6369 project-3 Report

Aishwarya Pothula - 1001743470

Overview

The project deals with the implementation of the DynaQ and priority sweep algorithms. These algorithms are implemented to train an agent to traverse a grid world environment from a given start position to the target position with the highest reward possible.

The grid environment and the related dynamics are implemented as GridWorldEnvironment in e.py.

The dynaQ and priority sweep algorithms are implemented in dynaQ.py and prioritySweep.py respectively.

Environment

The environment is a 15 x 25 grid. The environment is deterministic. There are 4 obstacles and 1 goal location. The start position is [[0,0],"n"] and the goal location is [14,24].

Action Space

The agent can perform "Forward", "Backward", "Right Turn" and "Left Turn". The grid location of the agent usually changes when the first two actions are performed. Case one of exceptions are when the agent tries to go out of the grid or when it encounters an obstacle. Case two is that there are 20% and 10% probabilities respectively for 'moving' and 'turning' actions that they are not performed by the agent; the agent retains its position and orientation within the grid. When the 'turn' actions are performed, the agent just changes its orientation and not the grid location

State Space

The environment has 1500 distinct states ranging from 0 to 1499. We derive this number by assigning observations to states. The size of the observation space is 15 x 25 x 4. While 15 and 25 represent the size of the grid, the 4 represents the four orientations (n, s, e, w) of the agent.

The formula used to assign observations to states is $x + 1 * y + 1 * b * z$ where x,y are the agent location grid coordinates and b z is the orientation. L and b represent the dimensions of the grid. I have encoded the orientations as [n:0,s:1, e:3, w:4]. For example, if the observation is [[2, 3], "s"], the associated state would be $2 + 3 * 15 + 15 * 25 * 4 = 1547$

Rewards

The environment gives a reward of 100 when the agent reaches the goal position, -100 when the agent encounters an obstacle or tries to move out the grid. For all other actions, the agent is given a reward of 0.

DynaQ

The DynaQ algorithm combines both online and offline learning. The learner samples from the visited states and the related taken actions

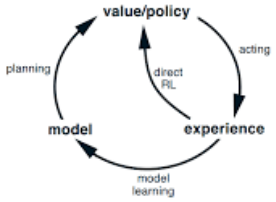


Figure 8.1: Relationships among learning, planning, and acting.

Priority Sweeps

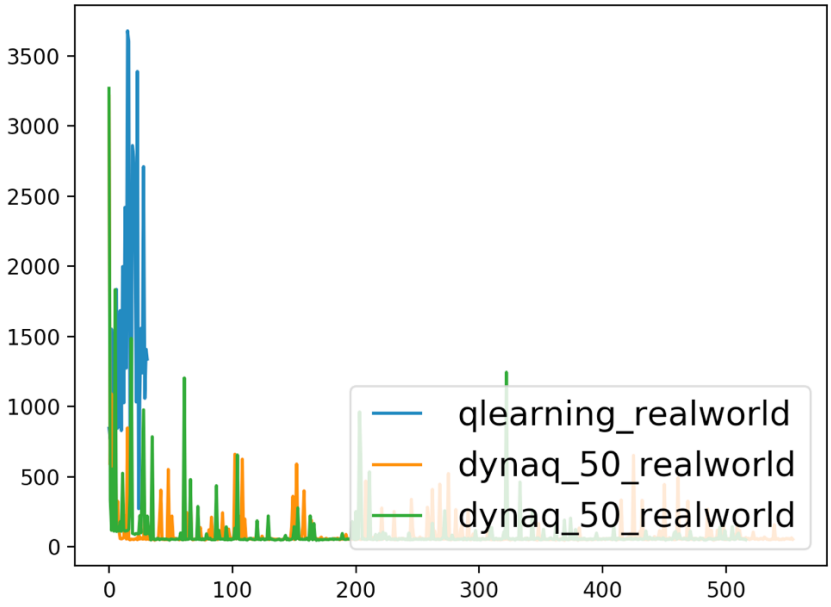
Priority sweeps are implemented to make the q_updates better. They allow for an update of the q_table of if the bellman error is above a certain value theta. Many q_table updates are unnecessary as as those states are not visited often. The priority updates ensures that states with more bellman error are sampled more often.

Trajectory Sampling

Trajectory sampling is done to distribute the updates. Here we distribute the updates according to the on-policy distribution by using epsilon greedy.

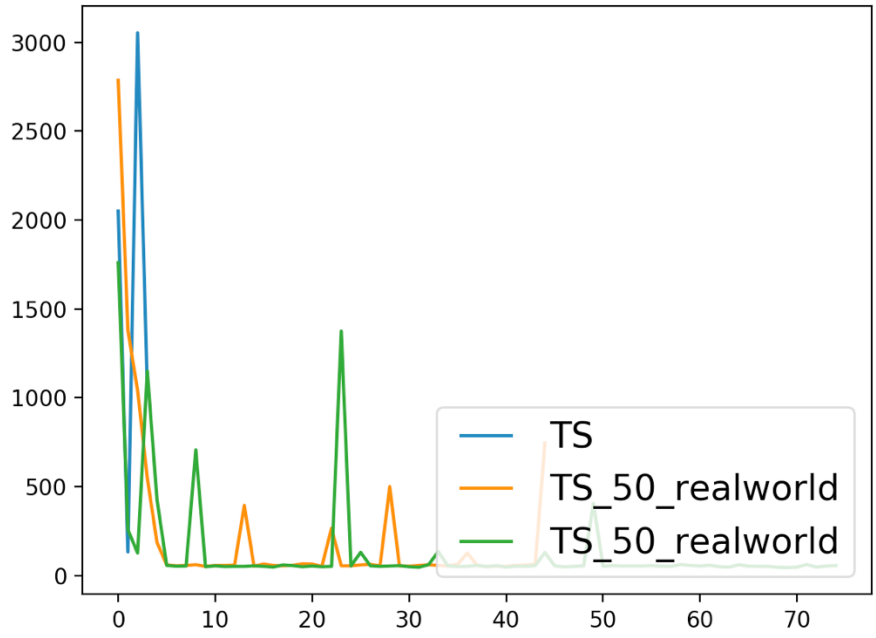
1. The implementation of this has been done in the e.py file.
    a. State space is 0 to 1499 consisting of 1500 state spaces. MDP P(s|a) = P(s_prime|r). Observation is [[x,y],"orientation"]
        i. S = initializing to start state of 0
        ii. A = choosen according to epsilon greedy algorithm
        iii. S_prime = given by the env based on previous state and action
        iv. Reward = generated by the env based on s,a
        v. Then, s = s_prime

    b. Implemented in dynaQ.py
    c. Outout of dynaQ for n=0, n=50 and n=100 with 50000 max_steps



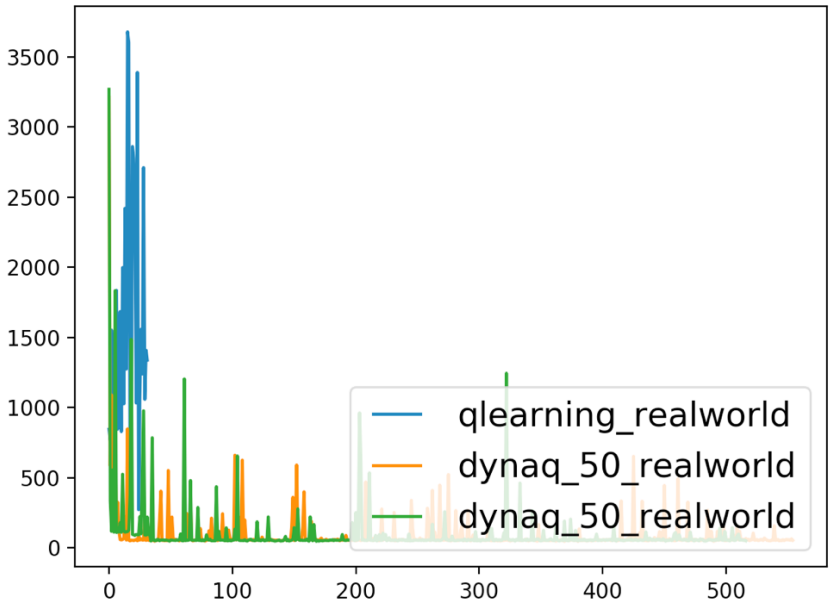
i.

2. We are using priority based system for for picking model_steps.
    a. Implemented in PrioritySweeps.py
    b. Comparision of DynaQ, Prioritysweeps and Trajectory Sampling



i. In both cases when n=0, the learning was faster as they reached the

minimum number of episodes very quickly



c. Implemented in trajectory_sampling.py.
   i. Result for trajectory sampling when n=0, n= 50 and n=100 with max_steps as 10,000