UDACITY

Logo

PROJECT

Train a Smartcab to Drive

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW | CODE REVIEW 2 | NOTES

agent.py 2

```
import random
import math
from environment import Agent, Environment
from planner import RoutePlanner
from simulator import Simulator

class LearningAgent(Agent):
    """ An agent that learns to drive in the Smartcab world.
        This is the object you will be modifying. """

    def __init__(self, env, learning=False, epsilon=1.0, alpha=0.5):
        super(LearningAgent, self).__init__(env)     # Set the agent in the evi
        self.planner = RoutePlanner(self.env, self)  # Create a route planner
        self.valid_actions = self.env.valid_actions  # The set of valid actions

        # Set parameters of the learning agent
        self.learning = learning # Whether the agent is expected to learn
        self.Q = dict()          # Create a Q-table which will be a dictionary
        self.epsilon = epsilon   # Random exploration factor
        self.alpha = alpha       # Learning factor

        ############
        ## TO DO ##
        ############
        # Set any additional class parameters as needed
        self.counter = 0
        random.seed(161) #https://stackoverflow.com/questions/22639587/random-s

    def reset(self, destination=None, testing=False):
        """ The reset function is called at the beginning of each trial.
            'testing' is set to True if testing trials are being used
            once training trials have completed. """

        # Select the destination as the new location to route to
        self.planner.route_to(destination)

        ############
        ## TO DO ##
        ############
        # Update epsilon using a decay function of your choice
        # Update additional class parameters as needed
        # If 'testing' is True, set epsilon and alpha to 0
        self.counter=self.counter+1
        if testing:
            self.epsilon = 0
            self.alpha = 0
        else:
            #Default Q Learning Parameter
            #self.epsilon=self.epsilon-0.05
            #Improved Q Learning agent
            #self.epsilon=0.9**self.counter
            #self.epsilon=1/(self.counter**3)
            #self.epsilon=math.exp(-0.01*self.counter)
            self.epsilon=math.cos(-0.001*self.counter)

        return None

    def build_state(self):
        """ The build_state function is called when the agent requests data fr
            environment. The next waypoint, the intersection inputs, and the de
            are all features available to the agent. """

        # Collect data about the environment
```

```
64          waypoint = self.planner.next_waypoint() # The next waypoint
65          inputs = self.env.sense(self)           # Visual input - intersection l
66          deadline = self.env.get_deadline(self)  # Remaining deadline
67
68          ###########
69          ## TO DO ##
70          ###########
71
72          # NOTE : you are not allowed to engineer eatures outside of the inputs
73          # Because the aim of this project is to teach Reinforcement Learning, w
74          # constraints in order for you to learn how to adjust epsilon and alpha
75          # With the hand-engineered features, this learning process gets entirel
76          state=None
77          # Set 'state' as a tuple of relevant data for the agent
78          state = waypoint, inputs['light'], inputs['left'], inputs['right'], inp
79
80          return state
81
82
83      def get_maxQ(self, state):
84          """ The get_max_Q function is called when the agent is asked to find th
85              maximum Q-value of all actions based on the 'state' the smartcab is
86
87          ###########
88          ## TO DO ##
89          ###########
90          # Calculate the maximum Q-value of all actions for a given state
91          maxQ=None
92          maxQ = self.Q[state][max(self.Q[state], key=lambda key: self.Q[state][k
93
94          return maxQ
95
96
97      def createQ(self, state):
98          """ The createQ function is called when a state is generated by the age
99
100         ###########
101         ## TO DO ##
102         ###########
103         # When learning, check if the 'state' is not in the Q-table
104         # If it is not, create a new dictionary for that state
105         #   Then, for each action available, set the initial Q-value to 0.0
106         if self.learning:
107             if state not in self.Q:
108                 self.Q[state]={None:0.0, 'forward':0.0, 'left':0.0, 'right':0.0
109         return
110
111
112     def choose_action(self, state):
113         """ The choose_action function is called when the agent is asked to cho
114             which action to take, based on the 'state' the smartcab is in. """
115
116         # Set the agent state and default action
117         self.state = state
118         self.next_waypoint = self.planner.next_waypoint()
119         action = None
120
121         ###########
122         ## TO DO ##
123         ###########
124         # When not learning, choose a random action
125         # When learning, choose a random action with 'epsilon' probability
126         # Otherwise, choose an action with the highest Q-value for the current
127         # Be sure that when choosing an action with highest Q-value that you ra
128         if not self.learning:
129             action = random.choice(self.valid_actions)
130         else:
131             if self.epsilon >= random.random():
132                 action = random.choice(self.valid_actions)
133             else:
134                 # Following piece of code implemented after 1st reviewer commen
135                 max_random_valid_actions = []
136                 max_Q_Value = self.Q[state][max(self.Q[state], key=lambda key:
137                 for act in self.Q[state]:
138                     if max_Q_Value == self.Q[state][act]:
139                         max_random_valid_actions.append(act)
140                 action = random.choice(max_random_valid_actions)
```

AWESOME

Excellent implementation of a tie-breaker function between best actions! For this part of the code, you car

```
action = random.choice([k for (k, v) in Q_state.items() if v == max_Q_Value])
```

```
141                 # Following piece of code implemented after 1st reviewer commen
142                 #action = max(self.Q[state], key=lambda key: self.Q[state][key]
143         return action
144
145
```

```
146     def learn(self, state, action, reward):
147         """ The learn function is called after the agent completes an action ar
148             receives a reward. This function does not consider future rewards
149             when conducting learning. """
150
151         ###########
152         ## TO DO ##
153         ###########
154         # When learning, implement the value iteration update rule
155         #   Use only the learning rate 'alpha' (do not use the discount factor
156         if self.learning:
157             self.Q[state][action] = self.Q[state][action] + self.alpha*(reward-
```

AWESOME

Great job, your Bellman equation uses only current rewards.

```
158         return
159
160
161     def update(self):
162         """ The update function is called when a time step is completed in the
163             environment for a given trial. This function will build the agent
164             state, choose an action, receive a reward, and learn if enabled. "'
165
166         state = self.build_state()          # Get current state
167         self.createQ(state)                 # Create 'state' in Q-table
168         action = self.choose_action(state)  # Choose an action
169         reward = self.env.act(self, action) # Receive a reward
170         self.learn(state, action, reward)   # Q-learn
171
172         return
173
174
175 def run():
176     """ Driving function for running the simulation.
177         Press ESC to close the simulation, or [SPACE] to pause the simulation.
178
179     ##############
180     # Create the environment
181     # Flags:
182     #   verbose     - set to True to display additional output from the simulat
183     #   num_dummies - discrete number of dummy agents in the environment, defau
184     #   grid_size   - discrete number of intersections (columns, rows), default
185     env = Environment(verbose=True)
186
187     ##############
188     # Create the driving agent
189     # Flags:
190     #   learning   - set to True to force the driving agent to use Q-learning
191     #    * epsilon - continuous value for the exploration factor, default is 1
192     #    * alpha   - continuous value for the learning rate, default is 0.5
193     agent = env.create_agent(LearningAgent, learning=True, alpha=0.95, epsilon=
194
195     ##############
196     # Follow the driving agent
197     # Flags:
198     #   enforce_deadline - set to True to enforce a deadline metric
199     env.set_primary_agent(agent, enforce_deadline=True)
200
201     ##############
202     # Create the simulation
203     # Flags:
204     #   update_delay - continuous time (in seconds) between actions, default is
205     #   display      - set to False to disable the GUI if PyGame is enabled
206     #   log_metrics  - set to True to log trial and simulation results to /logs
207     #   optimized    - set to True to change the default log file name
208     sim = Simulator(env, update_delay=0.01, log_metrics=True, display=False, op
209
210     ##############
211     # Run the simulator
212     # Flags:
213     #   tolerance  - epsilon tolerance before beginning testing, default is 0.(
214     #   n_test     - discrete number of testing trials to perform, default is (
215     sim.run(n_test=40, tolerance=0.0001)
216
217
218 if __name__ == '__main__':
219     run()
```

▸ logs/sim_improved-learning.txt

▸ logs/sim_default-learning.txt

RETURN TO PATH

Student FAQ