Homework 6: Reinforcement Learning

Thomas Wei
ThomasW219@gmail.com

***Abstract*—Reinforcement learning is the process of learning behaviors that maximize the reward gained over the lifetime of an agent. Different methods of reinforcement learning have been developed that take different approaches to the issue of finding optimal behavior as well as to deal with state spaces of varying complexity. Q-Learning is a widely employed form of model-free reinforcement learning. It works by approximating the expected future reward from performing an action in each state (assuming subsequent actions are optimal). When both the state and action space are finite, Q-Learning can be done in a tabular fashion, where each Q-value approximation is kept explicitly and updated when the corresponding action is taken while in the state.**

The task that we wish to have our agent complete is to navigate from one end of a 6×25 grid to the other while avoiding obstacles, staying on the sidewalk, and picking up litter. For the agent to consider the position of litter and obstacles as part of the state, the size of the state space (considering all possible arrangements of litter and obstacles) would become so large that tabular Q-Learning would no longer be able to be done in a realistic amount of time. We wish to approximate an agent trained on the complete state space by learning individual policies for different aspects of the state and then combining the policies to form the full agent. The sub-problems are referred to as modules. We will reduce the full state space to 4 modules: one that collects litter, one that avoids obstacles, one that remains on the sidewalk, and one that walks forward.

I. Introduction

Consider the problem of an agent interacting with an environment. The actions of the agent can affect the state of the environment, but the dynamics of how the state changes in response to the agents actions are unknown to the agent *a priori*. The agent is also given some reward signal from the environment at each time step depending on its state, action, and transition to the next state. Reinforcement learning is the process of learning behaviors that maximize the reward gained over the lifetime of the agent.

Different methods of reinforcement learning have been developed that take different approaches to the issue of finding optimal behavior as well as to deal with state spaces of varying complexity. Model-based reinforcement learning methods use experience of state-action-state transitions to learn the dynamics of the environment and then base the agents behavior off of the learned dynamics. On the other hand, model-free reinforcement learning techniques do not explicitly seek to model the dynamics of the environment. Rather, they learn their behavior from the data in other ways.

Q-Learning is a widely employed form of model-free reinforcement learning. It works by approximating the expected future reward from performing an action in each state (assuming subsequent actions are optimal). When both the state and action space are finite, Q-Learning can be done in a tabular fashion, where each Q-value approximation is kept explicitly and updated when the corresponding action is taken while in the state. The only condition that must be satisfied is that each state action pair must be visited infinitely often for an optimal policy to be guaranteed. For our purposes, we may not necessarily need complete optimality. However, we do need to visit each state action pair enough for the rewards, which are somewhat sparsely distributed, to propagate throughout the state space.

The task that we wish to have our agent complete is to navigate from one end of a 6×25 grid to the other while avoiding obstacles, staying on the sidewalk, and picking up litter. If the positions of the obstacles and litter were static (given to the agent during training) and the agent only had to learn a policy for the given litter and obstacle positions, the problem could be solved with a direct application of tabular Q Learning. However, we want our agent to be able to handle a sidewalk with randomly positioned obstacles and litter that are not given to the agent during the training phase. For the agent to consider the position of litter and obstacles as part of the state, the size of the state space (considering all possible arrangements of litter and obstacles) would become so large that tabular Q Learning would no longer be able to be done in a realistic amount of time. We wish to approximate an agent trained on the complete state space by learning individual policies for different aspects of the state and then combining the policies to form the full agent. The sub-problems are referred to as modules. We will split the full state space behavior into 4 modules: one that collects litter, one that avoids obstacles, one that remains on the sidewalk, and one that walks forward. One advantage of this approach is that once the policies for each module is learned, the priority of the full agent with respect to each module can be adjusted in the combination phase, rather than requiring any additional training or modification to reward functions.

II. Method

A. Q Learning

Since we're dealing with finite state and action spaces, we will employ tabular Q Learning. Each Q value is initialized to some initial value and is updated with every experience of the corresponding state and action using the following update:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(R(s,a,s') + \gamma \max_{a'} Q(s',a')) \tag{1}$$

Where s, a are the current state and action respectively, s' is the state returned by the environment after taking action a in

state s. Here α, γ are hyperparameters which dictate the rate of learning and the emphasis on future rewards respectively.

B. Full Problem Description

The full state space for our problem includes the position of the agent as well as whether or not each position has a piece of litter and/or an obstacle. In addition, the first and last rows are not sidewalk while the middle four rows are. This does not need to be encoded in the state as it is a static part of the environment. The actions are an idle, the four cardinal directions, and the four diagonals. In addition, whenever the agent moves to a location with litter, that litter is removed. The size of the state space can be computed as follows.

$$(6 \times 25)4^{6\times 25} \approx 3.05 \times 10^{92} \tag{2}$$

Since each position can contain litter, an obstacle, both, or neither and for each arrangement of litter and obstacles, the agent can be in any position. This state space is clearly too large to explore with our available resources so we turn to dividing the problem into modules.

C. Litter Module

The litter agent we train on the space of 5×5 matrices with binary entries, each entry representing whether or not there is a piece of litter in that position. In this encoding, the agent is assumed to be in the middle grid position. All the same actions are available from the original state space, they shift the grid so that the agent stays in the middle. As with the original, litter is collected when the agent moves to a position with litter, the reward function we define rewards the agent with a reward of 1 when this occurs. All other rewards are 0. During training we initialize randomly with litter and pad the movement of the agent with litter-free area. Whenever the agent clears all the litter we reinitialize with random litter again.

Although this reduced state space is still quite large, we opt for a 5×5 larger state space rather than a 3×3 state space that would only allow us to respond to adjacent litter. Our encoding allows us to respond to litter 2 actions away. Though we may not sufficiently explore the entire state space to get optimal actions throughout, we train with litter initialized with probability similar to what we expect in the full problem. This allows us to thoroughly explore the region of the state space that we are likely to encounter. We made the design choice that having more informed moves most of the time was more important than having a full optimal policy but that benefits the full agent only in a smaller set of situations.

D. Obstacles Module

The obstacles module is similar to the litter module except we do opt for the 3 state space rather than the 5. We initialize and train similarly with the modifications that obstacles are not removed when the agent moves to their position and the reward function gives -1 when an agent moves to a state where it lies on an obstacle. Again, all other transitions are given 0 by the reward function. Since this state space is far smaller than the litter state space, we have no problems exploring it entirely.

E. Sidewalk Module

The sidewalk module simply takes the position component and dimensions of the full state. Again all actions are available, they modify the position the same way that taking an action would in the original state space. In order to train this module we define a reward function that penalizes the agent for being in any of the positions in the first or last rows with a reward of -1 and gives 0 for every other transition. We initialize in one of the positions in the first column (on the sidewalk) and consider any position in the last column as terminal.

F. Forward Module

The forward module is exactly the same as the sidewalk module except that the reward function gives 1 for entering any of the terminal states and 0 otherwise.

G. Combining Modules

When considering the full problem, we first extract each of the states that the modules use from the full state. This allows us to access the Q values of interest from each of the modules. Then, in order to keep the influence of each module the same regardless of magnitude of Q values, we shift and linearly scale the Q values for each module so that the highest Q value goes to 1 and the lowest goes to 0. We then take the weighted sum of these Q values for each action which gives us a combined score of each action. The policy can then be determined based off of the respective scores for each action.

III. Results

We trained each of the modules for 500 periods of 10,000 time steps each. For each of the periods, we computed the average difference of the Q value prior to the update with the update itself and graphed that against our training time. All training used the ϵ-greedy exploration function with an ϵ value of 0.25 as well as α, γ values of 0.05 and 0.9 respectively.

A. Litter Module

In order to demonstrate the actions learned by the litter module, we run our combined algorithm with the weight of the litter module at 0.75 and the weight of the forward module at 0.25. All other weights are 0. We do this to show a sample trajectory in the full state space where only the region within 2 actions of the agent has been given to the litter module. This shows the limitations of our approach. We add a small amount of forward module bias as well so that we eventually reach a terminal state. The sample trajectory can be seen in figure 2.

We observe that the agent collects a large amount of the litter in the full state space. There are occasions where litter goes unobserved, for instance the upper-right most piece of litter, or where a piece of litter is visible but declined in favor of other litter. The latter occurred with the litter in row 4 column 10. When the agent was at row 2 column 10, the row 4 column 10 litter was visible, however there were two pieces of litter in the bottom right of the local state space. The agent decided two pieces was better than one and moved to collect the lower right pieces. In doing so it moved outside of the

Fig. 1. Difference of the Q value prior to the update with the update itself against training periods for the litter module

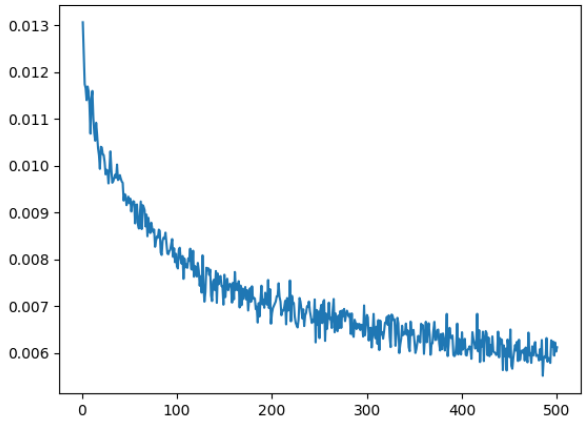
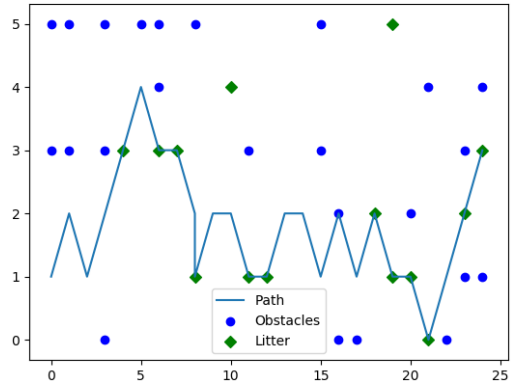


Fig. 2. A sample trajectory with litter weight at 0.75 and forward weight at 0.25



region where it could observe the row 4 column 10 piece of litter and moved on.

B. Obstacles Module

We similarly evaluate a sample trajectory of the full state space where the weight for the obstacle module is 0.75 and the weight for the forward module is 0.25. The obstacles module only perceives obstacles one move away from the agent. Again, we add a small amount of forward module bias as well so that we eventually reach a terminal state. The sample trajectory can be seen in figure 4.

In the sample trajectory, the agent avoids all obstacles on its way to the terminal state.

C. Sidewalk Module

We similarly evaluate the sidewalk module in conjunction with the forward module so we reach a terminal state. The weights used for this sample trajectory in the full state were

Fig. 3. Difference of the Q value prior to the update with the update itself against training periods for the obstacles module

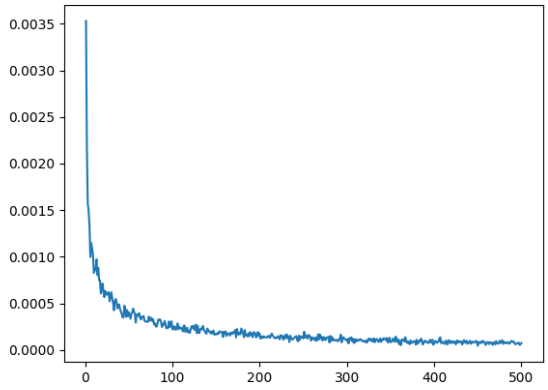
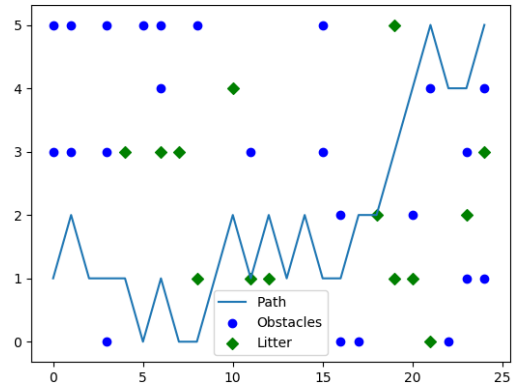


Fig. 4. A sample trajectory with obstacle weight at 0.75 and forward weight at 0.25



0.75 for the sidewalk module and 0.25 for the forward module. This can be seen in figure 6.

As we intended, the agent stays on the sidewalk rows (rows 1-4) on its path to the terminal state.

D. Forward Module

We test the forward module alone, using the same full state space configuration. The resulting trajectory can be seen in figure 8

E. Combining Modules

Here we finally tackle the initial problem, combining each behavior learned in the modules. Since the reduction of the state space given to the modules obfuscates some of the information, we ran into the issue of deterministic cycles occurring when simply choosing the highest weighted score action. To remedy this we simply added exploration, using ϵ-greedy with an ϵ value of 0.1. We first tried a weighting of 0.25 for all of the modules. That gave us the sample trajectory in figure 9.

Fig. 5. Difference of the Q value prior to the update with the update itself against training periods for the sidewalk module

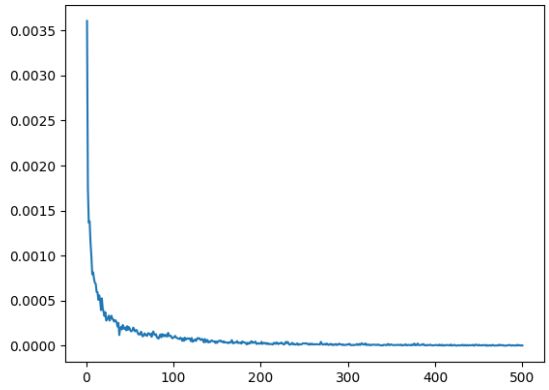


Fig. 6. A sample trajectory with sidewalk weight at 0.75 and forward weight at 0.25

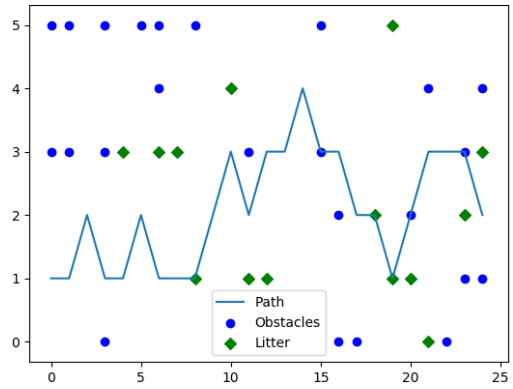


Fig. 7. Difference of the Q value prior to the update with the update itself against training periods for the forward module

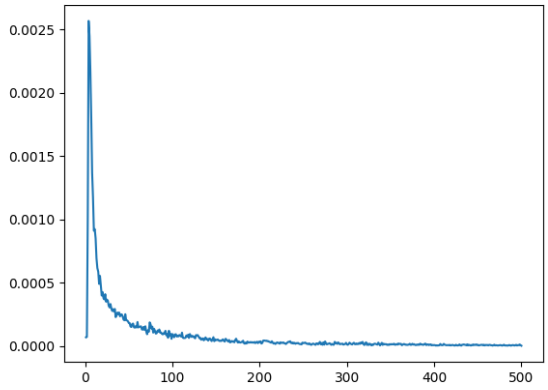
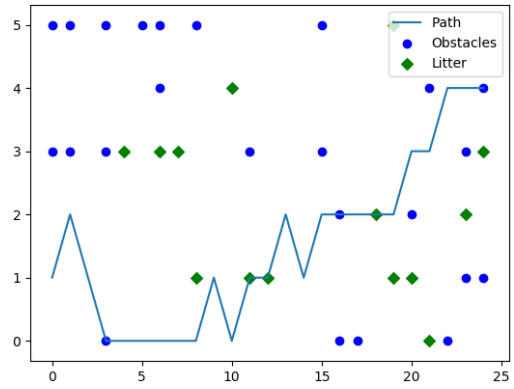


Fig. 8. A sample trajectory with forward weight at 1



As desired, the agent stayed on the sidewalk, collected a decent amount of the litter, avoided all obstacles, and reached a terminal state at the end of the sidewalk.

We then tried modifying the weights by increasing the weight of the litter module to 0.4 and decreasing the rest to 0.2. This yielded the sample trajectory shown in figure 10.

As we can see, the trajectory is similar to the equal except near the end, the agent leaves the sidewalk to collect an additional piece of litter. This is unsurprising as we weighted the value of the litter module as twice the value of the sidewalk module. Moreover this shows that with this approach, revising the agent's priorities is as simple as reweighting the modules. Had we performed Q Learning on the full state space, to accomplish the same revising of the priorities, we would've had to overhaul the reward function appropriately and then redo the training process with the new reward function.

This is an example of ideal behavior. We see some of the issues our model faces in figure 11 such as getting stuck idling due to conflicting information from modules and deterministic cycles. In addition, the conflicting priorities from modules causes easily detectable litter to be bypassed as well. As discussed before, we remedy these issues with randomness but, as is evident from the sample trajectories, there is room for improvement.

IV. Summary

We attempted to approximate the optimal policy in a large state space by distilling the full state into smaller state spaces that we could learn with tabular Q Learning and then construct a policy in the full state space by combining the information from each of the modules. We found that, while certainly not optimal, the agent created by combining the modules for the smaller problems exhibited many of the traits that an agent trained to solve the full problem would exhibit. Some notable deficiencies of the constructed agent is its limited knowledge of the locations of the litter and the deterministic cycles that occur as a consequence of the partially observed state. One advantage is that the values of each of the modules can be modified without requiring any retraining.

Fig. 9. A sample trajectory with all weights equal at 0.25

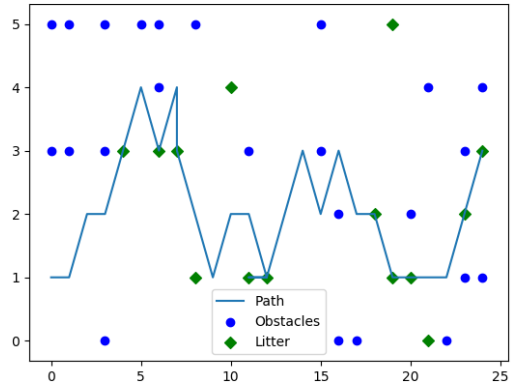


Fig. 10. A sample trajectory with litter weighted 0.4 and the rest weighted 0.2 each

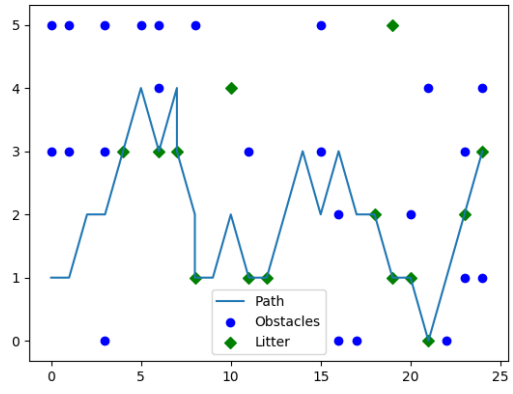


Fig. 11. More sample trajectories, different full states, weighted with litter at 0.4, all else at 0.2