report

October 13, 2016

1 Train a Smartcab to Drive

1.1 Reinforcement learning

1.1.1 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 to False and observe how it performs.

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?

Answer:

The angent's behaviour is, unsurprisingly, random. It would at times move in a circle (around the block) not really progressing in any net new direction. And it doesn't appear to have any logic behind its movement as rewards for previous states and actions are ignored. Another item of the agent's behavior is the agent can also drive over the right edge of the world and turn up on the left edge of the world. However, the smartcab does eventually make it to its destination on occasion.

1.1.2 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.

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?

Answer:

At each state or waypoint we perceive 6 inputs, left, right, oncoming, light, next_waypoint and deadline. Since traffic coming from the right, doesn't affect the smartcab's decision, this variable was removed - e.g. if the light is green, traffic from the right has stopped and can be ignored. Likewise, if light is red, the smartcab can only turn right and any traffic coming from the right doesn't affect the agent's options. The input deadline was not utilized because at this point it does not provide any significant information to the smartcab (enforce_deadline set to false).

Each state identifies a situation and option for the smartcab, which results in a particular reward. Once Q-learning is implemented (below) the smartcab will be able to use these states to learn and best determine its path to the destination.

Final states used are light, oncoming, left and next_waypoint, each of which are 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?

Answer:

Inputs and their possible values:

- Light: Red or green
- Oncoming: None, left, right, forward
- Left: None, left, right, forward
- Next_waypoint: None, left, right, forward

Capturing 4 inputs per state with each input having a variation between 2-4, gives us, 2x4x4x4, 128 states in total that exist for the smartcab in this environment.

1.1.3 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 to True. Run the simulation and observe how the smartcab moves about the environment in each trial.

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?

Answer:

At the beginning of trials the agent's behavior remains the same; however, with larger "n_trials" (more trials), the agent no longer seems to act randomly but its movement tends towards the destination. It also appears more "structured" or intentional instead of moving randomly as time progresses.

The agent also reaching its destination more often and seemingly quicker, even though the deadline was disabled during the random trials.

Metrics used to track the agent's performance are:

- Reward: Total reward, sum of positive and negative rewards received at each state
- Win: The number of times the agent reaches its destination (deadline is enforced)
- Lose: The number of times the agent runs out of time without reaching its destination (deadline is enforced)
- Penalties: Total negative reward, sum of negative rewards received

This behvaior occurs because it is learning from its past experience. The q-table starts off empty and as the agent explores its world, the q-table is being updated based on the outcome of each state, action and reward. Then, if the agent comes across a similiar state (already stored in the q-table) the agent will utilize this knowledge, helping the agent make "better" decisions that maximize reward.

The tables below show the results of 10 Runs of 50 Trials (n_trials = 50) each, for both random actions and those of the implemented Q-learning algorithmn.

Random actions: Average Rewards: 7.6, Win: 11, Lose: 39, Penalties: -568

Rewards	Win	Lose	Penalties
49.0	11	39	-589.5
-34.5	13	37	-593.0
-31	7	43	-558.0
49.5	14	36	-512.5
5	10	40	-587.0

Q-Learning: Average Rewards: 1059, Win: 40, Lose: 10, Penalties: -164

Rewards	Win	Lose	Penalties
887.5	36	14	-155.0
1016.0	41	9	-195.5
1154.5	47	3	-137.5
1097.0	36	14	-145.0
1140.5	42	8	-185.5

1.1.4 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 agent's ability to learn the best action for each state. To improve on the success of your smartcab:

- 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 agent's learning and smartcab's 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.

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?

Answer:

Mainly 3 parameters were tuned in order to find the best combination where the agent performs best, these parameters are and how they impact the agent is described below:

- Alpha (learning rate), epsilon (exploration rate) and gamma (discount factor) were utilized.
- If alpha is too small, learning takes too long, too large and learning will put too much importance on the most recent action
- If gamma is too small, too much importance is placed on the current reward, too large and emphasis is placed on the future reward

The agent performed best with the following set of parameters: alpha = 0.6, epsilon = 0.1, gamma = 0.1. At it's most optimal, the driving agent fails to reach its destination only once out of its 100 trials.

Tables below are 3 samples from multiple runs (5 runs, 100 trials) experimenting with alpha, epsilon and gamma values:

alpha = 0.2, epsilon = 0.1, gamma = 0.4

Rewards	Win	Lose	Penalties
2231.5	87	13	-255.5

Rewards	Win	Lose	Penalties
2226.0	86	14	-271.5
2083.5	83	17	-230.0
2068.0	93	7	-265.0
2056.5	82	18	-316.5

alpha = 0.4, epsilon = 0.3, gamma = 0.1

Rewards	Win	Lose	Penalties
2033.5	82	18	-402.0
2059.5	81	19	-408.0
1989.5	75	25	-369.5
2031.0	81	19	-396.5
1991.0	79	21	-380.5

alpha = 0.6, epsilon = 0.1, gamma = 0.1

Rewards	Win	Lose	Penalties
2253.0	95	5	-119.0
2300.5	96	4	-115.0
2180.5	94	6	-145.5
2233.5	95	5	-108.5
2275.5	99	1	-96.5

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 problem?

Answer:

Optimal policy: Always get to your destination without incurring penalties (illegal actions e.g. driving through a red light) and do it in the shortest amount of time.

The optimal / learned policy for this problem is realitively balanced between gaining maximium reward and always arriving at it's destination but it does incur penalities to do so. However, the learned policy does have a much lower rate of incurred penalities during higher number of trials.

Based on the above, and the previously supplied table data, it can be concluded that the learned policy does come very close to an optimal policy, particularly if you're willing to incur some penalties. But given that the agent is simulating a smartcab and penalties are illegal traffic actions, it would still be unacceptable for a real world scenario. In which case less emphasis on maximizing rewards while still reaching its destination would be ideal.