Udacity Machine Learning Nanodegree
Reinforcement Learning Project
Train a Smartcab to Drive

Veeresh Taranalli

March 9, 2016

1 Implementation of a Basic Driving Agent

Q. Mention what you see in the agent's behavior. Does it eventually make it to the target location?

A. Because we choose a random action in every state, even though the agent is allowed a lot of time in simulations, it rarely makes it to the correct destination. As the agent's world (road grid) is quite small in our example, there is a small probability that it will eventually make it to the target location.

2 Identify and Update State

Q. Justify why you picked these set of states, and how they model the agent and its environment.

A. The fundamental underlying assumption for the Q-Learning algorithm to work is practice is that the learning agent gets to visit all the states infinite number of times to compute the expected utility of states accurately. But in practice, the agent has to learn in finite time with partially observable state information. Therefore, if we have too many states, the agent might not be able to visit all the states in the given time. If we have too few states, the agent might not be able to distinguish certain states which call for truly different actions and hence would fail.

Therefore, keeping this mind we proceed to design the states for our learning agent as follows.

The following inputs are available to our learning agent from its environment which can be utilized to define states:

```
* traffic light: [Red, Green]
* oncoming traffic direction: [None, Forward, Left, Right]
* from left traffic direction: [None, Forward, Left, Right]
* from right traffic direction: [None, Forward, Left, Right]
* next waypoint: [Forward, Left, Right]
* deadline: integer number
```

As deadline is an integer number and keeps changing over time, it is clear that it should not be made a part of the state for our learning agent. Using all the remaining inputs, we would have 2 x 4 x 4 x 4 x 3 = 384 states. Given that the learning agent needs to visit all possible states multiple times in about 4000 time steps (assuming average of 40 available steps per trial and 100 trials) to correctly learn an optimal policy, defining 384 different states for our learning agent is not the correct choice.

However, we also can utilize our a-priori knowledge of traffic/right-of-way rules (USA) to reduce the number of possible states for our learning agent. The state reduction is achieved as follows:

If oncoming traffic is either 'None' or headed 'Right', it really does not affect our learning agent's decision and hence we can just map these two values to a single value called 'Safe'.

Similarly only traffic from left that is headed 'Forward' affects the decision of our learning agent. Therefore we map the remaining three values to a single value called 'Safe'.

For our learning agent, it does not matter which way the traffic from right is headed. Hence we do not use this input at all.

Therefore our state is defined as a tuple (traffic light, Variable 1, Variable 2, next waypoint) which can take the following possible values:

```
* traffic light: [Red, Green]
* Variable 1 (oncoming): [Safe, Forward, Left]
* Variable 2 (left): [Safe, Forward]
* next waypoint: [Forward, Left, Right]
```

The number of states has come down to 2 x 3 x 2 x 3 = 36! which can easily be handled using 100 trials and in a way this reduced set of states do represent the set of inputs a person would use while driving assuming US right-of-way rules.

3 Implement Q-Learning

Q. What changes do you notice in the agent's behavior?

A. We implement the Q-Learning algorithm and in each state select an action with the largest Q-Value. The Q-Learning algorithm is shown below

$$Q(s_k, a_k) = (1-\alpha)Q(s_k, a_k) + \alpha\Big(R(s_k, a_k) + \gamma \max_a Q(s_{k+1}, a)\Big)$$

where s_k, a_k is the state action pair at time k, $R(s_k, a_k)$ is the associated reward, α is the learning rate parameter and γ is the discount parameter.

For the learning agent, a right turn always results in a positive reward regardless of its state and hence once the right turn action has been selected using Q-Values, this decision reinforces the rewards and subsequently the learning agent ends up always choosing the right turn regardless of the state. The result is that the agent is stuck in a loop on the grid as we observe very clearly in the pygame simulator.

This problem leads to the ϵ-greedy learning approach where the learning agent gets to pick a random action independent of its state with a small probability. This kind of stochasticity is essential for the agent to get out of a loop as discussed above. The ϵ-greedy learning approach and other learning parameter optimizations are presented in the next section.

4 Enhance the Driving Agent

Q. Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform? Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties?

A. We implement the ϵ-greedy learning approach where with probability ϵ the learning agent chooses a random action and with probability $1 - \epsilon$ it chooses the best action according to the values computed by the Q-Learning algorithm. This allows the agent to visit previously unvisited states and gain 'experience and learning'. Typically a small value of ϵ will work. We fix the value of $\epsilon = 0.1$ as it seems reasonable for the current setup which results in 1 random action in 10 steps. The typical deadlines in each trial are around 40 time steps (which in turn depends on the grid size). Using the fixed ϵ parameter, we count the number of successes of the agent in 100 trials for different values of α and γ parameters as shown in Fig. 1. The agent ends up with positive reward in almost all the trials for all the parameters. α is the learning rate parameter directly controlling how fast the agent learns. γ is the discount which basically indicates if the agent should prioritize immediate reward or long-term reward. γ being close to 1 means the agent prioritizes long-term reward, while γ being close to 0 means the agent prioritizes immediate reward. From Fig. 1 we see that γ being too close to either 0 or 1 is not favorable which makes sense. Hence we choose $\gamma = 0.4$. We also see that increased learning rate is generally favorable, thus we choose a learning rate of $\alpha = 0.9$ for our final agent. Using these values of α and γ, we look at the number of successful trials in 100 trials for different values of ϵ as shown below:

$$\epsilon = 0.01 \rightarrow 66$$
$$\epsilon = 0.05 \rightarrow 89$$
$$\epsilon = 0.10 \rightarrow 95$$
$$\epsilon = 0.15 \rightarrow 90$$
$$\epsilon = 0.20 \rightarrow 83$$

Therefore ϵ value around 0.1 seems to be the best as chosen earlier and this will be used for our final agent. Using these chosen values of α, γ and ϵ parameters, we observe the performance of the agent in terms of actual steps v/s alloted steps as shown in Fig. 2. This is to evaluate if our agent has learned the optimal policy. We observe that the agent is able to reach its destination much earlier than the alloted deadline consistently. We also count the number of times our agent receives a negative reward for its action. In one such experiment with 100 trials, our agent receives negative rewards for only 4 steps. A negative reward is received when the agent breaks the traffic signals. Therefore these observations show that our agent has learnt a close to optimal policy.

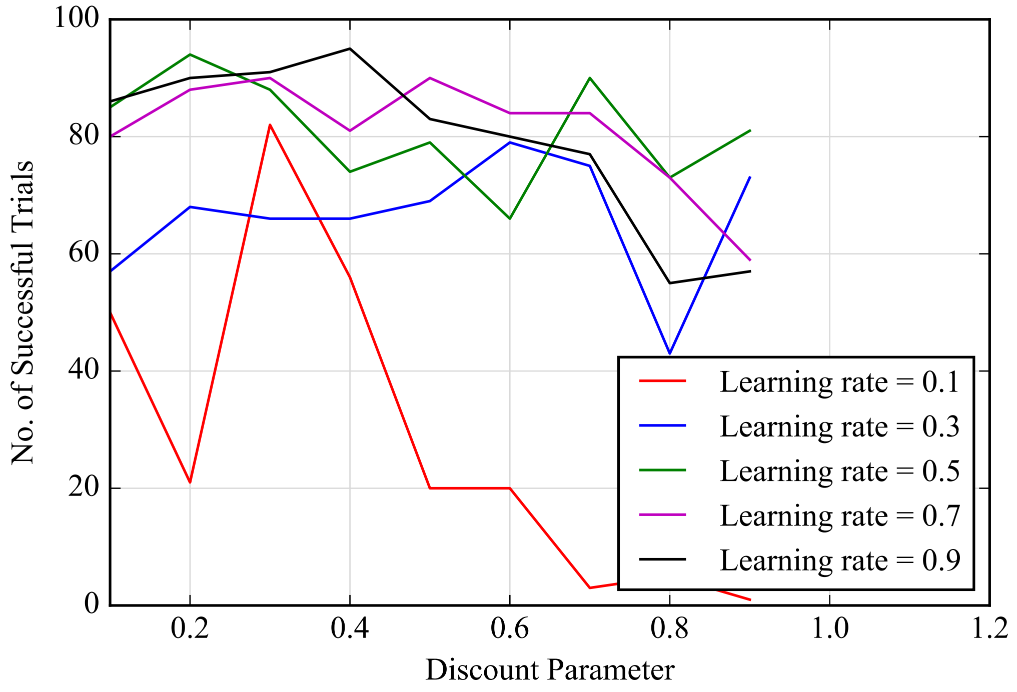

Figure 1: Plot of number of successful trials out of 100 trials of the learning agent versus the learning rate parameter and the discount parameter. $\epsilon = 0.1$.

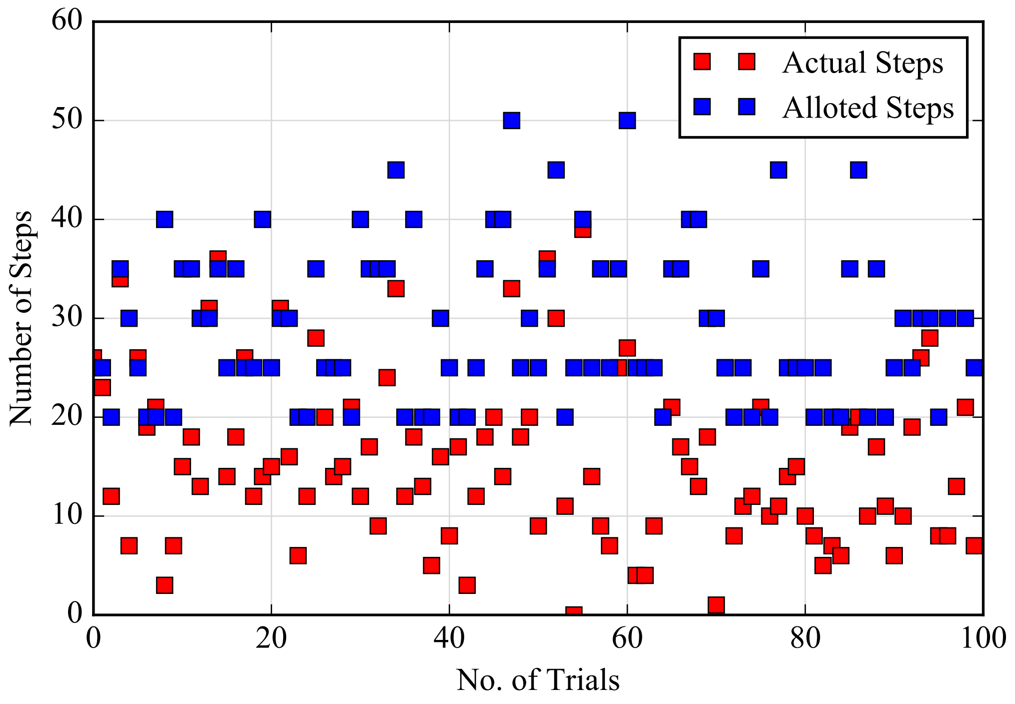

Figure 2: Plot of number of actual and alloted steps for all 100 trials of the learning agent. $\epsilon = 0.1$.