Free-response questions (5 points)

To answer some of these questions, you will have to write extra code (that is not covered by the test cases). You may include your experiments in new files in the `experiments` directory. See `experiments/example.py` for an example. You can run any experiments you create within this directory with `python -m experiments.<experiment_name>`. For example, `python -m experiments.example` runs the example experiment.

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?

The State Set (S)

the state set of player X contains all the boards when it is his turn to play, so in our case the state set contains all the boards that have the same number of X’s and O’s. For example, the empty board is in the state set. In addition, all the boards when the game is over, are also in the state set and they are called end states.

We will write a state as a 9 string that contains the numbers 1–9 to represent the empty spots in the board from up-left to down-right, and X’s and O’s instead of the numbers in the spots where the players played. For example, the string X234O6789 represents a state where X played in the upper-left corner and O played in the center.

The Action Set (A)

the action set contains all the options of player X to play, that is, all the numbers in a specific state.

Note here that the board after player X plays his turn is not a state in player X's state set because X cannot perform an action on this board (it is O's turn). The new state that the action will lead to will be the board after player O played his turn.

The current state will show you what rest actions you can take. For example, if the state is X234O6789, then your can take action 2, 3, 4, 6, 7, 8, or 9. 1 and 5 are not allowed.

- 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?

  Reward: we will define it as 1 if X won, -1 if O won, 0.5 if it is a tie(draw) and 0 otherwise.

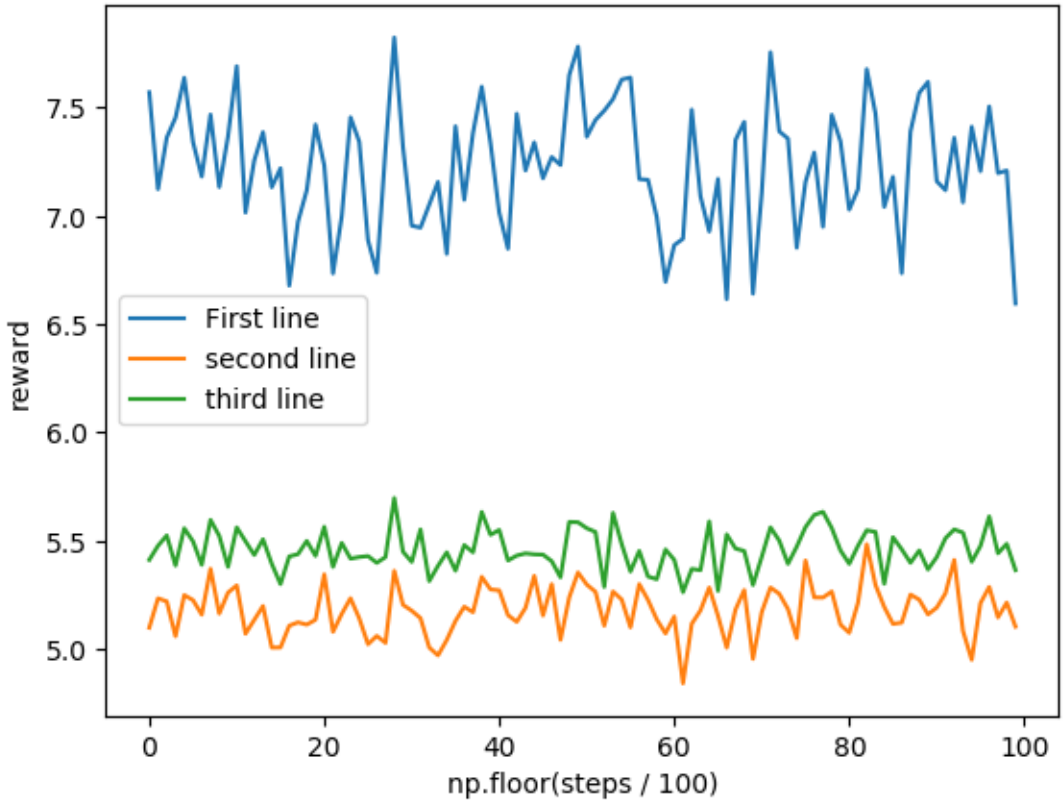
  Reward should be given after opponent plays.

2. (1.5 points) Bandits vs Q-Learning

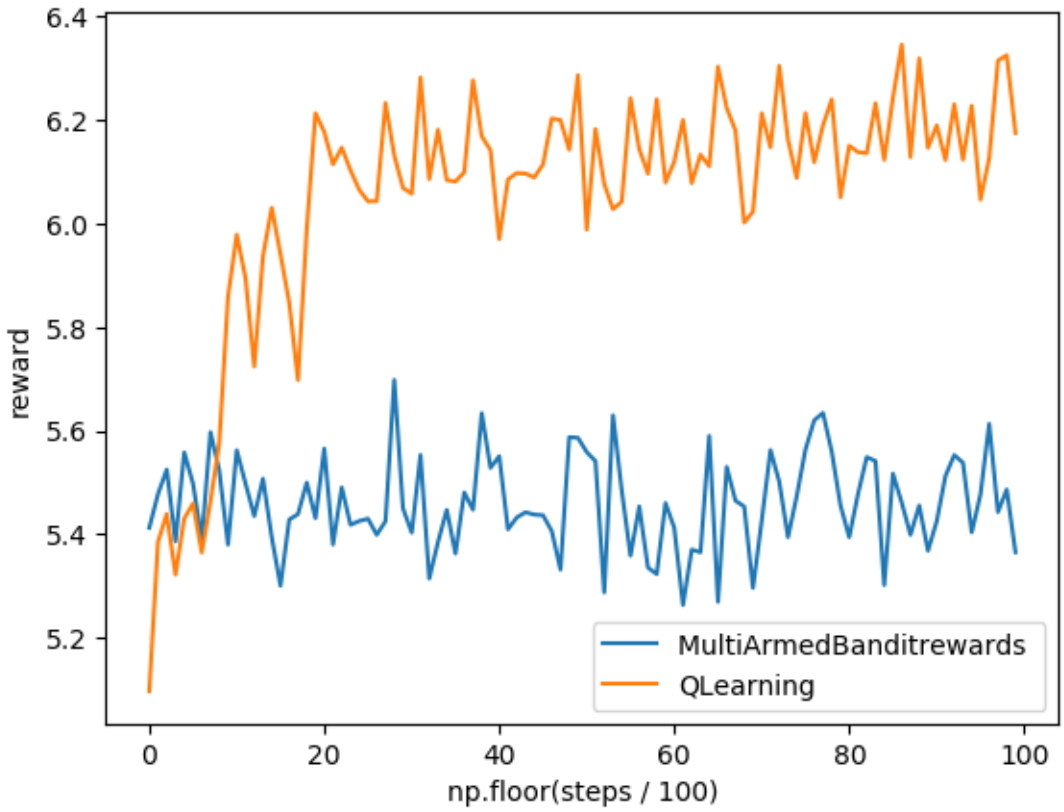
Here we will setup an experiment to train both the `MultiArmedBandit` and `QLearning` models on the `SlotMachines-v0`environment. Use the default values for both `MultiArmedBandit` (`epsilon` and `discount`) as well as `SlotMachines-v0`(`n_machines`, `mean_range`, and `std_range`). You should use the `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 trainings 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 `MultiArmedBanditrewards` 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?

  In order to avoid bias, we should average over multiple independent trials for the same learner. We use epsilon greedy. We might have bias in one trial. When the number of trials increases, the jaggedness decreases.

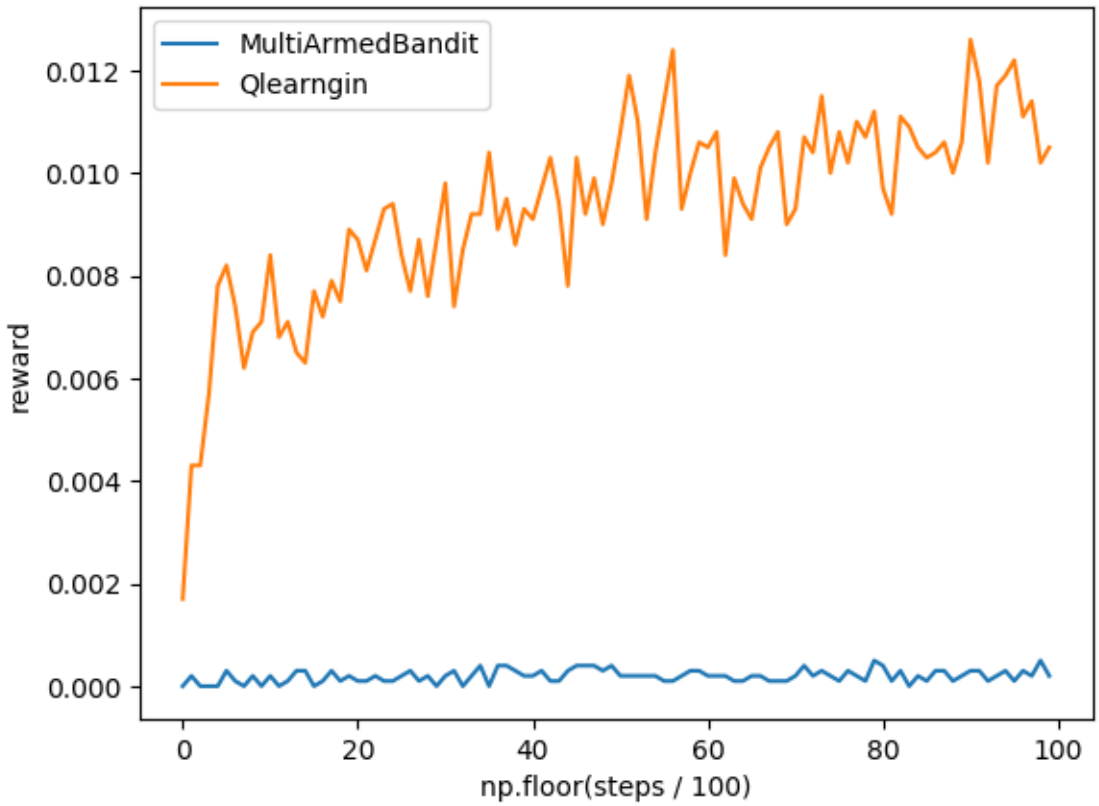
- d. (0.25 points) Look at your plot from question 2.b. How does the reward obtained by the two learners differ on the `SlotMachines-v0` environment? Does one learner appear to be significantly better (i.e., obtain higher reward) than the other?

  Qlearning is much better than multiarmedbandit.

- 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:

- a. (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?

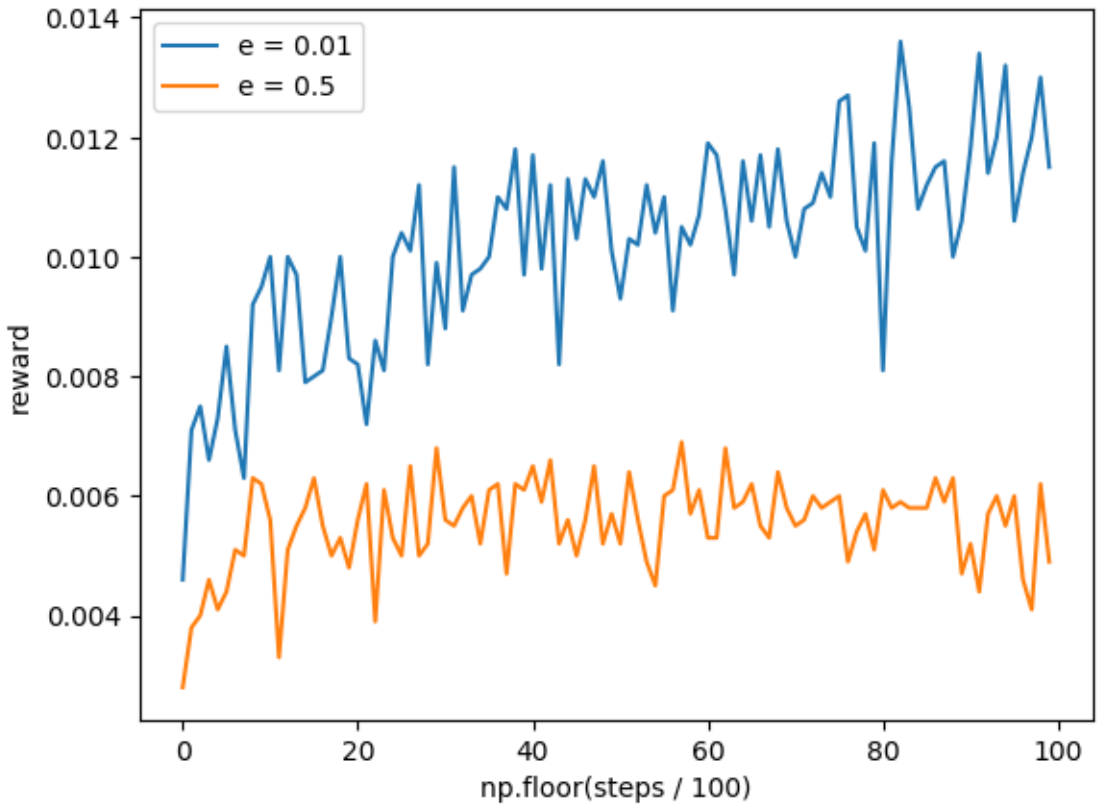


- b. (0.25 points) The best action to take within the `FrozenLake-v0` is state-dependent. For example, there are some states in which moving to the right will cause you to fall into the water. In these states, moving right is a bad choice of action. Look at your plot from question 2.e. You should observe 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.

`MultiArmedBandit` performed poorly. Since `MultiArmedBandit` use alpha (1 / #(action(a))) without considering the state. Therefore, when updating table, alpha doesn't consider state and action. Therefore, when one action might be good at most time, the learner will try to choose this action more no matter the state.

3. (1.5 points) Exploration vs Exploitation

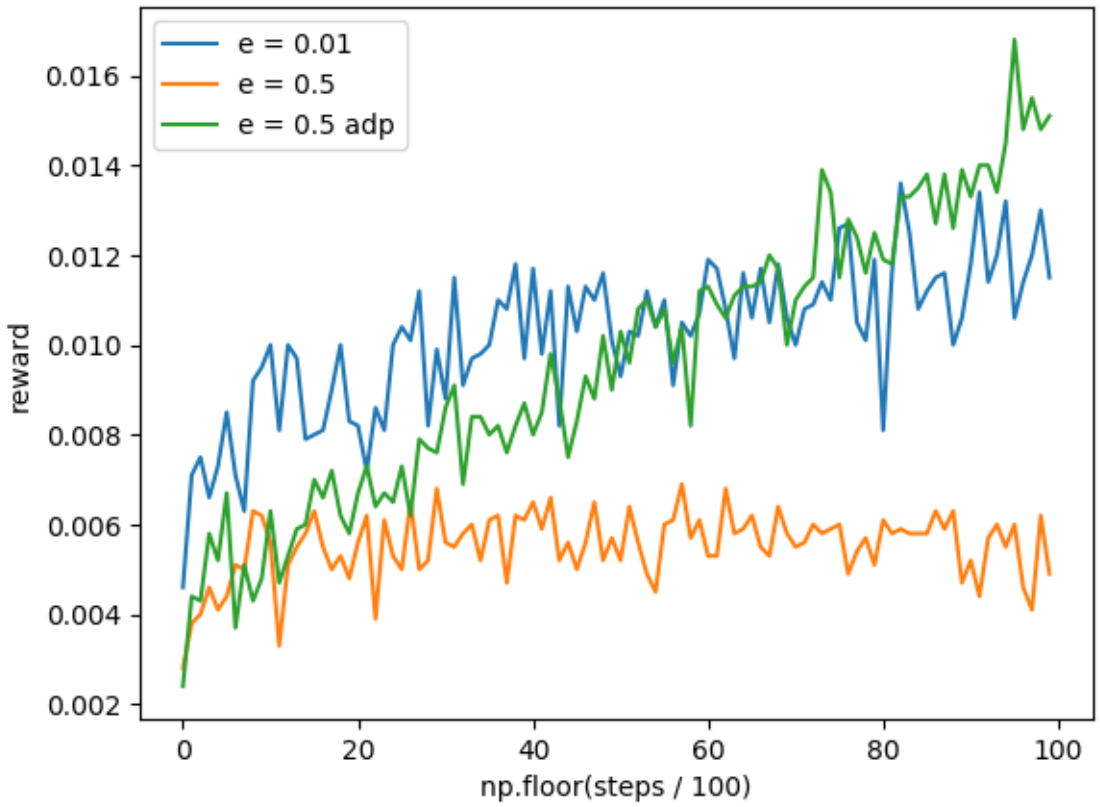
- a. (0.5 points) Setup an experiment to train the `QLearning` model on the `FrozenLake-v0` environment for two values of `epsilon`: `epsilon = 0.01` and `epsilon = 0.5`. For each value, train 10 `QLearning` learners, each for 100,000 steps (i.e., `steps=100000`), with the default discount rate of 0.95. 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.*

  e = 0.01 is better.

- 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?

  Smaller epsilon, higher reward. Spending more time on exploitation and less time on exploration will get a better reward.

- 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.

  If n is known, every n steps epsilon should become big enough and then gradually decreases.

If n is unknown, we should keep epsilon always not too big neither too small. Spending certain time on exploration.

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.

  The difference of on-policy and off-policy learners is that the learner choose the action assuming that greedy policy were followed or the current policy continues to be followed.

  Q-learning is off-policy and SARSA is on-policy.

  The reason that Q-learning is off-policy is that it updates its Q-values using the Q-value of the next state s′ and the greedy action a′. In other words, it estimates the return (total discounted future reward) for state-action pairs assuming a greedy policy were followed despite the fact that it's not following a greedy policy.

  The reason that SARSA is on-policy is that it updates its Q-values using the Q-value of the next state s′ and the current policy's action a′′. It estimates the return for state-action pairs assuming the current policy continues to be followed.

  The distinction disappears if the current policy is a greedy policy. However, such an agent would not be good since it never explores.

- 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.

  We should use on-policy algorithm.

  Q-learning(off-policy) learns values for the optimal policy, that which travels right along the edge of the cliff. Unfortunately, this results in its occasionally falling off

the cliff because of the "epsilon-greedy action selection. Sarsa(on-policy), on the other hand, takes the action selection into account and learns the longer but safer path through the upper part of the grid. Although Q-learning actually learns the values of the optimal policy, its online performance is worse than that of Sarsa, which learns the roundabout policy. Of course, if epsilon were gradually reduced, then both methods would asymptotically converge to the optimal policy.

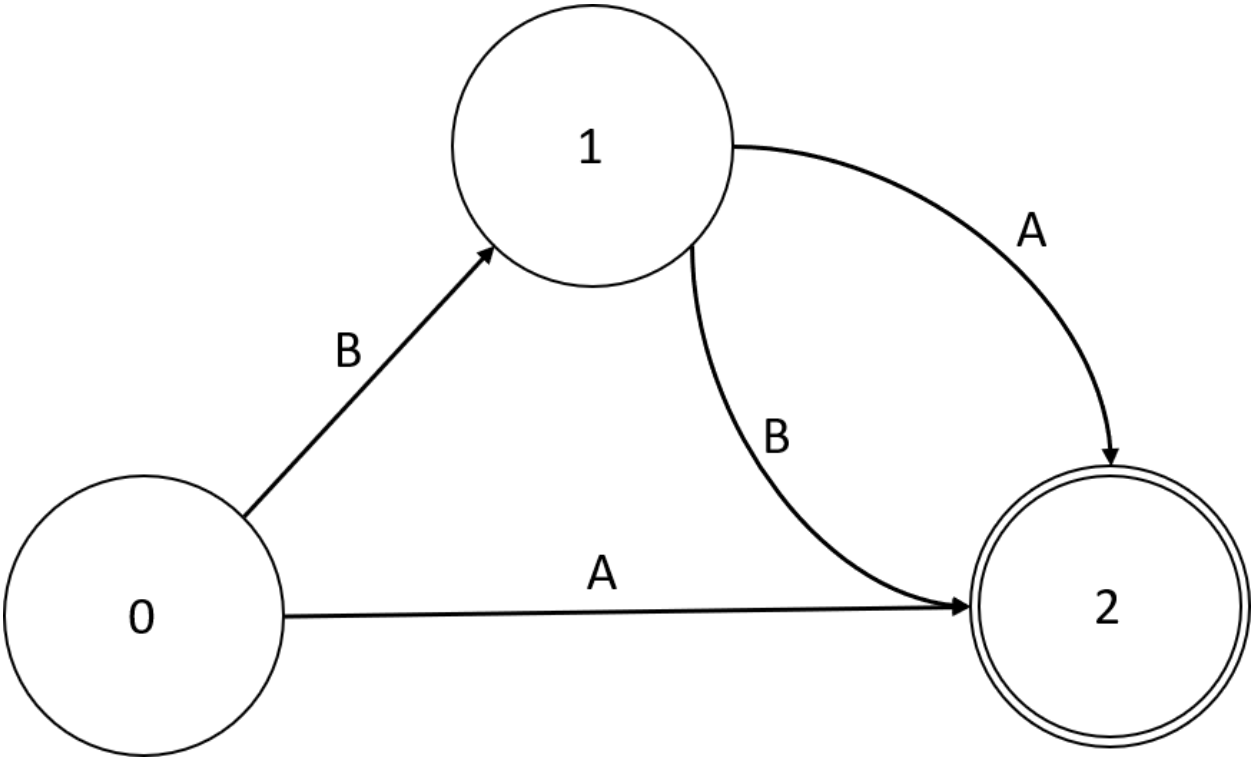
5. (1 point) Markov Decision Processes

- 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.

  We have to times 0.9 for every steps. That is, the expected reward for first step * 0.9, the expected second reward * 0.9 * 0.9 and so on. Then, MDP formulation will also consider the probability that robot fails.

- 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.



- a. (0.25 points) What are the optimal state-values and state-action-values for this environment?
  - V(0) Q(0, A)
  - V(1) Q(1, B)
  - V(1) Q(1, A)
  - V(0) = 1
  - V(1) = 1
  - Q(0, A) = 1
  - Q(0, B) = 1
  - Q(1, A) = 1
  - Q(1, B) = 1

- b. (0.25 points) What is the optimal policy for this environment?
  - Choose whatever you like and you will have the same reward.
  - Pi(0) = A or B , pi(1) = A or B

- c. (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.

V(0) = 1

V(1) = 1

Q(0, A) = 1

Q(0, B) = 0.95

Q(1, A) = 1

Q(1 , B) = 1

Pi(0) = A, Pi(1) = A or B

Because of the effect of the discount factor, choosing B in state 0 is no more a good choice. It reduces the reward inherit from state 1.