CSE 4618: Artificial Intelligence Lab

Lab 5 (Markov Decision Process)

Name: K.M.Tahlil Mahfuz Faruk

ID: 200042158

Islamic University of Technology, OIC

Gazipur, Bangladesh

Apr 28, 2024

Task-1(Value Iteration)

Introduction

In this task we have to implement the value iteration algorithm.

Analysis of the problem

- To implement the value iteration algorithm we need to calculate the q-values for each of the states for particular actions.
- For a particular state, the value would be the maximum of the q-values if any q-value exists or action exists.

Explanation of the solutions

- First we need to iterate through a specified number of iterations. Each iteration updates the value function approximation towards the optimal one.
- We need to initialize a counter to keep track of the states and update their values accordingly.
- Now we need to go through each state. States can have multiple actions. According to those actions, we calculate the q-values for each particular action.
- To calculate the q-value we use this formula,

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right]$$

- Here, we have :
  - A transition function T(s,a,s')
    - Give us the probability of getting to the state s'
  - A reward function R(s,a,s')
    - The reward that we get after getting to the next state.
  - A discount factor γ
    - Discount factor helps us to determine the policy
    - Whether to take the shorter path(Increase the γ)
    - Whether to take the longer path(Decrease the γ)
    - Encourages convergence
  - The value of the previous state V_k(s')

    - The value of the next state V_{k+1} (s')
- This formula will calculate the q-values.
- For a particular value of a state we take the maximum q-value of all the action and take the maximum value as the value of the state. We now have the values of all the states.
- We need to select the best action now. Using argMax() of the q-values we will get the key for the maximum q-value. That will be our optimal action or policy.

Challenges and Solutions

No significant challenges were faced for this problem.

Behavior for Different Hyperparameters

- Discount Factor(γ):
    - Low Discount Factor (Close to 0):
        - With a low discount factor, the agent prioritizes immediate rewards over future rewards.
        - Might not be optimal in the long run.
    - High Discount Factor (Close to 1):
        - With a high discount factor, the agent values future rewards almost as much as immediate rewards.
        - Tends to choose actions that lead to higher cumulative rewards over time.
- Iterations:
    - High Number of Iterations:
        - Increasing the number of iterations allows the agent to perform more updates and potentially converge to a more accurate value function estimate.
        - However, a higher number of iterations also increases computational time and resources required for training.
    - Low Number of Iterations:
        - A lower number of iterations may result in quicker but potentially less accurate convergence of the value function estimate.
        - However, fewer iterations reduce computational time and resources required for training.

Interesting Findings

No interesting findings for this problem.

Task-2(Bridge Crossing Analysis)

Introduction

In this problem, we need to set the numerical parameters such as Discount Factor(γ) and Noise so that the optimal policy causes the agent to attempt to cross the bridge.

Analysis of the problem

- Discount Factor:
    - Discount determines how much the agent cares about rewards in the distant future relative to those in the immediate future.
- Noise:
    - Noise refers to how often an agent ends up in an unintended successor state when they perform an action.

Explanation of the solutions

- We have to cross the bridge avoiding the cliff.
- Observing the states, we can say we have to take the shortest path to successfully cross the bridge. Else we will end up falling.
- For the **Discount Factor**, we need to set it close to 1(0.9). It will encourage the agent to take the shortest path. Also it will prioritize the immediate rewards rather than the future rewards.
- For **Noise**, we set it very close to 0(0.01). Because any unintended action will end up falling into the cliff. Taking 0.01 signifies that we have 1% chance of unintended action over intended action. That means 99% times we will get our intended action from the agent.
- It will make the agent cross the bridge successfully.

Challenges and Solutions

No significant challenges were faced for this problem.

Behavior for Different Hyperparameters

- If we took the discount factor close to 0 then it would have encouraged the agent to take a longer path and prioritize the future rewards.

- If we took noise factors close to 1, the agent's probability to take an unintended action would have increased drastically. So if it wanted to go straight it might take right or left and fall into the cliff.

Interesting Findings

No interesting findings for this problem.

Task-3(Policies)

Introduction

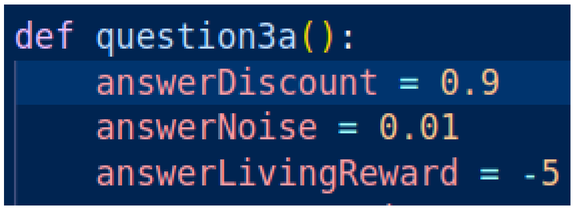
In this problem, we need to set the numerical parameters such as Discount Factor(γ), Noise and living reward so that the optimal policy exhibits the agent to perform 5 given behaviors.

Analysis of the problem

- We need the increase or decrease discount factor to get the bottom and top row of the grid respectively
- We need to increase or decrease the noise for creating or omitting the chances of unintended actions.
- Living rewards need to be positive or negative according to the policy that we need to fulfill.
- Higher living reward will make the agent live longer.
- Lower living reward will make the agent exit sooner.

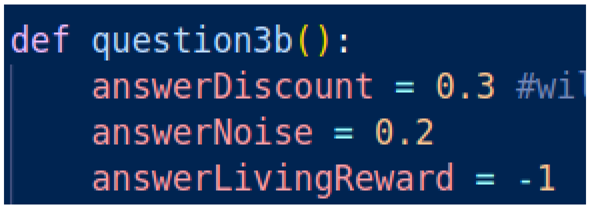
Explanation of the solutions

1. Prefer the close exit (+1), risking the cliff (-10)



```
def question3a():
    answerDiscount = 0.9
    answerNoise = 0.01
    answerLivingReward = -5
```

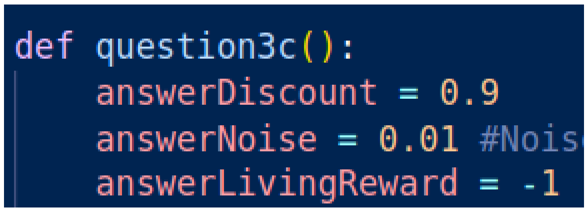
    - In this case, we increase the discount factor close to 1. That will encourage the agent to take the shorter path.
    - Noise is very low as we don't want any unintended action to be performed.
    - Living reward is negative that signifies a lot of penalty for living longer.
    - The combination of discount factor and living reward makes the agent take the shorter path and negative living reward makes the agent exit faster. So it exits from a close exit.
2. Prefer the close exit (+1), avoiding the cliff (-1)



```
def question3b():
    answerDiscount = 0.3 #wi
    answerNoise = 0.2
    answerLivingReward = -1
```

- In this case, we decrease the discount factor. That will encourage the agent to prioritize the future rewards and live longer to achieve the highest reward possible. So the agent will take the longer path.
- Noise is 0.2 signifying 20% of the time the agent performs unintended action. Makes the agent head upwards.
- Decreased discount factor will make the agent take the above path and agent will try to get to the closest exit as there is negative living reward.

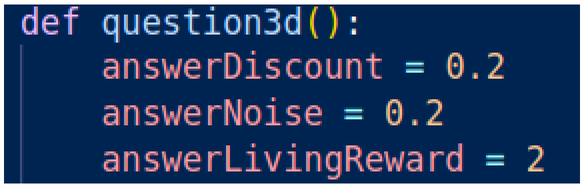
3. Prefer the distant exit (+10), risking the cliff (-10)



```
def question3c():
    answerDiscount = 0.9
    answerNoise = 0.01 #Nois
    answerLivingReward = -1
```

- In this case, we increase the discount factor close to 1. That will encourage the agent to take the shorter path.
- Noise is very low as we don't want any unintended action to be performed.
- Living reward is negative that signifies penalty of -1, that emphasizes the agent not to exit sooner. It waits for the future rewards.

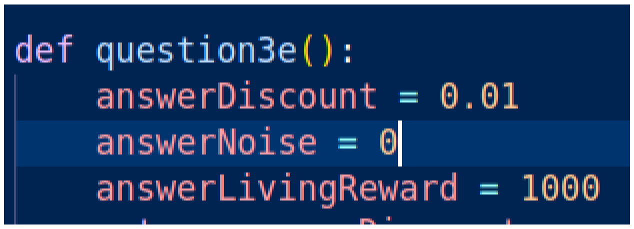
4. Prefer the distant exit (+10), avoiding the cliff (-10)



```
def question3d():
    answerDiscount = 0.2
    answerNoise = 0.2
    answerLivingReward = 2
```

- In this case, we decrease the discount factor. That will encourage the agent to prioritize the future rewards and live longer to achieve the highest reward possible. So the agent will take the longer path.
- Noise has the default value. So 20% chance of unintended actions.
- Living reward is increased so that the agent emphasizes on living longer for greater rewards.

5. Avoid both exits and the cliff (so an episode should never terminate)



```
def question3e():
    answerDiscount = 0.01
    answerNoise = 0
    answerLivingReward = 1000
```

- In this case, we decrease the discount factor. That will encourage the agent to prioritize the future rewards and live longer to achieve the highest reward possible. So the agent will take the longer path.
- Noise is 0. No unintended actions.
- Living reward is the most important factor in this case. We make the living reward positive that will encourage the agent not to take the exit ever.
- Also we are avoiding the cliff through the short discount factor.

Challenges and Solutions

No significant challenges were faced for this problem.

Behavior for Different Hyperparameters

- Discount Factor(γ):
  - Low Discount Factor (Close to 0):
    - With a low discount factor, the agent prioritizes immediate rewards over future rewards.
    - Might not be optimal in the long run.
  - High Discount Factor (Close to 1):
    - With a high discount factor, the agent values future rewards almost as much as immediate rewards.
    - Tends to choose actions that lead to higher cumulative rewards over time.
- Noise:
  - Noise refers to how often an agent ends up in an unintended successor state when they perform an action.
  - If the noise parameter is close to 1 it is more likely that the agent would take an unintended action.
  - If the noise parameter is close to 0 it is more likely that the agent would take intended actions.
- Living rewards:

    - Living rewards are some positive or negative values that the agent gets for living into the grid.
    - More living reward encourages the agent to live longer so that they can maximize their score as long as possible.
    - Negative or less living reward encourages the agent to die sooner or achieve the goal sooner respectively.

Interesting Findings

No interesting findings for this problem.

Task-4(Asynchronous Value Iteration)

Introduction

In this problem we need to implement an Asynchronous Value Iteration algorithm that updates the values of the states one by one rather than batch-style updates.

Analysis of the problem

- Unlike the standard Value Iteration algorithm, which updates the value estimates for all states in each iteration, the asynchronous version updates the value of only one state per iteration.
- In terminal states no updates will occur.
- During the iterations we need to traverse through the states and keep updating. If the states are done updating, return to the first state.
- This can be achieved through mod of iterations and states.

Explanation of the solutions

- The implementation is quite similar to the value iteration algorithm.
- The difference is that in this case we update only one state's value instead of all the states.
- We take the number of iterations and number of states.
- Then we get their mod value.
- When the number of states are updated once the mod value will return back to 0. That starts the whole update process again from the start.
- This process makes the value iteration process more efficient.
- For large amounts of state space, value iteration would be inefficient. Asynchronous Value Iteration would perform in those cases.
- Value Iteration updates all states simultaneously in each iteration. But Asynchronous Value Iteration updates states individually that leads to faster convergence.
- It also avoids updates that have already converged.

Challenges and Solutions

No significant challenges were faced for this problem.

Behavior for Different Hyperparameters

- Similar to Question 1.

Interesting Findings

No interesting findings for this problem.

Task-5(Reflex Agent)

Introduction

In this problem, we have to implement the Prioritized Sweeping Value Iteration Agent.

Analysis of the problem

- Prioritized Sweeping Value Iteration algorithm focuses on updating the states where the value function changes most frequently rather than updating all states uniformly.
- It uses the difference of the predecessor's state value and the current state value to calculate the magnitude of change.
- A threshold value theta is predefined to conditionalize when an update should occur.
- Using set() the algorithm avoids duplication of the predecessor states.
- A priority queue helps to keep track of the state that has the highest magnitude of change from the predecessor.

Explanation of the solutions

- The algorithm uses a dictionary of sets called predecessors. It keeps track of the predecessors of states. Set is used to avoid any kind of duplications of the predecessors for a particular state.
- A priority queue consisting of a min-heap is defined. It will contain the state that has changed with the highest magnitude at the top.
- Theta is a threshold that measures the magnitude of change. If the change is greater than theta then we update the priorityQueue with the particular state and the magnitude.
- As the priority queue is a mean heap, it will keep the highest changed state at the top. To ensure this we take the negative value of that change.
- The algorithm will encourage the agent to work with the top element of the heap.
- By prioritizing states based on the magnitude of the change in their values, we focus on updating states where the value function changes the most significantly.

- This allows us to converge to an optimal policy more efficiently, especially in large MDPs.

Challenges and Solutions

- It was difficult to understand the motive of this algorithm. But through some analysis and brainstorming we can understand the benefits.

Behavior for Different Hyperparameters

- Same as Question 1.
- Theta
    - Increasing theta:
        - With a larger theta, updates are only propagated to predecessor states when the change in value exceeds a higher threshold.
        - This can lead to fewer updates being propagated, resulting in slower convergence.
    - Decreasing theta:
        - With a smaller theta, updates are propagated more freely, even for smaller changes in value.
        - This can lead to faster convergence, as the algorithm responds more quickly to changes in the value function.

Interesting Findings

No interesting findings for this problem.