward value at any point along the x-axis throughout your plot. An aside: note that we are only evaluated epsilon during the training process. In practice, when choosing a "best" value of epsilon, it is important to evaluate which value of epsilon maximizes reward during **prediction**. You can evaluate this by **running your predict function for**, e.g., 1000 episodes for each value of epsilon and determining which value of epsilon produced the highest average reward. For simplicity, we are omitting this step.

Answer: Epsilon = 0.5 yields the highest rewards as the learner does more exploration at the beginning, and learns about the environment faster.

c. (0.25 points) Implement the _adaptive_epsilon function in q_learning.py to update the value of epsilon as training progresses. Let progress be the fraction of training steps currently completed. Implement _adaptive_epsilon so that the value of epsilon at step s is (1 - progress) * self.epsilon, where self.epsilon is the initial epsilon value. Set the initial epsilon **value to be 0.5**.

Train 10 QLearning learners with the default discount rate of 0.95 using your _adaptive_epsilon function (**i.e., adaptive = True**). Plot the averaged rewards array for your adaptive epsilon on the same plot that you created in question 3.a. Include your plot and answer the following questions.



d. (0.25 points) Compare your results to your plot from part a. How **does your average reward compare** to the average reward for static (non-adaptive) values of epsilon? What does this say about the tradeoff between exploration and exploitation during training?

Answer:
The adaptive epsilon method yields the highest average reward. This means at the initial steps, more exploration should be conducted to learn about the environment faster. As average reward becomes stable over time, exploitation should be conducted more as the state-action values are already close to their true values.

e. (0.25 points) Consider the following modification to the FrozenLake-v0 environment: every n steps, the location of the holes and the goal **move to random locations**. Assume the goal is

still accessible. How should **epsilon change over time** to accommodate the stochasticity in this environment? Consider both the case when the value of n is known to you and when it is not.

Answer:
Espilon should be reset to its initial value every n steps, when n is known. When n is not known, first develop a probabilistic model of the number of steps for changes to occur (i.e, Gaussian distribution). Then, come up with a sequence of step ranges, for resetting epsilon, based on the distribution.

4. (0.5 points) On-Policy vs Off-Policy

a. (0.25 points) Describe in your own words the difference between on-policy and off-policy learners. Provide an example of each.

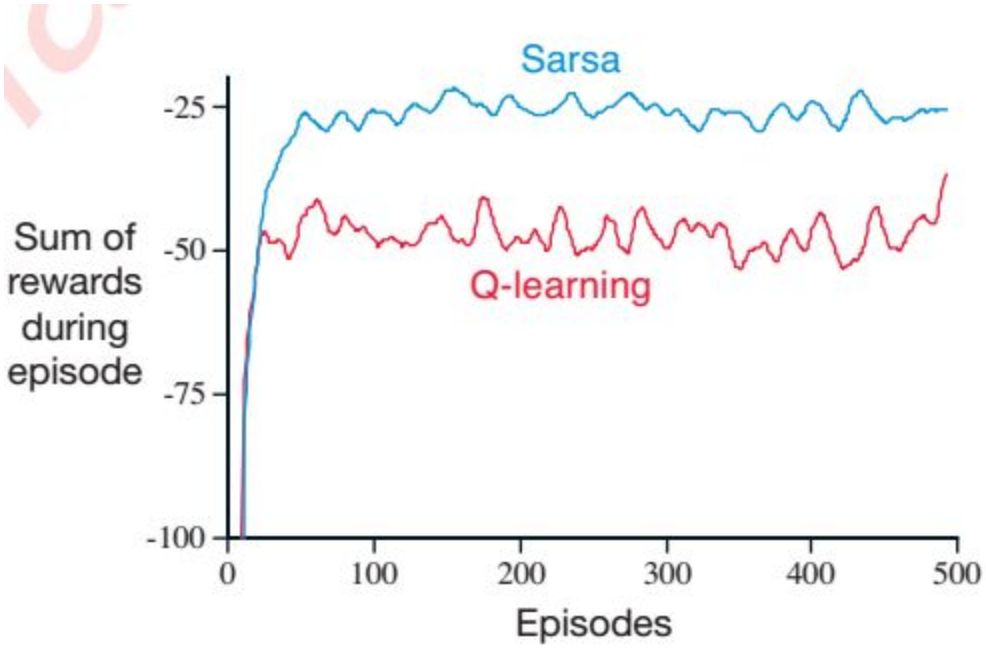
Answer: on policy learners will learn the Q(state, action) values based on the given policies, while off-policies will learn Q(state, action) values based on different actions at each state, which does not require a policy

b. (0.25 points) Consider the following environment: your agent is placed next to a cliff and must get to the goal. The shortest path to the goal is to move along the edge of the cliff. There is also a longer path to the goal that requires the agent to first move away from the cliff, and then towards the goal. The reward for reaching the goal is 100 points, and the reward for falling of the cliff is -1000 points. Every move we make incurs a reward of -1. Assume we use an epsilon-greedy policy for exploration. If we would like to learn the shortest path, should we use an on-policy or off-policy algorithm? Explain why. Note: reading chapter 6 of Sutton & Barto will help you answer this question.

Answer:
On-policy algorithm should be used. First, off-policies, such as Q_learning, will involve random actions as part of its epsilon-greediness. Those random actions might make the agent go off of the cliff.

Second, on-policies, such as Sarsa, considers action selection and learns safer paths than Q_learning, even though it might take longer. In this application, the penalty for death is much outweighs the reward for reaching the goal. Therefore safer paths can yield higher rewards on average.

An excerpt image from Sutton & Barto shows the performance of the two algorithms.



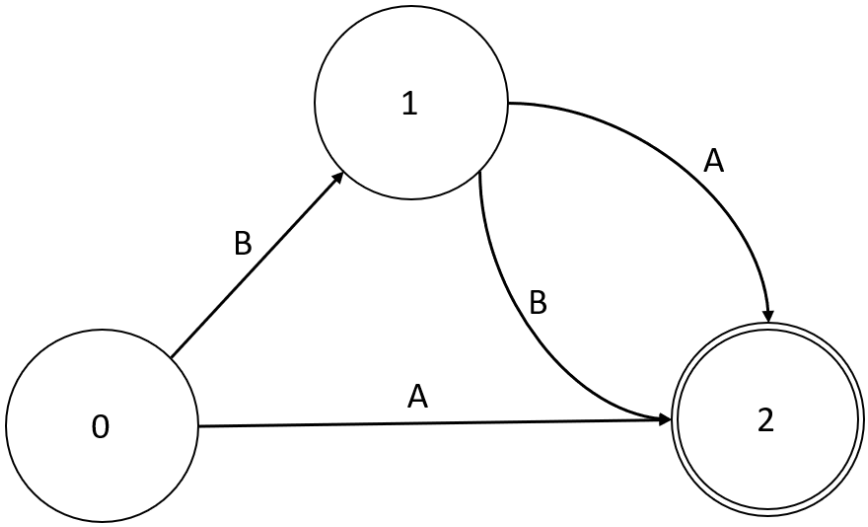
5. (1 point) Markov Decision Processes

a. (0.25 points) Consider the following environment: you are designing a trash-collecting robot in a grid-like environment. Trash is located at varying distances around a single trash can. The robot can hold an **unlimited number** of pieces of trash. The possible actions are to move in one direction along the grid, pickup a piece of trash, or deposit all currently held trash in the trash bin. The robot must be in the same grid square as a piece of trash to pick it up, and must be in the same square as the trash bin to deposit the trash. The robot is rewarded 10 point for each piece of trash it collects, and 100 points for each piece of trash it places in the trash bin. Each movement incurs a reward of -1. Assume we modify the environment so that there is **a 10% chance** at each time step of the robot suddenly failing. If the robot fails, it can no longer increase its reward. How does adding a 10% chance of the robot suddenly failing affect the optimal behavior of the **robot?** Describe how to modify the MDP formulation to account for these spurious failures.

Answer: if the robot can suddenly fail under any circumstance, i.e, no matter what the robot's next action is, there is an equal chance of failing, then the robot will try to maximize "instant reward" (e.g, collect as much garbage near itself as it can) and puts less emphasis on reaching the goal than no failing at all.

In the MDP, we can modify each state with 90% "live" probability and 10% "death" probability. So everytime the robot have state transitions, there is always a 10% probability failing.

b. (0.75 points) Consider the following environment represented as a directed graph, in which circles represent states, double circles represent the goal, and arrows represent actions. Assume you start at state 0. Assume all actions are deterministic. Transitioning to state 1 produces a reward of 0, while transitioning to state 2 produces a reward of 1. Markov Decision Process



I. (0.25 points) What are the optimal state-values and state-action-values for this environment?

Answer: The optimal state-value function V*(s) is the maximum value function over all policies. It's the best possible solution of an MDP.

Optimal State values:

V(0) = 1

V(1) = 1

State action values: (S,A)

Q(0,A) = 1, Q(0, B) = 1

Q(1,A) = 1, Q(1,B) = 1

Ii. (0.25 points) What is the optimal policy for this environment?

Answer:

Optimal policy: There is no single optimal policy as all policies are optimal.

Pi(0) = A or B

Pi(1) = A or B

Iii. (0.25 points) Assume we introduce a discount factor of 0.95 into our value functions. Determine the new values of the state-value and state-action-value functions as well as the new optimal policy. Describe the effect of the discount factor on the optimal policy.

Optimal State values:

V(0) = 0.95

V(1) = 1.0

V(2) = 0

State action values: (S,A)

(0,A) = 1.0, (0, B) = 0.95

(1,A) = 1.0, (1,B) = 1.0

Optimal Policy

Pi(0) = A

Pi(1) = A or B

Effect of the discount factor on the optimal policy:

The value function of the discount factor is defined as :

$$V^{\pi}(s) = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

Therefore, 0 -> 1 -> 2 will yield a value of 0.95*0.95, which is less than 0.95.