Lab02_problem

December 20, 2021

1 Lab 02

In this lab, you're going to:

1. Implement Q agent and SARSA agent
2. Study about on-policy, off-policy and how they trade-off convergence speed for sample-efficiency
3. Create simple plot to visualize your agent's results

NOTICE 1: This lab calls to the ***MABe_agent*** class of previous lab. **IF** you have not finish the previous lab, you should not be aable to complete this lab

NOTICE 2: Convert your previous lab's solution or instructor's solution to format **.py** then place that file in this file's directory. We're going to inherit some classes.

NOTICE 3: This lab's solution **will not** be given due to the inheritance of the given lab 01's solution. You can easily implement Q-learning and SARSA by changing the estimation fomula.

Also this lab, we're going to reuse last week's environment: **GridWorld**. The different is our agent. If you don't remember what is GridWorld, please revise your last lab excercise in advance

[ ]:
```
from lab01 import environment, MABe_agent
```

Estimation updates: >Q-learning: $Q(S_t, A_t) = (1-\alpha)Q(S_t, A_t) + \alpha(R + \gamma max_a Q(S_{t+1}, a))$

SARSA: $Q(S_t, A_t) = (1-\alpha)Q(S_t, A_t) + \alpha(R + \gamma Q(S_{t+1}, A_{t+1}))$

On-policy and Off-policy

---

1. On-policy:

On-policy is a group of reinforcement learning methods that **use the value resulting from a different policy to update the current policy**

2. Off-policy:

Off-policy is a group of reinforcement learning methods that **use the value resulting directly from the current policy to update itself**

3. On-policy, off-policy, cons & pros:

- Off-policy methods are usually learning more about the environment they can use multiple policies' experience. Thus, they're more data-sampling efficient. Famous: *Q-learning, DQN* *and its extensions*

- On-policy methods are usually more realistic (most of the time, we can only depend on our own policy for the data we can collect, especially in policy-based methods) and fast converging but not as sampling-efficient as off-policy methods

1.1 Q-learning method

In this section, we're going to implement tabular Q-learning agent and see its performance.

Q-agent inherits from MABe-agent we implemented earlier in this course

[ ]:
```
# Implement your agent
class Q_agent(MABe_agent):
  def __init__(envir, init_location, epsilon):
    super(Q_agent, self).__init__(envir, init_location, epsilon)
    # TODO: initialize your Q table
    self.Q_table = None

  # Overide method
  def getAction(self, observation):
    # TODO: return your action
    ocation_now, action_space, pre_reward = observation
    # NOTICE: the first observation is (NONE, [0,1,2,3], None)
    # You should process the 'None' value
    if location_now is not None:
      self.location_now = location_now

    if pre_reward is not None:
      self.reward_trace.append(pre_reward)

    # example: get random action
    action = np.random.choice(action_space, p=[1/(len(action_space)) for action␣
 ↪in action_space])

    # Assert valid action
    assert action in action_space, "INVALID action taken"
    return action
```

[ ]:
```
# Create environment
Envir = environment(8,8)
Envir.map_Designate(17,56,-15)
Envir.map_Designate(18,56,-15)
Envir.map_Designate(19,56,-15)
Envir.map_Designate(21,56,-15)
Envir.map_Designate(25,56,-15)
Envir.map_Designate(33,56,-15)
Envir.map_Designate(41,56,-15)
Envir.map_Designate(42,56,-15)
```

```
Envir.map_Designate(43,56,-15)
Envir.map_Designate(46,56,-15)
Envir.map_Designate(47,56,-15)
Envir.map_Designate(47,56,-15)
Envir.map_Designate(15,56,+15)
Envir.map_Designate(1,10,+5)
Envir.map_Designate(26,56,+20)
```

[ ]:
```
init_location=0
dummy_q_agent = Q_agent(envir=Envir, init_location=init_location)

num_iter = 1000

log_freq = 100
Data_plot1 = []

for i in range(num_iter):
  env_observation = (init_location, Envir.action_space, None)
  if i > 0:
    env_observation = Envir.get_Observation(location=dummyAgent.location_now,␣
↪action=chosen_action)

  chosen_action = dummyAgent.getAction(observation=env_observation)
  if (i + 1) % log_freq == 0:
    aver = np.mean(dummyAgent.reward_trace)
    Data_plot1.append(aver)
    print('iter: ' + str(i + 1) + '\t Total reward: ' + str(dummyAgent.
↪get_TotalReward()) + '\t Average: ' + str(aver))
```

1.2 SARSA

SARSA is the on-policy version of Q-learning The different is observable through its updating rule

$$Q(S_t, A_t) = (1 - \alpha)Q(S_t, A_t) + \alpha(R + \gamma Q(S_{t+1}, A_{t+1}))$$

Meaning: the sequence of observation is $> S_t \to A_t \to R \to S_{t+1} \to A_{t+1}$

HENCE: **S-A-R-S-A**

[ ]:
```
# Implement your agent
class SARSA_agent(MABe_agent):
  def __init__(envir, init_location, epsilon):
    super(Q_agent, self).__init__(envir, init_location, epsilon)
    # TODO: initialize your Q table
    self.Q_table = None

  # Overide method
  def getAction(self, observation):
    # TODO: return your action
```

```
    ocation_now, action_space, pre_reward = observation
    # NOTICE: the first observation is (NONE, [0,1,2,3], None)
    # You should process the 'None' value
    if location_now is not None:
      self.location_now = location_now

    if pre_reward is not None:
      self.reward_trace.append(pre_reward)

    # example: get random action
    action = np.random.choice(action_space, p=[1/(len(action_space)) for action␣
 ↪in action_space])

    # Assert valid action
    assert action in action_space, "INVALID action taken"
    return action
```

[ ]:
```
init_location=0
dummy_sarsa_agent = SARSA_agent(envir=Envir, init_location=init_location)

num_iter = 1000

log_freq = 100
Data_plot2 = []

for i in range(num_iter):
  env_observation = (init_location, Envir.action_space, None)
  if i > 0:
    env_observation = Envir.get_Observation(location=dummyAgent.location_now,␣
 ↪action=chosen_action)

  chosen_action = dummyAgent.getAction(observation=env_observation)
  if (i + 1) % log_freq == 0:
    aver = np.mean(dummyAgent.reward_trace)
    Data_plot2.append(aver)
    print('iter: ' + str(i + 1) + '\t Total reward: ' + str(dummyAgent.
 ↪get_TotalReward()) + '\t Average: ' + str(aver))
```

1.3 Plot your results

Compare Q-learning's and SARSA's performance

[ ]:
```
import matplotlib.pyplot as plt
```

[ ]:
```
# TODO: visualize your agent's performance
```