MLND: Training a Smart Cab

Gendith M. Sardane

admin@adalace.org ‖ *Github://gmsardane*

August 31, 2016

1 Overview

In the not-so-distant future, taxicab companies across the United States no longer employ human drivers to operate their fleet of vehicles. Instead, the taxicabs are operated by self-driving agents known as smartcabs to transport people from one location to another within the cities those companies operate. In major metropolitan areas, such as Chicago, New York City, and San Francisco, an increasing number of people have come to rely on smartcabs to get to where they need to go as safely and efficiently as possible. Although smartcabs have become the transport of choice, concerns have arose that a self-driving agent might not be as safe or efficient as human drivers, particularly when considering city traffic lights and other vehicles. To alleviate these concerns, your task as an employee for a national taxicab company is to use reinforcement learning techniques to construct a demonstration of a smartcab operating in real-time to prove that both safety and efficiency can be achieved.

1.1 Definitions

1.1.1 Environment

The smartcab operates in an ideal, grid-like city (similar to New York City), with roads going in the North-South and East-West directions. Other vehicles will certainly be present on the road, but there will be no pedestrians to be concerned with. At each intersection there is a traffic light that either allows traffic in the North-South direction or the East-West direction. U.S. Right-of-Way rules apply:

On a green light, a left turn is permitted if there is no oncoming traffic making a right turn or coming straight through the intersection. On a red light, a right turn is permitted if no oncoming traffic is approaching from your left through the intersection. To understand how to correctly yield to oncoming traffic when turning left, you may refer to this official drivers education video , or this passionate exposition.

1.1.2 Inputs and Outputs

Assume that the smartcab is assigned a route plan based on the passengers starting location and destination. The route is split at each intersection into waypoints, and you may assume that the smartcab, at any instant, is at some intersection in the world. Therefore, the next waypoint to the destination, assuming the destination has not already been reached, is one intersection away in one direction (North, South, East, or West). The smartcab has only an egocentric view of the intersection it is at: It can determine the state of the traffic light for its direction of movement, and whether there is a vehicle at the intersection for each of the oncoming directions. For each action, the smartcab may either idle at the intersection, or drive to the next intersection to the left, right, or ahead of it. Finally, each trip has a time to reach the destination which decreases for each action taken (the passengers want to get there quickly). If the allotted time becomes zero before reaching the destination, the trip has failed.

1.1.3 Rewards and Goal

The smartcab receives a reward for each successfully completed trip, and also receives a smaller reward for each action it executes successfully that obeys traffic rules. The smartcab receives a small penalty for any incorrect action, and a larger penalty for any action that violates traffic rules or causes an accident with another vehicle. Based on the rewards and penalties the smartcab receives, the self-driving agent implementation should learn an optimal policy for driving on the city roads while obeying traffic rules, avoiding accidents, and reaching passengers destinations in the allotted time.

2 Implement a Basic Driving Agent

To begin, your only task is to get the smartcab to move around in the environment. At this point, you will not be concerned with any sort of optimal driving policy. Note that the driving agent is given the following information at each intersection:

The next waypoint location relative to its current location and heading. The state of the traffic light at the intersection and the presence of oncoming vehicles from other directions. The current time left from the allotted deadline.

To complete this task, simply have your driving agent choose a **random action** from the set of possible actions (None, 'forward', 'left', 'right') at each intersection, disregarding the input information above. Set the simulation deadline enforcement, **enforce_deadline = False** and observe how it performs.

2.1 Question

Observe what you see with the agent's behavior as it takes random actions. Does the smartcab eventually make it to the destination? Are there any other interesting observations to note?

After 100 trials of this set up, the (not-so-smart) cab reached its destination 62 times (within the alloted *hard* deadline). In sixteen instances, the cab reached the destination within deadline, obtaining a **reward = 12**. The deadline is defined as $5\times$ the Manhattan (L_1) distance between the starting and end points. The table below summarizes the results from the last 10 trials. From the table only once [$Ntrial = 99$] did the car get to destination on time. This random walk scenario is equivalent to the robot in constant exploration mode, but never learning and capitalizing from the results of its experiences.

3 Inform the Driving Agent

Now that your driving agent is capable of moving around in the environment, your next task is to identify a set of states that are appropriate for modeling the smartcab and environment. The main source of state variables are the current inputs at the intersection, but not all may require representation. You may choose to explicitly define states, or use some combination of inputs as an implicit state. At each time step, process the inputs and update the agent's current state using the self.state variable. Continue with the simulation deadline enforcement enforce_deadline being set to False, and observe how your driving agent now reports the change in state as the simulation progresses.

Table 1 Results from the last 10 trials.

NTrial	Start	Destination	t_{end}	Position at t_{end}	Reached Destination w/in t_end
90	(4,1)	(2,3)	-100	(3, 1)	No
91	(1, 2)	(1, 6)	-100	(8, 2)	No
92	(4, 4)	(8, 6)	-97	(8, 6)	No
93	(8, 3)	(5,1)	-100	(3,3)	No
94	(1,5)	(6,5)	-100	(3,4)	No
95	(6,3)	(8,5)	-58	(8,5)	No
96	(1,1)	(4,2)	-100	(8,2)	No
97	(3,5)	(7,1)	-30	(7,1)	No
98	(8,1)	(4,1)	-100	(6,5)	No
99	(2,2)	(8,3)	48	(8,3)	**Yes**

3.1 QUESTION

What states have you identified that are appropriate for modeling the smartcab and environment? Why do you believe each of these states to be appropriate for this problem? OPTIONAL: How many states in total exist for the smartcab in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?

For this problem, the relevant observables in the environment of the cab are:

- **Light** - the state of the traffic light at the location of the robot, {red, green}
- **Next_Waypoint** - the location of the next waypoint relative to where the destination is and where the robot is headed. Assumes three possible values: **left, right, forward**
- **Left** - whether a vehicle is approaching from the *left*, and can have four possible values, **left, right, oncoming, None**, depending on the direction this other vehicle is headed.
- **Right** - whether a vehicle is approaching from the *right*, and can have four possible values, **left, right, oncoming, None**, depending on the direction this vehicle other is headed.
- **Oncoming** - whether a vehicle is approaching the robot from the *forward* direction. Four values are possible: **left, right, oncoming, None**, depending on where this vehicle is headed.
- **Deadline** - The amount of time the cab needs to reach destination. This is defined as 5 $\times$ the Manhattan distance between the starting and end points. The parameter is relevant, as we would all want, in the real world, a smartcab to reach the destination in a timely fashion. Otherwise, there would be no incentive in getting a smartcab if it takes "forever" to get to where one needs to be.

In designing a self-driving vehicle, safety is a primary concern. Hence, it is important to be as close to the real-world driving situation as much as possible. With these factors in mind, I am choosing **light, oncoming, left, right, and next_waypoint** as the relevant observables defining the state.

The **next_waypoint** is the direction where the cab should move next and the **left, right, oncoming** variables tell the robot if a car is nearby, and hence are necessary pieces of information to model (many) real-world driving scenarios.

In total *my* smartcab can have $2 \times 4 \times 4 \times 4 \times 3 = 384$ possible combinations of states. The **action** the cab can take have four possibilities: **None, left, right, forward**. This results to a total of $1,536$ state-action pairs.

If the deadline becomes a state parameter to consider, a grid-size of 8×6 implies a separation distance that can range from $\Delta r = 1 - 12$ units, corresponding to deadlines that range from [5, 60]. This leads to an additional of 54 more states, resulting to $54 \times 1536 = 82,944$ possible state-action pairs! A scenario would not be practical given the current set-up. Moreover, having such a large Q-table would not be as useful, since many of these states would not even be visited given $n_{trials} = 100$.

For practical purposes, I choose *not* to use the deadline as a state observable. Since the starting position and destinations are randomly chosen variables, the deadline becomes a state variable that is randomly chosen as well. Including the deadline in the list will make the Q-table to be unnecessarily large and sparse.

4 Implement a Q-Learning Driving Agent

With your driving agent being capable of interpreting the input information and having a mapping of environmental states, your next task is to implement the Q-Learning algorithm for your driving agent to choose the best action at each time step, based on the Q-values for the current state and action. Each action taken by the smartcab will produce a reward which depends on the state of the environment. The Q-Learning driving agent will need to consider these rewards when updating the Q-values. Once implemented, set the simulation deadline enforcement **enforce_deadline = True**. Run the simulation and observe how the smartcab moves about the environment in each trial.

4.1 QUESTION

What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?

4.1.1 Q-learning Algorithm

My implementation of the Q-learning algorithm starts by creating a dictionary of all possible state-action pairs (i.e. 1536), with each element initialized to zero. As the simulation progresses, the agent will eventually pick the action that gives it the most utility. I started with a scenario with intermediate parameters: $(\alpha, \gamma) = (0.5, 0.5)$.

In comparison to the basic driving agent, the Q-learning agent is able to obey traffic rules quickly, as well as reach the destination in time. It is also quick to learn good driving, and is evolving to be more efficient. I simulated the process 1000 times, and got 978 successful trials, equivalent to a 97.8% success rate. Figure 1 shows a distribution of the fraction of time left before deadline for those all successful trials. On average, the smartcab finishes with roughly 17 more tries left before the deadline.

This result demonstrates the basic idea of Q-learning. The agent gets to choose that best possible decision based on prior information it had gathered via incentivizing "good behaviour". It learns to obey traffic rules, correctly implement the required action, and reach the destination in time — and with all of these recorded in the annals of its Q-table.

5 Improve the Q-Learning Driving Agent

Your final task for this project is to enhance your driving agent so that, after sufficient training, the smartcab is able to reach the destination within the allotted time safely and efficiently. Parameters in the Q-Learning algorithm, such as the learning rate (alpha), the discount factor (gamma) and the exploration rate (epsilon) all contribute to the driving agents ability to learn the best action for each state. To improve on the success of your smartcab:

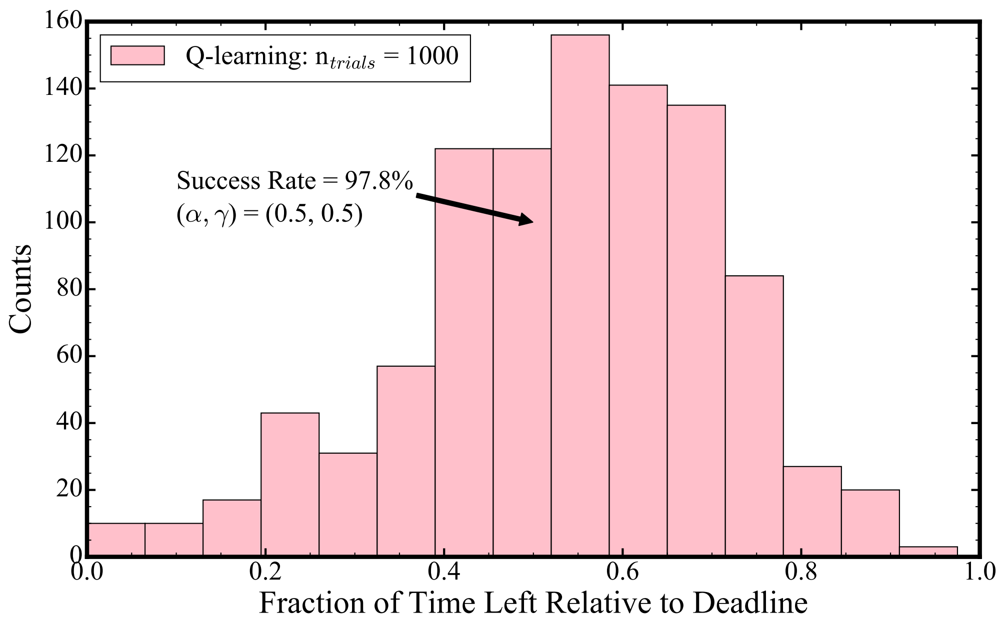

Figure 1 Q-learning smartcab with a learning rate, $\alpha = 0.5$, and a discount factor, $\gamma = 0.5$. The process is simulated 1000 times.

- Set the number of trials, n_trials, in the simulation to 100.
- Run the simulation with the deadline enforcement enforce_deadline set to True (you will need to reduce the update delay update_delay and set the display to False).
- Observe the driving agents learning and smartcabs success rate, particularly during the later trials.
- Adjust one or several of the above parameters and iterate this process.

This task is complete once you have arrived at what you determine is the best combination of parameters required for your driving agent to learn successfully.

5.1 Question

Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?

To illustrate the performance of the Q-learning agent, I will present a series of plots for example values of the parameters, α and γ. In addition to the success rate, one thing I decided to track, as an important performance measure, is in terms of how quickly does the agent reach its destination, if in case it does. This is quantified via of the fraction of time, relative to the total available time per trial, that agent has left upon successfully completing a trip. The succeeding plots will show the distributions of this quantity at various combinations of α and γ.

Figure 2 shows the results for $(\alpha, \gamma) = (1.0, 1.0)$. This is a long-term-centric agent. The success rate slightly increased to 98.0%.

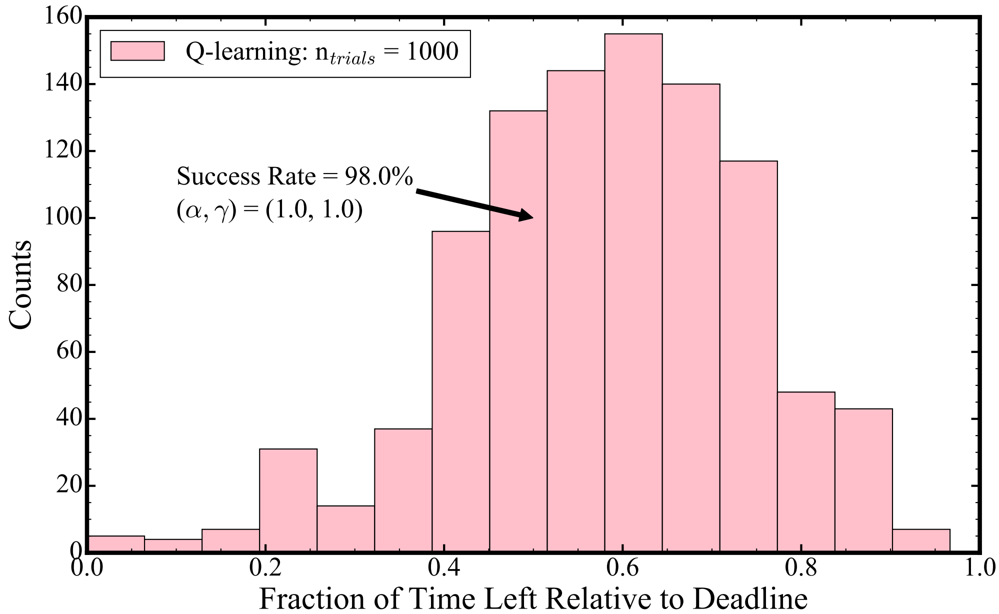

Figure 2 Q-learning smartcab with using a learning rate $\alpha = 1.0$ and a discount factor $\gamma = 1.0$. The process is simulated 1000 times.

Figure 3 shows the results for $(\alpha, \gamma) = (0.5, 0.1)$. This is a short-term-centric (myopic) agent. The success rate increased to 99.3%.

Figure 4 shows the results for $(\alpha, \gamma) = (0.5, 1.0)$. This is a short-term-centric (myopic) agent. The success rate decreased to 97.5%.

There are a couple others that I tried, but for brevity, will only be showing the combination of parameters giving the best success rate. This is for the case when $(\alpha, \gamma) = (0.75, 0.25)$. This results in a remarkable success rate of 99.5% (i.e. 995 out of 1000 trials, the robot reaches its destination!). On average, the robot arrives to its destination, with roughly 62% of the available time remaining. Figure 5 shows this distribution for all 995 successful trials.
Note: In cases when two or more states bear the same best Q-value, the tie is broken by randomly choosing an action amongst the ties.

5.1.1 On the Initial Q_0

The examples discussed above, all have initial Q-values set to zero. In this subsection, we explore how setting a (high) non-zero initial Q-value, which encourages exploration, change the results. The effects of tweaking Q_0 is shown in Figures 6 - 7, witf $(\alpha, \gamma) = (0.75,\ 0.25)$ at an initial $Q_0 = 10$ and 20, respectively. Both cases show a slightly reduced success rate of $\sim$99% in 1000 trials. Note that in both (and in all) cases, the agent had roughly 60% of its alloted time remaining. Although the lower success rate, the last failed trip occurred at $\lesssim 450^{th}$ trial.

5.2 QUESTION

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? How would you describe an optimal policy for this

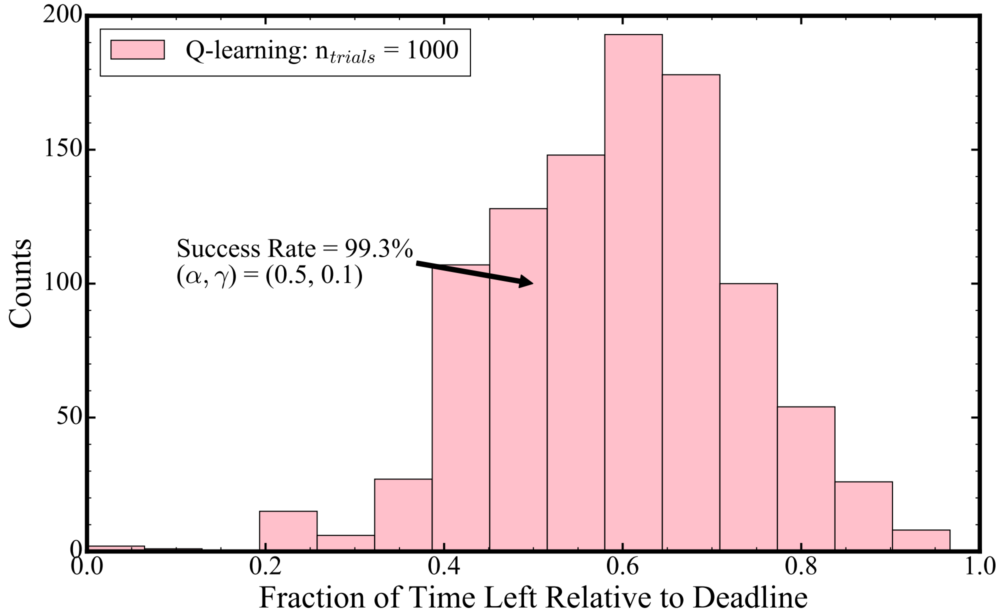

Figure 3 Q-learning smartcab with using a learning rate, $\alpha = 0.5$, and a discount factor $\gamma = 0.1$. The process is simulated 1000 times.

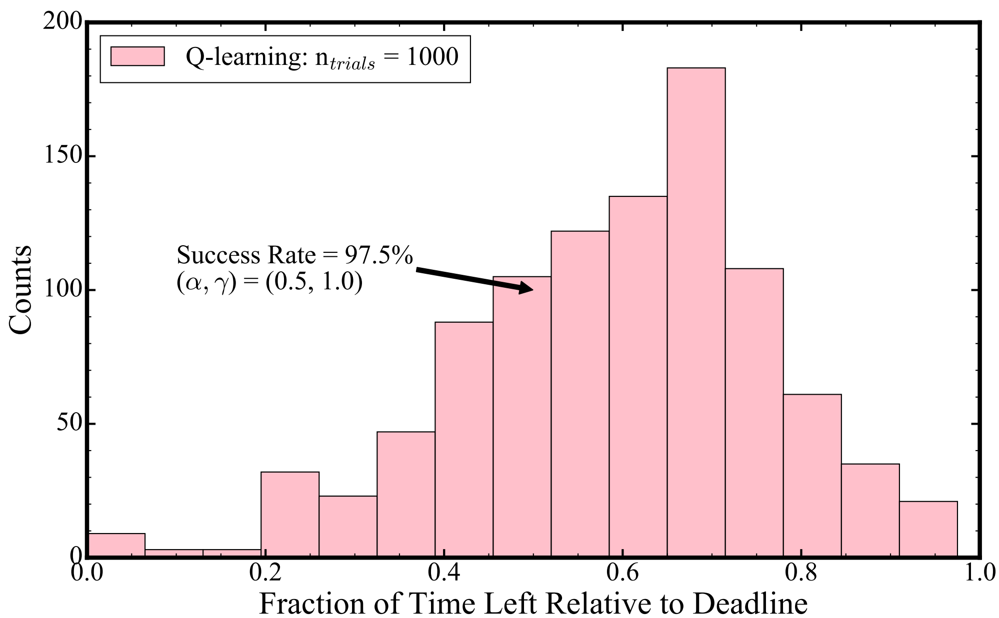

Figure 4 Q-learning smartcab with using a learning rate, $\alpha = 0.5$, and a discount factor, $\gamma = 1.0$. The process is simulated 1000 times.

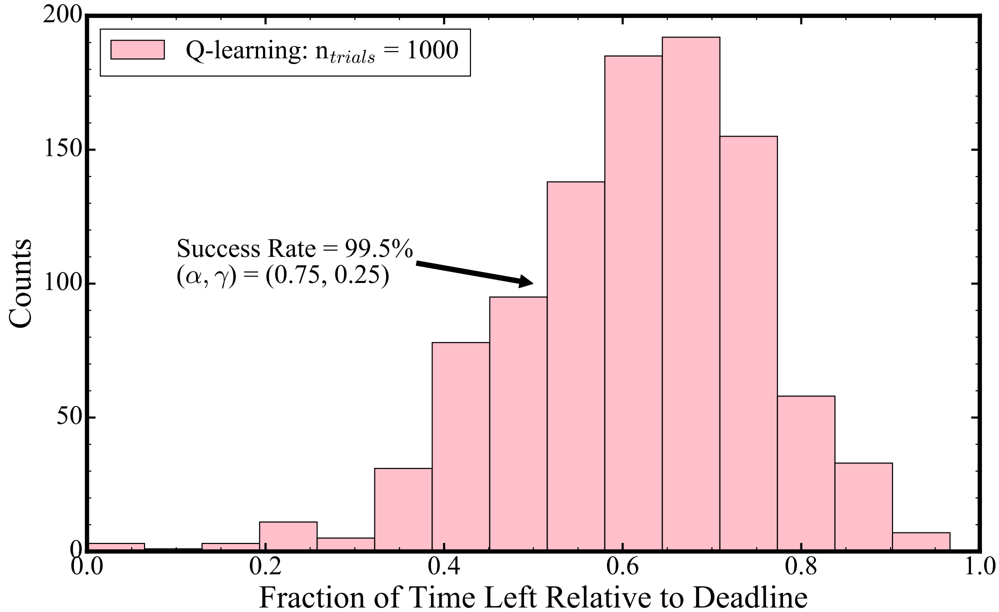

Figure 5 Q-learning smartcab with using a learning rate $\alpha = 0.75$ and a discount factor $\gamma = 0.25$. The process is simulated 1000 times.

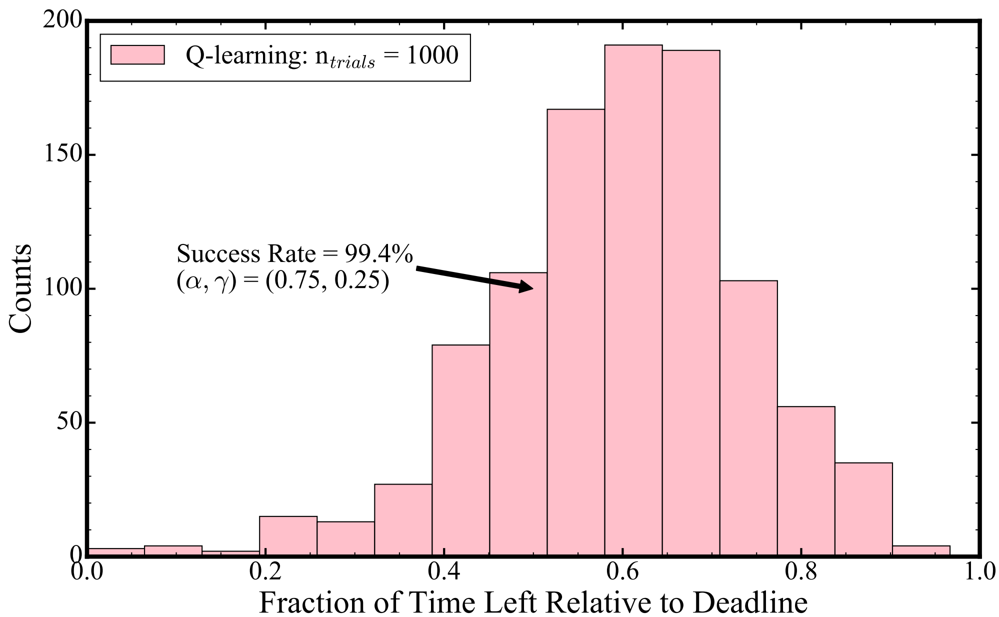

Figure 6 Q-learning smartcab with using a learning rate $\alpha = 0.75$, a discount factor $\gamma = 0.25$, and $Q_0 = 10.0$. The process is simulated 1000 times.

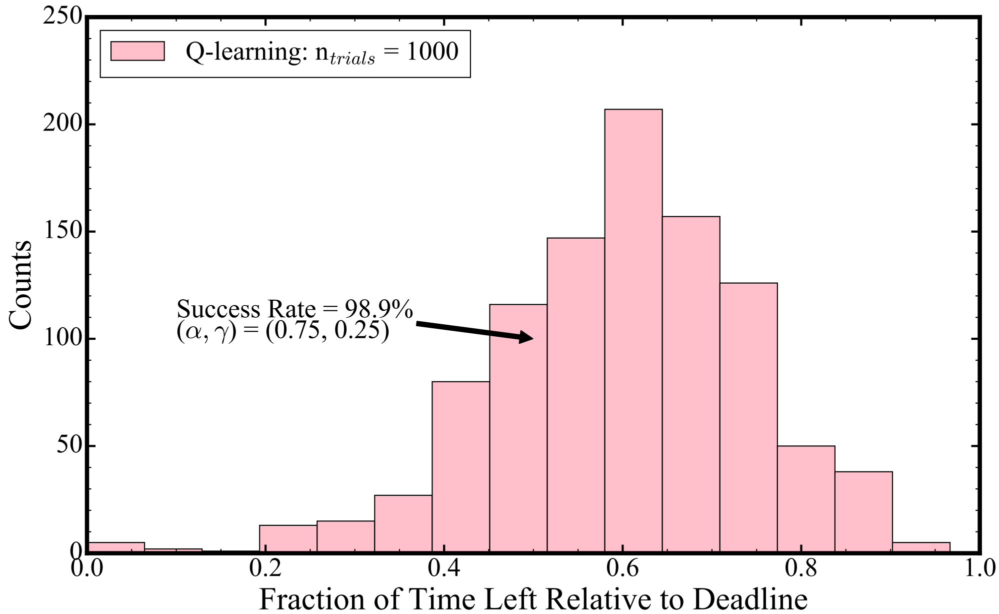

Figure 7 Q-learning smartcab with using a learning rate $\alpha = 0.5$ and a discount factor $\gamma = 0.1$, and $Q_0 = 20.0$. The process is simulated 1000 times.

problem?

As discussed above, the best performance is obtained by using $(\alpha, \gamma) = (0.75, 0.25)$.

I re-ran the agent for 100 trials at the optimal parameter values. The success rate was at 100%, reaching the destination on time at cumulative positive rewards per trial (see Figure 8). Zooming into the last 20 trials, the agent has especially learned to follow traffic rules: stopping at a red light and not making a left at a red light, and in the presence of oncoming vehicle(s), and proceeds according to where the next waypoint is. (This behaviour becomes apparent right at the first trials, which to me is quite remarkable.)

Over the final ten trials, the agent incurred a negative reward of -0.5 at the 91st trial. Without any other vehicle at the vicinity, the agent made a **right** action at a *red* light, despite having a *left* next waypoint.

An optimal policy for the agent is one that ***both follows traffic rules, and proceeds in the direction indicated by the next_waypoint parameter (i.e. follows the optimal path to destination)*, thereby reaching the destination efficiently** (i.e not going around in circles until time runs out). Observations of the last ten trials indicate that the agent was indeed able to learn this optimal policy, gaining all positive cumulative rewards upon reaching the destination.

From the last 10 trials, there were two instances when the agent was not (or not yet) able to proceed to where next waypoint is:

One is when the traffic **light** = red, the **next_waypoint** = **forward**, so that **action** = **None**, earning a zero reward. Note that a red traffic light is a major limiting factor in terms of how fast the agent reaches the deadline, at least for this simulation, where only three other cars are present.

The other is, as described above, it moved in the opposite direction of the next waypoint, but without violating traffic rules, earning a penalty of -0.5. Also, the agent reached destination with more tha 60% of the time left. Such behaviour was not apparent at the few early trials.

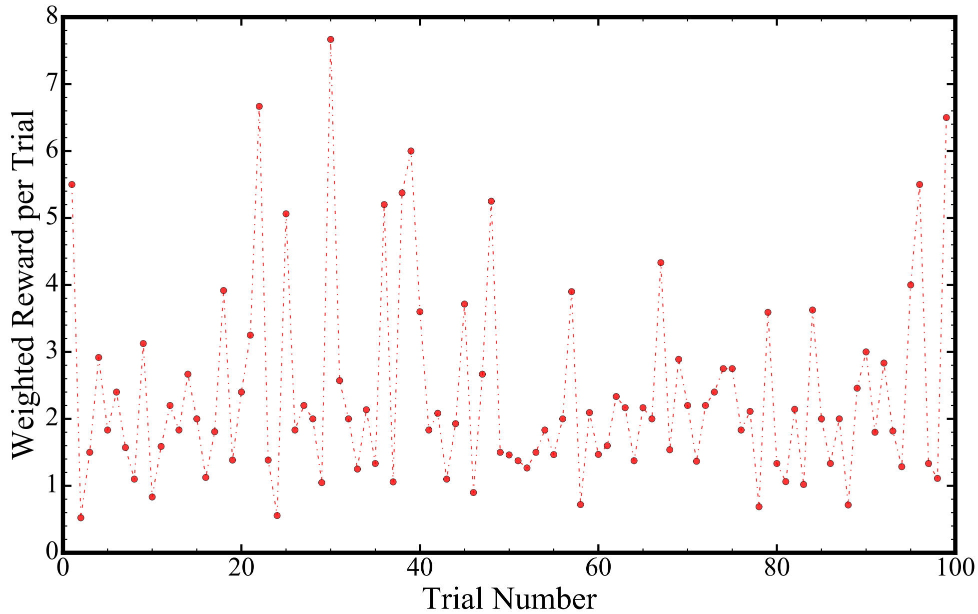

Figure 8 The evolution of the cumulative reward gained by the agent at the end of each trial, weighted by the time it took to reach the deadline.

6 Summary of Results and Conclusion

To summarize, the highest success rate is achieved by the following combination of parameters: ($\alpha = 0.75$, $\gamma = 0.25$, $Q_0 = 0$). The differences, however, for this combination is not too far from the cases where $Q_0 =$ [10,20]. The Q-learning agent has shown remarkable success rates. Any of these three combinations would be quite satisfactory.

7 References

- Udacity Forums, and references therein
- http://www.cs.utexas.edu/∼dana/MLClass/
- http://artint.info/html/ArtInt_266.html