Sarsa-Copy1

May 15, 2019

1 First, we will import the dependencies and examine the mountain car environment, using the following code:

```
In [1]: import gym
        import numpy as np

        #exploring Mountain Car environment

        env_name = 'MountainCar-v0'
        env = gym.make(env_name)

        print("Action Set size :",env.action_space)
        print("Observation set shape :",env.observation_space)
        print("Highest state feature value :",env.observation_space.high)
        print("Lowest state feature value:",env.observation_space.low)
        print(env.observation_space.shape)
```

```
Action Set size : Discrete(3)
Observation set shape : Box(2,)
Highest state feature value : [0.6  0.07]
Lowest state feature value: [-1.2  -0.07]
(2,)
```

```
In [2]: #import gym
        #import numpy as np
        #env = gym.make("Moun tai").env

        #env.render()
```

2 hyperparameters

```
In [2]: n_states = 40   # number of states
        episodes = 3 # number of episodes

        initial_lr = 1.0 # initial learning rate
```

```
min_lr = 0.005 # minimum learning rate
gamma = 0.99 # discount factor
max_steps = 300
epsilon = 0.05

env = env.unwrapped
env.seed(0)
np.random.seed(0)
```

function to perform discretization of the continuous state space. Discretization is the conve

In [3]:
```
def discretization(env, obs):

    env_low = env.observation_space.low
    env_high = env.observation_space.high

    env_den = (env_high - env_low) / n_states
    pos_den = env_den[0]
    vel_den = env_den[1]

    pos_high = env_high[0]
    pos_low = env_low[0]
    vel_high = env_high[1]
    vel_low = env_low[1]

    pos_scaled = int((obs[0] - pos_low)/pos_den)  #converts to an integer value
    vel_scaled = int((obs[1] - vel_low)/vel_den)  #converts to an integer value

    return pos_scaled,vel_scaled
```

Now the SARSA implementation starts with initializing the Q-table and updating the Q-values a

In [ ]:
```
q_table = np.zeros((n_states,n_states,env.action_space.n))
total_steps = 0
for episode in range(episodes):
    obs = env.reset()
    total_reward = 0
    # decreasing learning rate alpha over time
    alpha = max(min_lr,initial_lr*(gamma**(episode//100)))
    steps = 0
    #action for the initial state using epsilon greedy
    if np.random.uniform(low=0,high=1) < epsilon:
        a = np.random.choice(env.action_space.n)
    else:
        pos,vel = discretization(env,obs)
        a = np.argmax(q_table[pos][vel])
    while True:
        env.render()
        pos,vel = discretization(env,obs)
```

```
            obs,reward,terminate,_ = env.step(a)
            total_reward += abs(obs[0]+0.5)
            pos_,vel_ = discretization(env,obs)
            #action for the next state using epsilon greedy
            if np.random.uniform(low=0,high=1) < epsilon:
                a_ = np.random.choice(env.action_space.n)
            else:
                a_ = np.argmax(q_table[pos_][vel_])
                #q-table update
                q_table[pos][vel][a] = (1-alpha)*q_table[pos][vel][a] + alpha*(reward+gamma
                steps+=1
            if terminate:
                break
                a = a_
        print("Episode {} completed with total reward {} in {} steps".format(episode+1,tota
    while True: #to hold the render at the last step when Car passes the flag
        env.render()
```

In [ ]: