Part 2 - Project report

Årne, Runar and Lars

March 17, 2024

1 Introduction

In this project we will first consider solving gymnasiums 'Mountain-Car' environment using feature engineering. Then we will use DynaQ to solve a different problem using planning.

2 Feature Engineering

Before running the 'Mountain-Car' environment, we observe that the observation space is bounded by $[-1.2, 0.6]$ for the x-position of the car, and $[-0.07, 0.07]$ for the velocity of the car. Here, there are several discretization steps we could choose, but we landed on $\Delta \in \{0.05, 0.01, 0.005, 0.001\}$. We use Q-learning to learn the optimal policy, and plot the rewards per episode as a comparison measure between the different discretization steps (figure 1).

From the plot we can see that the choice of the discretization step Δ really matters in how fast and well the policy is learned. We can see that 0.5 seems to be too big to be able to accurately represent all the different states, while 0.01 seems to be too small to efficiently learn the policy.

In fact, our choice of 0.1 seems to be the most optimal in this instance, as this consistently has the highest mean reward of them all!

When reflecting on other forms of feature engineering we could consider applying to this problem, we could try to do coarse coding. This is a good way of minimizing the number of possible states we need to store while still keeping a lot of the information and precision about the current state. Here, we could also tune the coarseness and number of circles to our liking.

For this task, the most relevant coarse coding would probably be tile coding, where we tile the space into overlapping tiles which represent our state. This is similar to our discretization step, but as we now could represent our state with several tiles (as they overlap) it will not be exactly the same.

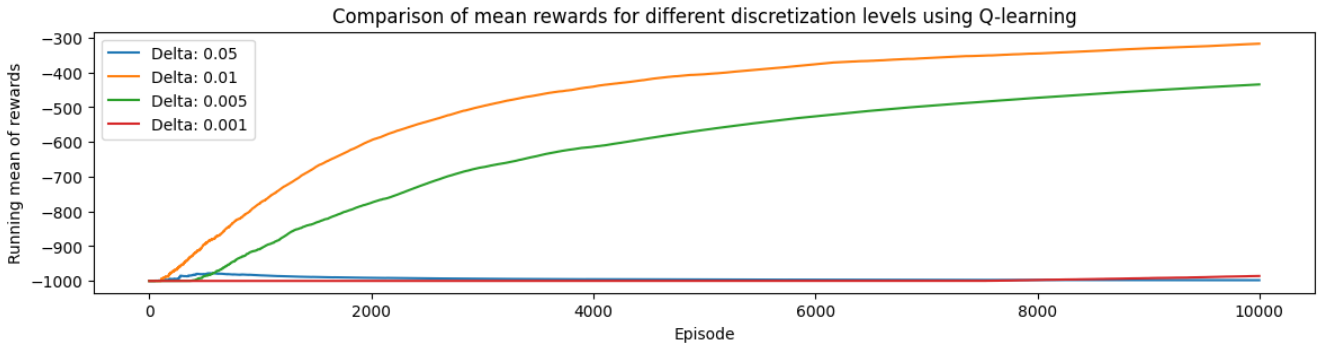

Figure 1: Running mean of rewards per episode performing Q-learning with different discretization steps.

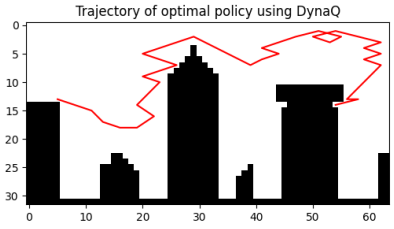

Figure 2: Optimal trajectory using DynaQ

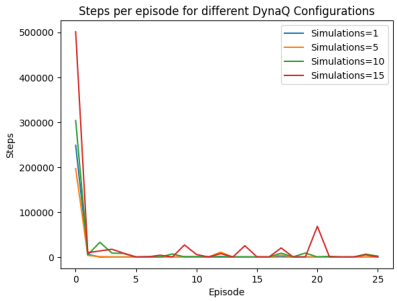

Figure 3: Comparison of DynaQ algorithm with differing simulation count

3 Planning

For this problem we are given an environment to learn an optimal policy in. We develop the model $\hat{\mathcal{M}}$ using a tabular solution, where the transition function $f_T(s, a; w_T)$ and the reward function $f_R(s, a; w_R)$ are represented. The transition function seems to be deterministic from the problem description, therefore the approximated transition function $f_T(s, a; w_T)$ could be simply represented as a single value per state-action pair (s, a). This is all done within the DynaQ algorithm. We also use the already gathered data to "initialize" our model approximation $\hat{\mathcal{M}}$.

After training using DynaQ with the hyperparameters $\gamma = 0.9$, $\alpha = 0.1$, $\epsilon = 0.5$, $\epsilon_{\text{decay}} = 0.3$, episodes $= 50$ and simulations $= 10$ we have obtained the optimal trajectory plotted in figure 2.

Let's now run DynaQ using a different amount of simulations per timestep.

To compare the results by varying the number of simulations run by the agent we will plot and analyze the number of steps taken per episode. This will give us information about how good and how fast each configuration learns the policy (figure 3).

As we can see, setting simulation count to 15 drastically increases the number of steps needed to reach the target in the first episode, as well as creating bigger extremes in some random fluctuations.

Here, it seems setting simulation count to 5 would be the optimal one, as it has fewer and smaller fluctuations, as well as having the minimum amount of steps before reaching the target in the first episode.