RL Path Planning

Vansh Thakur

1 Introduction

Used the canvas widget functionality of tkinter library to create a maze like structure. From the *env* file we import the final state of our agent after each time step, as our Temporal Difference Algorithms update online(after every time step) The agent is called via the *run* scripts Temporal Difference methods are focused on prediction learning, our goal is to estimate a value function that estimates the returns starting from a given state. Like Monte-Carlo methods, TD methods use experience, a key difference between them being them that in TD, V is updated on the next time step.

2 Q-Learning

Q-learning is a form of Off Policy TD Control Algorithm that estimates the optimal policy. We use the Bellman Optimality Equation for Action Values to directly compute q^* without policy evaluation and polices improvement steps. There is only value it - ration in q- learning.

Q learning follows greedy target policy selection $[\max(Q(s,a))]$, but has stochastic behaviour policy. It should he noted that there is no implementation of importance sampling here, even though it is mentioned that Q-learning is off policy. This is because we learn about the best action we could **possibly** take, rather than actually taking an action.

Update Algorithm

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + y_{\max}\left(Q\left(S',a'\right)\right) - Q(S,A)\right]$$

where :-

1. Q(S,A) is the array estimates of the action value function.
2. α is the step size parameter
3. R is the reward
4. y is the reward decay parameter

5. $Q(S', A')$ is the action value function of the next state-action pair which is transitioned to.

The updates are computed and stored Via the Q- Table.

3 SARSA

The SARSA algorithm is a Generalised Policy Iteration(GPI) algorithm where an action value function if learnt. The GPI algorithm involves steps of policy evaluation and policy improvement, here we focus on state-action transitions instead of state-state transitions. This an on-policy control algorithm. While slower than Q Learning, SARSA can produce more reliable results.

SARSA is a mnemonic for $(S_t)(A_t)(R_{t+1})(S_{t+1})$

Update Algorithm

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + y \left(Q \left(S', A'\right)\right) - Q(S, A)\right]$$

where :-

1. Q(S,A) is the array estimates of the action value function.
2. α is the step size parameter
3. R is the reward
4. y is the reward decay parameter
5. $Q(S', A')$ is the action value function of the next state-action pair which is transitioned to.

SARSA and Q Learning can be further extrapolated upon, as in the form of Expected SARSA, which takes into account the probability of an action being taken, or Double Learning Methods, which eliminate maxmisation bias in TD methods, which entails splitting update algorithms into two halves