Krishnan Ramaswamy
Smartcab project

1. Approach

In this project, I have demonstrated the re-enforcement learning with Q learning agent and discuss the pros and cons and recommend the optimal hyper parameters for Q-learning agent.

Naïve Random Action Agent

Chooses the random action from the pool of actions against the environment.

Learning Agent 1
Uses Q-learning method to learn the true values of each state and perform actions accordingly.

You can control what type of Agent you want to run in the main method of agent.py For example, in the call below, you can change the learrningAgentType to specify the type of agent that you want to experiment.

```
simulate(n_trials=2,
        update_delay=1,
        dummyAgents=1,
        start =(2, 6),
        destination = (6, 3),
        learningAgentType='LearningAgent1',
        alpha=0.9,
        gamma=0.3 )
```

As part of this project. I have developed several automation scripts that enables one to run experiment with several different alpha, gamma and epsilon values in a flexible way that can be initiated from command line. This automaton enable one to initiate an experiment with different trials, hyperparametes without changing the code.

Moreover, I have instrumented the code base (agent.py, environment.py) etc to record run time statistics about the trial and its progress. Up on completion of experiment, an analysis document is created (by analysis the results program) which summarizes the top trials/experiments having high rewards and completion rate. More details about automation, logs etc are described in section : Automation tools, logs and embedded runtime statistics

2. Tasks

2.1. 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?*

In this experiment, I have used random action – the logic picks a random legal action to go ahead.

OUTPUT file = logs/random_action_experiment/smartcab_randomactionagent_100trials.log

I did not use the reward to decide the next best action as I wanted to demonstrate what random action would results in. In most run, this Agent could not reach the destination. Also, it tries to act on the boundaries where there is no grid slot available and the cab moves to non-valid position. What this means is that agent is only exploring and not learning, collecting many negative rewards.

2.2. 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?*

I have identified following variables are suitable to formulate *state* identifier which is used in Q-table.

- traffic light (signal)
- left going traffic
- oncoming traffic
- next_waypoint
- action

Following variables are not suitable for including in *state* identifier

1. Start position & Destination position
    a. both changes each trial run and it is very costly for learning algorithm because of infinite number of states. There will not be any true learning
2. Location cannot used in Q Value as well because no of locations may be more if the grid size is more
3. Right going traffic
    a. According to traffic regulations & law in US, ongoing traffic and right taking car (in the case smartcab) can co-exists which means there is no state information for us to model using this variable
4. Deadline variable is meant to indicate the algorithm to stop/exit if the destination is not found. This variable is increases each time of the trial and always will be changing. Of course, if we were to include this in state definition, then each and every decision will become a unique entry in Q-table and there is no learning will happen and unlimited resources (Q-table) will be consumed.

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?*

No of states in the state definition I have selected is

*= [#of possible traffic light color] * [left, None] * [oncoming, None] * [# possible next_waypoint] * [# possible actions]*

*= 3 * 2 * 2 * 4 * 4*

= 192 states

Yes.. the total (max) number of states is 192 and is reasonable. Because there are only 192 states, Q-table data structure is compact and for each run look up time to get the Q info will be efficient. Also, after several trial, all the state info likely to be having value (learning happened).

2.3. Automation tools, logs and embedded runtime statistics

As part of this project. I have developed several automation scripts that enables one to run experiment with several different alpha, gamma and epsilon values in a flexible way that can be initiated from command line. This automaton enable one to initiate an experiment with different trials, hyperparametes without changing the code.

Moreover, I have instrumented the code base (agent.py, environment.py) etc to record run time statistics about the trial and its progress. Up on completion of experiment, an analysis document is created (by analysis the results program) which summarizes the top trials/experiments having high rewards and completion rate.

Following are the details

2.3.1. Automation

Agent.py

- Enhanced with command line based run. Various hyper-parameters like alpha, gamma, epsilon values can be passed to this program.
- findOptimalParams.py will be calling agant.py for possible combination of alpha, gamma, epsilon

RunTimeStat.py

- Captures various health and run time information about each trial. For example, whether the trial succeeded or not, total rewards, deadline value etc will be captured in this object. After each run, the data gets appended to runTimeStat.json.

- Later, after experiment is completed, analyzeRunTimeStat.py can be executed to print the optimal trial and experiment that is desired.

findOptimalParams.py

- Wrapper/Driver program which executes agent.py for various experimental values.
- Pygames library sometimes hangs and to run several experiments automatically, we need a way to monitor the hanging experiment and restart
- This program monitors the run and kill if it hangs and restart the missed ones. It uses clever techniques to monitor the progress and restart the hanging ones

runOptimalParams.sh

- Shell program to clean up logs, temp files and execute findOptimalParams.py

analyzeRunTimeStat.py

- Reads runTimeStat.json and print the analysis summary – such as which trial is the optimal one

LearningAgentQTable.py

- Q-Table implementation

2.3.2. *Log files*

1) Logs

For each experiment, the corresponding log files are created under logs/experiment{n} directory.
For example, there is a log file in logs/experiment1 with the format

- findOptimalParam_0.2_0.2_0.2.log

using the format findOptimalParam_alpha_gamma_epsilon.log

2) runTimeStat.json

Captures run time stat information or each run. Here is the sample records.

“LearningAgent1, alpha 0.5, gamma 0.5, epsilon 0.9, trial 49": {"rewards": 822.0, "deadline": 0, "reached_dest": "no", "time_taken": ""}, "LearningAgent1, alpha 0.9, gamma 0.9, epsilon 0.2, trial 43": {"rewards": 1344.0, "deadline": 22, "reached_dest": "yes", "time_taken": ""},

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

The formulas for updating Q-values can be found in **this** video.

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?*

I have re-implemented Q-learning chooseAction based on your feedback. Thank you for your guidance and clarifications. Now the algorithm looks for the all values and select the action with maximum value. It also considers epsilon value for making random actions. I have also updated the logic to set Q-value properly.

```
def chooseAction(self, state, nextwaypoint):

    print "\n"
    if random.random() < self._epsilon:
        legalActions = self.getLegalActions()
        bestAction = random.choice(legalActions)
        print " Going for random action :", bestAction
        return bestAction

    else:

        maxQ = 0.0
        bestAction = nextwaypoint
        for action in ['forward', 'left', 'right',None]:
                q =  self.getQValue(state,action=action)
                if q >= maxQ :
                    bestAction = action
                    maxQ = q
                print " Going for best action for the state:", state, "=>",
action,"\t\t",  q, ":", maxQ


        if maxQ == 0.0  :
            legalActions = self.getLegalActions()
            bestAction = random.choice(legalActions)
            print " No prev state-action found. Going for random action for the
state:", bestAction

        else :
            print " +++++++++++++++++++ Self Learned action for the state:",
bestAction, "having Q:", maxQ

        return bestAction


def updateQTable1(self,state,action,reward,newstate):

    oldQvalue = self.getQValue(state,action=action)
    maxValue = self.getMaxQValue(state)
    newQvalue = oldQvalue  + self._alpha * ( (reward + self._gamma * maxValue)
- oldQvalue)
    self.setQValue(state,action=action,new_value=newQvalue)


def getMaxQValue(self, state):

    values = []
    for action in ['forward', 'left', 'right', None]:
        values.append(self.getQValue(state,action=action))

    return max(values)
```

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?

- Initial trials in many cases timeout – meaning deadline approaches 0. What this means is that Agent is not getting sufficient time to complete.
- But as the time goes (more trial), the agent seems to be reaching the destination. What this means is that Q-table is getting updated and learning happens
    - For evidence, please look at the log file. You will see that in later part of the file, the cab reaches the destination quickly. Also, look at the state information and you will see higher Q-table value for each state as more and more trials complete
        - logs/experiment2/findOptimalParam_0.5_0.2_0.2.logIf you look at the log file

- Sometimes, I see the cab getting stuck in local minima. Usually it happens, when epsilon value is very low – such as 0.001. That means, more exploitation is desired.

2.5. 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 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.

Now that I have automation tools, logs and runtime stats and analysis over runtime stats, we will run two experiments.
Each experiments consist of several trials of various combination of alpha, gamma and epsilon.

We will use following parms set and findOptimalParams.py use this value as default. The values for this can be passed in command line as well.

```
alpha_params = [0.2, 0.5, 0.9]
gamma_params = [0.2,0.5,0.9]
epsilon_params = [0.2,0.9]

total trials = 100
```

Now, when you run runOptimalParams.sh, it will automatically run for variaous combination of alpha, gamma and epsilon from the above sets. In this case, total of 18 run (#alpha_params * #gamma_param * #epsilon_par) => 3 * 3 *2 => 18 will be executed.

For each run, the corresponding log file is created under logs/ directory. For example, there is a log file with the format

- findOptimalParam_0.2_0.2_0.2.log using the format findOptimalParam_alpha_gamma_epsilon.log

You will see total 18 logs files corresponding to each run wirh different alpha, gamma and epsilon hyper parms.

2.5.1. Experiment1

All logs related to experiment 1 is under logs/experiment1

Following is the snippet from logs/experiment1/analyseRunTime.log

Please open the file in editor for full details.

```
*************************** Sorted by Highest Reward
*************************

LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 99  ==> Total Reward: 4125.0
LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 98  ==> Total Reward: 4085.0
LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 97  ==> Total Reward: 4033.0
LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 96  ==> Total Reward: 3990.0
LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 95  ==> Total Reward: 3948.5
LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 94  ==> Total Reward: 3911.5
LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 93  ==> Total Reward: 3877.5
LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 92  ==> Total Reward: 3847.5
LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 91  ==> Total Reward: 3808.5
LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 90  ==> Total Reward: 3771.0
LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 89  ==> Total Reward: 3729.0
LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 99  ==> Total Reward: 3662.0
LearningAgent1, alpha 0.2, gamma 0.9, epsilon 0.2, trial 87  ==> Total Reward: 3639.5
```

Observation:

- *High rewards are always associated with later runs such as 99 etc. This proves that agent is learning and getting more rewards as the time goes*
- *Most of the high reward runs are having low epsilon values and high gamma*

```
*************************** Sorted by Maximum Success Experiment
*************************

Experimental Param LearningAgent1 alpha 0.5 gamma 0.9 epsilon 0.2  Success rate: 92
Experimental Param LearningAgent1 alpha 0.5 gamma 0.5 epsilon 0.2  Success rate: 91
Experimental Param LearningAgent1 alpha 0.5 gamma 0.2 epsilon 0.2  Success rate: 89
Experimental Param LearningAgent1 alpha 0.9 gamma 0.5 epsilon 0.2  Success rate: 87
Experimental Param LearningAgent1 alpha 0.9 gamma 0.9 epsilon 0.2  Success rate: 80
Experimental Param LearningAgent1 alpha 0.2 gamma 0.2 epsilon 0.2  Success rate: 80
Experimental Param LearningAgent1 alpha 0.9 gamma 0.2 epsilon 0.2  Success rate: 73
Experimental Param LearningAgent1 alpha 0.2 gamma 0.5 epsilon 0.2  Success rate: 56
Experimental Param LearningAgent1 alpha 0.2 gamma 0.9 epsilon 0.2  Success rate: 42
Experimental Param LearningAgent1 alpha 0.2 gamma 0.9 epsilon 0.9  Success rate: 39
```

Observation:

- *High completion rate is associated with alpha 0.5 or 0.9*
- *This proves that agent with alpha 0.5 is towards optimal value that we are looking for. Further experiments (experiment 2) may support this* **hypothesis**.
- *Most of the high completed runs are having low epsilon values*

2.5.2. Experiment2

All logs related to experiment 1 is under logs/experiment2

Following is the snippet from logs/experiment1/analyseRunTime.log
Please open the file in editor for full details.

```
*************************** Sorted by Highest Reward
*************************

LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 99  ==> Total Reward: 3617.0
LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 98  ==> Total Reward: 3571.5
LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 97  ==> Total Reward: 3542.5
LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 96  ==> Total Reward: 3498.0
LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 95  ==> Total Reward: 3448.5
LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 94  ==> Total Reward: 3414.0
LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 93  ==> Total Reward: 3374.0
LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 92  ==> Total Reward: 3328.0
LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 91  ==> Total Reward: 3303.0
LearningAgent1, alpha 0.9, gamma 0.9, epsilon 0.2, trial 99  ==> Total Reward: 3280.0
LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 90  ==> Total Reward: 3266.5
LearningAgent1, alpha 0.9, gamma 0.9, epsilon 0.2, trial 98  ==> Total Reward: 3253.5
LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 89  ==> Total Reward: 3241.5
```

Observation:

- *High rewards are always associated with later runs such as 99 etc. This proves that agent is learning and getting more rewards as the time goes*
- *Most of the high reward runs are having low epsilon values and medium alpha*

```
**************************  Sorted by Maximum Success Experiment
************************

Experimental Param LearningAgent1 alpha 0.5 gamma 0.2 epsilon 0.2  Success rate: 96
Experimental Param LearningAgent1 alpha 0.9 gamma 0.2 epsilon 0.2  Success rate: 90
Experimental Param LearningAgent1 alpha 0.9 gamma 0.9 epsilon 0.2  Success rate: 87
Experimental Param LearningAgent1 alpha 0.5 gamma 0.5 epsilon 0.2  Success rate: 83
Experimental Param LearningAgent1 alpha 0.5 gamma 0.9 epsilon 0.2  Success rate: 80
Experimental Param LearningAgent1 alpha 0.9 gamma 0.5 epsilon 0.2  Success rate: 79
Experimental Param LearningAgent1 alpha 0.2 gamma 0.2 epsilon 0.2  Success rate: 66
Experimental Param LearningAgent1 alpha 0.2 gamma 0.5 epsilon 0.2  Success rate: 60
Experimental Param LearningAgent1 alpha 0.2 gamma 0.9 epsilon 0.2  Success rate: 33
```

Observation:

- *High completion rate is associated with alpha 0.5 or alpha 0.9 AND gamma 0.2*
- *This proves that agent with alpha 0.5 is towards optimal value*
- *Most of the high completed runs are having low epsilon values*

2.5.3. Summary

By looking at experment1 and experiment 2 logs results, we can realize that

Agent with alpha as 0.5 or 0.9 , gamma as 0.2 and epsilon as 0.2 runs with high reward and best completin rate with minimal penality and best run time.

Higher alpha means that agent uses higer learning rate. Learning rate controls the learning step size, that is, how fast learning takes place. In our experiment, higher alpha does not always result in best completion runs. Seems like 0.5 alpha is optimal. You can review the detail logs to confirm this.

Future reinforcements are weights controlled by a value gamma between 0 and 1. A higher value of gamma means that the future matters more for the Q-value of a given action in a given state.

In our experiment, higher gamma does not help much. Gamma with 0.2 value provides the best results. The reason for higher gamma is not helpful because the information just received may be faulty for one reason or another. It is better to update more gradually, to use the new information to move in a particular direction, but not to make too strong a commitment.

Higher epsilon is counterproductive in our experiment. Usually higher epsilon helps when three is more randomness in the behavior of env response. Seems like the smartcab env, there is not much of variation of responses from env and higher epsilon does not help much – hence low epsilon results in better success rate.

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?

Agent with alpha as 0.5 or 0.9 , gamma as 0.2 ad epsilon as 0.2 runs with high reward and best completion rate with minimal penalty and best run time.

The deadline value (look in analyseRunTime.log) line no 702 in (logs/experiment2/ *analyseRunTime.log)*

```
700     : LearningAgent1, alpha 0.5, gamma 0.2, epsilon 0.2, trial 99 ==>  {u'rewards': 3617.0, u'deadline': 29, u'reached_dest': u'yes', u'time_taken'     : u''}
```

When a deadline value high, it means that cab reached the destination quickly.

The optimal policy for this kind of agent is to balance between exploration and exploitation.
Alpha with 0.5 and gamma 0.2 achieves the optimal exploitation and 0.2 epsilon is optimal for exploration.