Reinforcement Learning

Upayan Mathkari
upayan@cs.utexas.edu

The University of Texas at Austin
Dept. of Mechanical Engineering

1. Introduction

With supervised learning, in theory, a machine learning model can only be as insightful or intelligent as the human that provided it the training examples. In unsupervised learning, models are trained to learn independently without training examples provided by humans. One such technique is reinforcement learning, designed to somewhat mimic the way humans learn. A software agent faces a game-like situation where it tries to learn the optimal sequence of decisions that will allow it to maximize its cumulative reward function. This function is based on rewards and penalties that programmer assigns to different actions that the machine may perform.

In this project, we simulate a sidewalk with randomly appearing litter and obstacles. Q-learning methods are used to train the computer agent to traverse the sidewalk in as few steps as possible while making sure to avoid the grass and obstacles and picking up any litter that appears in its path.

2. Methods

Module Generation

Different modules, analogous to video game maps, are created to train the agent to perform each of the following tasks:

1. Reaching the end of the sidewalk
2. Staying on the sidewalk
3. Avoiding obstacles
4. Collecting litter along the way

Each of these modules is generated by initializing a 6x24 matrix, *M*, which represents an overhead view of the sidewalk (see figure 1).

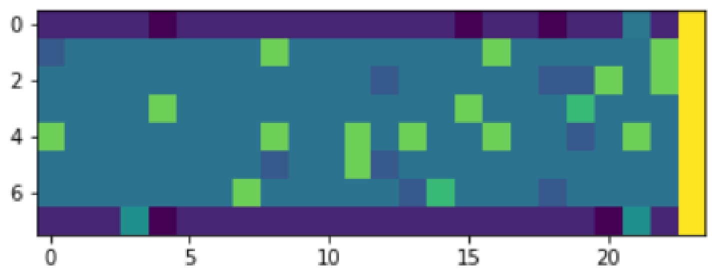

Figure 1: Visual combination of modules 1-4. (a) Yellow bar represents end of sidewalk, (b) Purple represents grass, (c) Blue squares represent obstacles (d) Green squares represent litter

Each element, m_{ij}, corresponds to a coordinate, *(i, j)*, on the map with a value representative of the reward for it contains (e.g. grass or litter). All modules utilize the 24th column to represent end of the sidewalk. In the module 2 (grass), the rows 1 and 8 represent the grass while the middle rows correspond to sidewalk. In modules 3 and 4, obstacles and litter, respectively, are randomly generated throughout the map with a 10% occurrence probability.

Reward Matrix Generation
At any given time, the state of the agent is defined by its coordinate location on the map allowing for a state space of dimension=6x24. The agent is allowed to move in 5 directions described from the perspective of an overhead observer: (1) right, (2) up, (3) down, (4) diagonal, and (5) diagonally downward which allows for an action space of dimension = 6x24x5. Initializing the state space and action space with the following reward values, we generate the modules described above and a reward matrix which is discussed in this section, respectively.

Item	**Reward**
Litter	+ 5
Obstacle	-10
Grass	-20
Empty Sidewalk	-1
Sidewalk End	15, 5, 5 (depending on module used)

Most of the reward values given are quite self-explanatory. Litter and the sidewalk end are assigned positive values to encourage picking of litter and making it to the end of the sidewalk, respectively. Meanwhile, the obstacles and grass are given negative values to encourage avoidance of these states. Lastly, the empty sidewalk is given a negative value to encourage taking the most efficient path possible by minimizing the total number of steps taken.

The relative magnitudes of the rewards are important as well. In general, the sidewalk end is given the highest reward to provide motivation to reach to the end destination rather than spending endless time picking litter. Other relative magnitudes were experimentally determined based on visually determining the success of the agent.

Q-matrix
Simply following the reward matrix, the agent will pick the its next step based on the goal of trying to maximize the instantaneous reward possible from the state transition without any foresight of the future rewards that the state may give it access to. The Q-matrix (dimension = 6x24x5), also known as the policy matrix, allows the agent to store its experience about the rewards that may be accessible in the future after conducting a given state transition. Once trained, the values of the Q-matrix map a state-action pair to the expected cumulative reward for taking the action in a given state. The trained Q-matrix can be used to provide the agent with the strategy for the optimal action choice at any given state as follows:

$$a(s) = \max\big(Q_i(s,a)\big) \qquad (1)$$

By selecting the action with the maximal Q-value, the agent selects the action that will yield the maximal expected cumulative reward based on its experience.

Q-learning
How an agent encapsulates it experience into the Q-matrix is the topic of Q-learning. The Q-matrix is initialized to 0 for all its state action pairs in the beginning. However, after every action the agent takes, Q-values are updated according to the Bellman equation shown below:

$$Q_i(s,a) = (1-\alpha_i)Q_i(s,a) + \alpha_i(r_i + \gamma_i(\max_a Q_i(s',a')) \qquad (2)$$

The parameters α, γ, and *r* are, respectively, the learning rate, discount factor, and reward. The learning rate determines the extent to which the new information changes the policy. As α tends to 1, almost all the old information is discarded in favor of the new information. This new information is comprised of the second term on the right equation, most importantly in the reward, *r*. Every time an agent takes a step (state-action pair), it updates the valuation for that step based on the reward it got for taking it. However, the agent also updates the valuation of that step based on the expected future rewards which constitutes the second part of the term. The discount rate determines how much the value of the future rewards is reduced based on the time delay.

To train the agent to determine the optimal policy based on this method we employ the epsilon-greedy strategy. Before arriving at the optimal policy, the agent will take steps with highest Q-value with probability, *1- ε*, and all other steps with probability, *ε/n*, where *ε* is a parameter between 1 and 0 and *n* is the total number of possible state-action pairs minus 1. Therefore, a high ε value will favor exploration of the environment while a low ε while favor exploitation of what is known. In this project, we vary ε from 1 to 0 linearly through to the learning iterations so that the agent changes its bias towards exploitation as it learns more.

3. Results

Forward Module

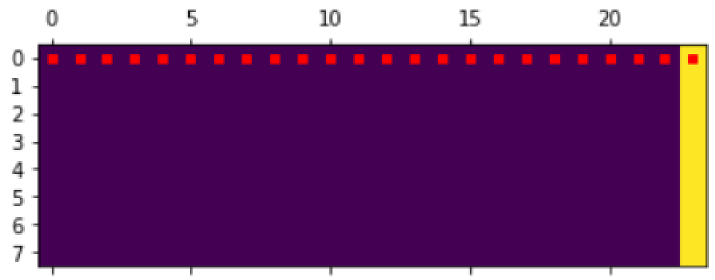

Figure 2: (a) Yellow bar represents end of sidewalk, (b) Purple represents empty sidewalk, (d) Red is the agent path

Grass Module

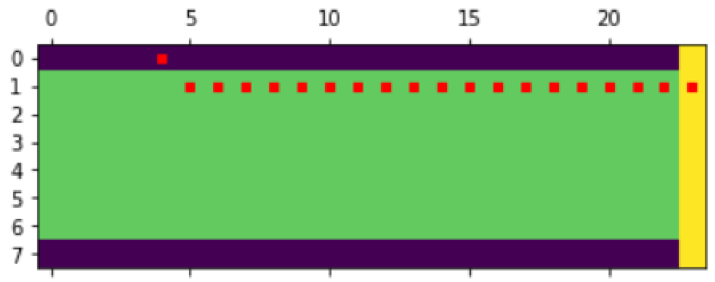

Figure 3: (a) Yellow bar represents end of sidewalk, (b) Purple represents grass, (c) Green represents empty sidewalk, (d) Red is the agent path

Litter Module

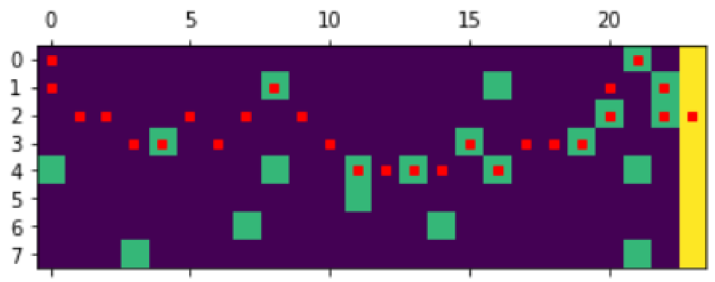

Figure 4: (a) Yellow bar represents end of sidewalk, (b) Purple represents empty sidewalk, (c) Green squares represent litter. (d) Red is the agent path

Obstacle Module

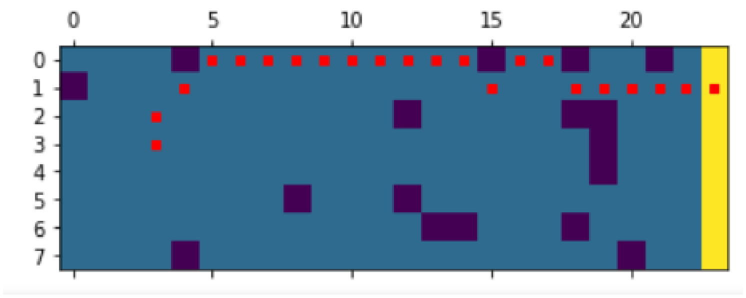

Figure 5: (a) Yellow bar represents end of sidewalk, (b) Purple squares represents obstacles, (c) Blue squares represent obstacles. (d) Red is the agent

Combined Module

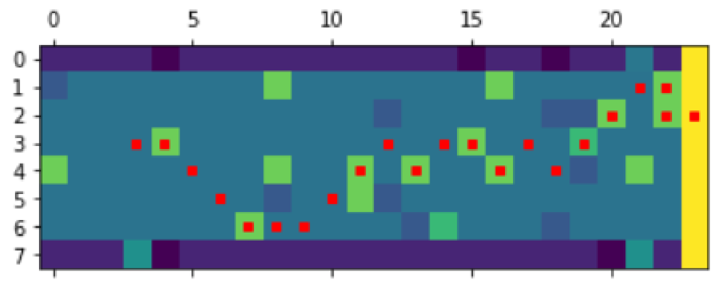

Figure 6: (a) Yellow bar represents end of sidewalk, (b) Purple represents grass, (c) Green blocks represent litter, (d) Blue blocks represent obstacles (e) Turquoise represents empty sidewalk, (f) Red is the agent path

4. **Summary**

Reinforcement Learning (RL) is powerful technique allowing computers to learn in a way that mimics human learning using rewards to promote certain behaviors and penalties to discourage others. In this project, Q-learning algorithms using the epsilon greedy method were successfully implemented to train a computer agent to navigate a sidewalk with the goal of avoid obstacles and grass, reaching the end, and picking up any litter on the way.