CS 391L HW4: Reinforcement Learning - Self-Driving Car Simulation

Mit Shah (UT EID: mks3226)
University of Texas at Austin
Austin, TX 78712
mkshah@utexas.edu

Abstract

This assignment is aimed at implementing basic Reinforcement Learning algorithms for simulating a self driving car. Reinforcement learning (RL) is a sub area of machine learning, where agents are motivated or discouraged to do something in the form of reward. Q-Learning is one such algorithm to RL. We will use this algorithm to teach an agent-Car in our case - how to drive on the road. We will reinforce agent to follow basic things, such as moving forward, waiting at traffic signal, stopping for pedestrians, avoiding static obstacles and dynamic obstacles (such as other cars), etc. We will use modular approach for learning such multiple tasks. We will visualize the results in a small simulator built in Pygame and see our Self-Driving Car in action!

1. Introduction

First let's get familiar with what is Reinforcement Learning and then its Q-Learning algorithm. Also, we will talk about what we are trying to achieve here in this section.

1.1. Reinforcement Learning

It is an area of machine learning inspired by behaviorist psychology, concerned with how software agents take actions in an environment so as to maximize some notion of cumulative reward. The main difference between the classical techniques and reinforcement learning algorithms is that the latter do not need knowledge about the Markov Decision Processes (MDPs). It differs from standard supervised learning in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected. It also tries finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

The basic reinforcement learning model consists of:
1) A set of environment and agent states S;
2) A set of actions A of the agent;
3) Policies of transitioning from states to actions;
4) Rules that determine the scalar immediate reward of a transition;
5) Rules that describe what the agent observes.

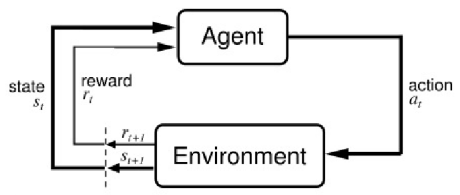

Figure 1. RL Framework

A reinforcement learning agent interacts with its environment at discrete time steps. At each time t, the agent receives an observation o_t, which typically includes the reward r_t. It then chooses an action a_t from the set of actions available, which is subsequently sent to the environment. The environment moves to a new state s_{t+1} and the reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}) is determined. This framework is shown in Figure 1 The goal of a reinforcement learning agent is to collect as much reward as possible. The agent can choose any action as a function of the history and it can even randomize its action selection.

1.2. Q-Learning

Q-learning is a model-free reinforcement learning technique. It can be used to find an optimal action-selection policy for any given MDP. It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. When such an action-value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state. It is able to compare the expected utility of the available actions without requiring a model of the environment. Also, it can handle problems with stochastic transitions and rewards, without requiring any adaptations.

For each given (state, action) pair, we maintain a Q-value, which is updated as in 2. All of them can be initialized with zeros. When fully learned, we select the action with highest value in each state.

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha_t}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \right)$$

Figure 2. Q-Value Explained

1.3. Goal

Here our goal is to reinforce some basic driving sense to an agent. We will mainly focus on teaching following things to agent:

1) Moving Forward
2) Stopping when Traffic Signal is Red
3) Waiting for Pedestrians to cross the road
4) Avoiding Static Obstacles (potholes, etc)
5) Avoiding Dynamic Obstacles (other cars, etc.)

Let's see how we will be implementing all these using Q-Learning in next section.

2. Method

Let's define our environment first. It will be 2 x 20 grid, in which agent can move. Agent will start on one end of the gird and the other end is the destination. In between, environment will have traffic signals, pedestrians crossing the roads, some potholes (static obstacles) and moving cars (dynamic obstacles). Our agent has to cross all these things successfully to reach the destination. Environment and Agent are shown in Figure 3.

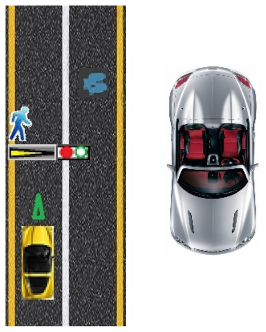

Figure 3. Environment And Agent

We will use the Modular approach to learn all these different tasks. That is, we will train our Car to do all these things individually and then we will develop a policy to combine all these learning. Now, we will see how to learn each task one by one.

For each of them, we need to define what our states are. Four actions will be common for all of them: 1) Forward 2) Left 3) Right 4) Stop. (Action will not be considered if it puts the agent outside the environment). Also, we need to define a reward for each (state, action) pair in all these scenarios.

2.1. Modular Training

2.1.1 Moving Forward

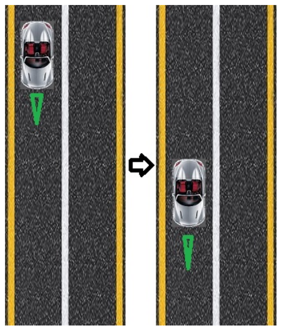

Figure 4. Car Moves Forward

Here, we have to give agent motivation to move forward. There are no other tasks. So, there is only one state. Agent is always in the same state and expected to move forward. We give +5 reward for the action Forward, and -1 for taking any other action (Left, Right or Stop). Also, at the end of the grid we will give it a reward of +10 for reaching the destination. This will enforce agent to move forward.

Table 1. Rewards - Moving Fwd

State/Action	Forward	Left	Right	Stop
-	+5	-1	-1	-1

From these rewards 1, agent learns to move forward in 4 to 5 Episodes on an average. Minimum steps to reach destination is 19 and max reward it can get is 100. It starts with around 37 Steps and accumulates 82 points in Episode 0, but quickly reaches the optimal level.

Table 2. Q-Values - Moving Fwd

State/Action	Forward	Left	Right	Stop
-	14	3.6	3.9	3.9

From the Q-values 2 we can see that agent will pick forward action always. Result is shown in Figure 4.

2.1.2 Stopping At Traffic Signal

Here, we have to make agent stop when traffic signal is Red. So, we will define two states. First state will include conditions when to stop: If traffic signal is ahead and it is red. Second state will include conditions when not to stop: Either traffic signal is not ahead Or it is ahead but green. We will define our rewards as follows, which heavily penalizes agent for violating traffic signal.

From these rewards 3, agent learns follow traffic signal again in 4 to 5 Episodes on an average. Max reward it can get is 128 and min steps are 21. It starts with around 70

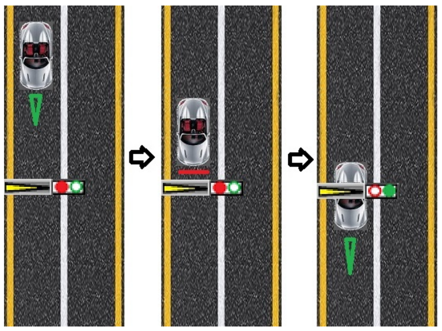

Figure 5. Car Follows Traffic Signal

Table 3. Rewards - Following Traffic Signal

State/Action	Forward	Left	Right	Stop
Stop	-300	-1	-1	100
Don't Stop	1	-1	-1	-1

Steps and accumulates -275 points in Episode 0, but quickly reaches the optimal level.

Table 4. Q-Values - Following Traffic Signal

State/Action	Forward	Left	Right	Stop
Stop	-236	0	0	104.9
Don't Stop	13	3.9	3.9	4

From Q-values 4, we can see that agent will move forward in second state, but will stop in first state. Result is shown in Figure 5.

2.1.3 Waiting For Pedestrians

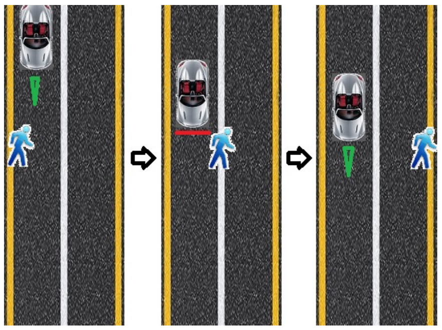

Figure 6. Car Waits for Pedestrian

For Pedestrians Module, states, rewards and q-values are identical to traffic signal module. Just instead of checking for red traffic signal ahead, now you have to check for Pedestrians crossing the road ahead. Result is shown in Figure 6.

2.1.4 Avoiding Static Obstacles

In this module, we teach agent to change the lane, when there is a static obstacle such as pothole in the current lane.

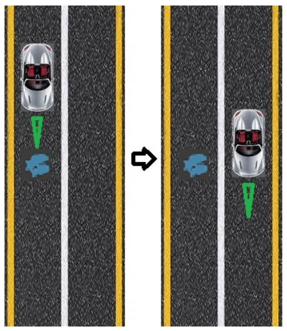

Figure 7. Car Avoids Static Obstacles

Again, we will define two states. First one is there is obstacle ahead and second one is no obstacle ahead. Define rewards as follows, which penalizes agent for going through obstacle.

Table 5. Rewards - Avoiding Static Obstacles

State/Action	Forward	Left	Right	Stop
Obstacle	-300	100	100	-50
No Obstacle	1	-1	-1	-1

From the rewards 5, agent learns to change the lane when obstacle is there. It takes about 5-6 episodes for it to learn. It starts with 48 steps and -1 reward in Episode 0, but reaches optimal level quickly with 128 points and 20 steps.

Table 6. Q-Values - Avoiding Static Obstacles

State/Action	Forward	Left	Right	Stop
Obstacle	0	104.99	104.99	0
No Obstacle	13.2	4	4	3.96

From Q values 6, learned we can see that agent will take the actions Left or Right when obstacle is ahead to change the lane. And will move forward, otherwise. Result is shown in Figure 7.

2.1.5 Avoiding Dynamic Obstacles

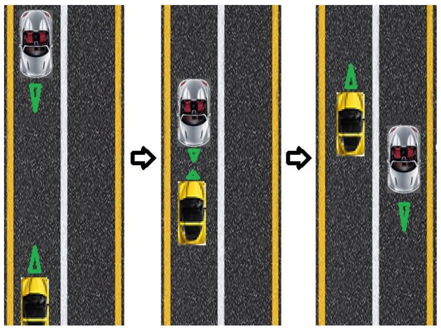

Figure 8. Car Avoids Dynamic Obstacles

Here, we will deal with other cars in the environment. They can have different velocities. So, what we will do is keep checking whether a car is in same lane as us with in finite (let's say 5) steps from now. If so, we need to change the lane. Otherwise we can continue. Here are the rewards for the same.

Table 7. Rewards - Avoiding Dynamic Obstacles

State/Action	Forward	Left	Right	Stop
Cars	-30	5	5	-30
No Cars	5	-5	-5	-1

Here it very quickly learns to avoid cars. Rewards are shown in 7. It starts with 21 steps and 40 points and at end it terminates in 20 steps and 75 points, which is optimal value.

Table 8. Q-Values - Avoiding Dynamic Obstacles

State/Action	Forward	Left	Right	Stop
Cars	-25	10	10	-20
No Cars	14	0	0	0

From Q values 8, learned we can see that agent will take the actions Left or Right when some car is there in the same lane ahead to change the lane. And will move forward, otherwise. Result is shown in Figure 8.

2.2. Combining Modules

Now, we are done with Modular training. We have taught agent to how to tackle different situations separately. In real world scenario, all of these will be happening simultaneously. So, let's look at how we can merge this modules.

At each step, we sense different parameters discussed previously such as traffic signal ahead, pedestrian ahead, obstacle ahead, car ahead etc. Now, for that particular situation, we retrieve the q-values from each of the different modules and do a weighted average of them. After doing that, the action with the highest weighted average is chosen to be performed.

For our five modules, we combine the Q-Values with following weights: Moving Forward (0.1), Following Traffic Signal (0.2), Waiting For Pedestrians (0.2), Avoiding Static Obstacles (0.25) and Avoiding Dynamic Obstacles (0.25). Based on these values, we will choose which action to perform.

2.3. Testing

Now, let's test our agent, which will choose an action based on the strategy mentioned in the previous section.

Look at Figure 9. Here, our agent (Car) encounters a traffic signal, a pothole, a pedestrian and a moving car simultaneously. Though we trained different modules for different tasks, by combining them, Agent should be able to deal with all of them successfully. We can see that initially it stops at red traffic signal. As signal goes green, it encounters a pothole and changes its lane. After some distance, it waits for a pedestrian to cross the road. And then it faces a car coming from opposite direction in the same lane, so again it changes the lane. Finally, it successfully reaches the other end of the road, which was our goal!

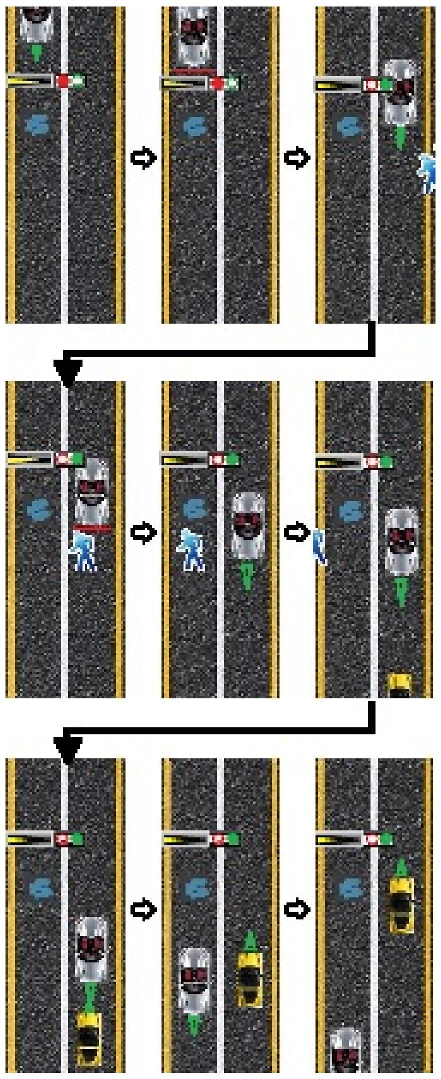
Figure 9. Testing - Self-Driving Car in Action!

3. Conclusion

In this assignment, we implemented a simple reinforcement learning algorithm - Q-Learning to model a basic self driving car. It was trained to move forward, stop at Red traffic signal, wait for pedestrians to cross road and avoid static and dynamic obstacles. Different modules were trained separately to perform all these different tasks. All of them were combined into one modular agent, which choses Action based on highest value of, weighted average of Q-values of different modules. During testing, we found that Agent was able to successfully complete all the tasks simultaneously.