07 - Reinforcement Learning

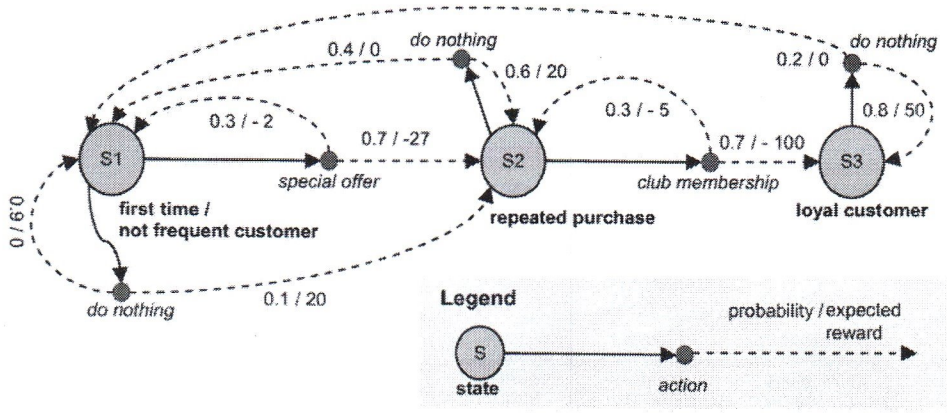
In the last exercise session, we have seen ways to solve the prediction and control problems for a sequential decision-making task modeled with a known MDP.

In this exercise session, we consider a similar setting except that the underlying MDP is now unknwon. Can we still solve the prediction and control problems from sampled interactions with the environemnt?

Environment and Transition Model

In reinforcement learning, the full MDP model is not available to the agent. Instead, we define an environment class that only exposes the *transition model* of the MDP in a black-box method. This method takes as input the action selected by the agent and it returns the next state observation along with the immediate reward.

Let us define the environment class for the same *Advertising Problem* that we considered in the last exercise session.



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

class Environment(object):

  def __init__(self):
    # states and actions
    self.nS = 3
    self.nA = 3
    self.allowed_actions = np.array([[1, 1, 0], [1, 0, 1], [1, 0, 0]])
    # initial state distribution and discount factor
    self.mu = np.array([1., 0., 0.])
    self.gamma = 0.9
    # transition model (SA rows, S columns)
    self.P = np.array([[0.9, 0.1, 0.],
                       [0.3, 0.7, 0.],
                       [0., 0., 0.],
                       [0.4, 0.6, 0.],
                       [0., 0., 0.],
                       [0., 0.3, 0.7],
                       [0.2, 0, 0.8],
                       [0., 0., 0.],
                       [0., 0., 0.]])
    # immediate reward (SA rows, S columns)
    self.R = np.array([[0, 20, 0],
                       [-2, -27, 0],
                       [0., 0., 0.],
                       [0, 20, 0],
                       [0., 0., 0.],
                       [0, -5, -100],
                       [0, 0, 50],
                       [0., 0., 0.],
                       [0., 0., 0.]])

  # method to set the environment random seed
  def _seed(self, seed):
    np.random.seed(seed)

  # method to reset the environment to the initial state
  def reset(self):
    self.s = s = np.random.choice(self.nS, p=self.mu)
    return s

  # method to perform an environemnt transition
  def transition_model(self, a):
    sa = self.s * self.nA + a
    self.s = s_prime = np.random.choice(self.nS, p=self.P[sa, :])
    inst_rew = self.R[sa, s_prime]
    return s_prime, inst_rew
```

it gets an action and it returns the next state and the reward

Having defined the class, we can instantiate an environment object and interact with it. For example, with a fixed policy that always selects the action *do-nothing*, we get:

In [2]:
```
env = Environment()
# env._seed(0) # set a seed to have reproducible results
```

```
T = 10 # interaction steps
a = 0 # policy with a fixed action

ret = 0 # return initialization
s = env.reset()
print('step', 0, 's:', s)
for i in range(T):
  s, r = env.transition_model(a)
  ret = ret + r * (env.gamma ** (i+1))
  print('\nstep:', i+1, 's:', s, 'a:', a, 'r:', r)
print('\nthe final return is:', ret)
```

```
step 0 s: 0

step: 1 s: 0 a: 0 r: 0.0

step: 2 s: 0 a: 0 r: 0.0

step: 3 s: 0 a: 0 r: 0.0

step: 4 s: 1 a: 0 r: 20.0

step: 5 s: 1 a: 0 r: 20.0

step: 6 s: 1 a: 0 r: 20.0

step: 7 s: 1 a: 0 r: 20.0

step: 8 s: 1 a: 0 r: 20.0

step: 9 s: 0 a: 0 r: 0.0

step: 10 s: 0 a: 0 r: 0.0

the final return is: 53.735902200000005
```

Reinforcement Learning for Prediction

In the last exercise session, we have seen methods (i.e., the exact solution and the recursive formulation of the Bellman expectation equation) to evaluate the performance V^π of a fixed policy π. Without the knwoledge of the MDP, we can evaluate a given policy π by sampling a batch of interactions from the environment and performing a Temporal Difference (TD) estimation:

$$V(s) \leftarrow V(s) + \alpha\big(r + \gamma V(s') - V(s)\big).$$

First, we define the *far-sighted* policy we aim to evaluate:

In [3]:
```
# far-sighted policy
pi_far = np.array([[0., 1., 0.],
                   [0., 0., 1.],
                   [1., 0., 0.]])
```

Then, we sample interactions with an instance of the Environment class and we update the value function estimate:

In [4]:
```
import numpy as np

# instantiate the environment
env = Environment()
# env._seed(9)
# learning parameters
M = 5000
m = 1
# initial V function
V = np.zeros(env.nS)
actions = [0, 1, 2]

# initial state and action
s = env.reset()
# TD evaluation
while m < M:
  alpha = (1 - m/M)
  # environment step
  a = np.random.choice(actions, p=pi_far[s])
  s_prime, r = env.transition_model(a)
  # TD update
  V[s] = V[s] + alpha * (r + env.gamma * V[s_prime] - V[s])
  # next iteration
  m = m + 1
  s = s_prime

print('The final V function is:\n', V)
```

```
The final V function is:
 [-17.06444793  20.14955817 134.91138504]
```

Since the TD estimator is consistent, V would converge to the actual V^π with a sufficiently large number of samples.

Reinforcement Learning for Control

In the last exercise session, we have seen methods (i.e., brute force, policy iteration, value iteration) to compute an optimal policy $\pi^* \in \arg\max_{\pi \in \Pi} V^\pi(s), \forall s \in \mathcal{S}$ for a given MDP. Without the knowledge of the MDP, can we still find an (approximately) optimal policy from mere interactions with the environment?

Reinforcement learning gives a positive answer to this crucial question. Especially, we are going to implement two remarkable methods:

1. *SARSA*, which combines TD evaluation with ϵ-greedy policy improvemnt. It is in a way an empirical policy iteration procedure;
2. *Q-learning*, which embeds TD estimation into a typical value iteration update.

SARSA

randomized algorithm, without a seed we may get different results

To implement SARSA, we need three elements:

- a dataset or a generative process (which we can get from the environment class defined above);
- a policy improvement step (ϵ-greedy policy);
- an evaluation step (TD estimate update).

First, we define the ϵ-greedy policy as a function that takes as input the current state s and the current Q function estimate and returns the selected action:

In [5]:
```
def eps_greedy(s, Q, eps, allowed_actions):
  if np.random.rand() <= eps:
    actions = np.where(allowed_actions)
    actions = actions[0] # just keep the indices of the allowed actions
    a = np.random.choice(actions, p=(np.ones(len(actions)) / len(actions)))
  else:
    Q_s = Q[s, :].copy()
    Q_s[allowed_actions == 0] = - np.inf
    a = np.argmax(Q_s)
  return a
```

with probability ϵ we take a random choice

with probability $1-\epsilon$ we take the best action given the current estimate of the Q function (optimal action = the one which maximize the Q function)

Then, we define the TD update as before, but we now consider an estimate of the Q function instead of V, which is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big),$$

and we combine the three components in the *SARSA* algorithm:

In [6]:
```
import numpy as np

# instantiate the environment
env = Environment()
env._seed(10)
# learning parameters
M = 5000
m = 1
# initial Q function
Q = np.zeros((env.nS, env.nA))

# initial state and action
s = env.reset()
a = eps_greedy(s, Q, 1., env.allowed_actions[s])
# SARSA main loop
while m < M:
  alpha = (1 - m/M)
  eps = (1 - m/M) ** 2
  # environment step
  s_prime, r = env.transition_model(a)
  # policy improvement step
  a_prime = eps_greedy(s, Q, eps, env.allowed_actions[s_prime])
  # SARSA update (evaluation step)
  Q[s, a] = Q[s, a] + alpha * (r + env.gamma * Q[s_prime, a_prime] - Q[s, a])
  # next iteration
  m = m + 1
  s = s_prime
  a = a_prime

print('The final Q function is:\n', Q)
```

```
The final Q function is:
 [[40.22746377  8.5816356   0.        ]
 [67.39322817  0.          6.08671501]
 [79.70051083  0.          0.        ]]
```

Q-learning

To implement Q-learning, we just need to take the SARSA algorithm above and to substitute the SARSA update with the Q-learning update, which is a TD approximation of the Bellman optimality equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma \max_{\tilde{a}\in\mathcal{A}} Q(s',\tilde{a}) - Q(s,a)\big).$$

In [7]:
```
import numpy as np

# instantiate the environment
env = Environment()
env._seed(10)
# learning parameters
M = 5000
m = 1
# initial Q function
Q = np.zeros((env.nS, env.nA))

# initial state and action
```

```
s = env.reset()
a = eps_greedy(s, Q, 1., env.allowed_actions[s])
# Q-learning main loop
while m < M:
  alpha = (1 - m/M)
  eps = (1 - m/M) ** 2
  # environment step
  s_prime, r = env.transition_model(a)
  # policy improvement step
  a_prime = eps_greedy(s, Q, eps, env.allowed_actions[s_prime])
  # Q-learning update
  Q[s, a] = Q[s, a] + alpha * (r + env.gamma * np.max(Q[s_prime, :]) - Q[s, a])
  # next iteration
  m = m + 1
  s = s_prime
  a = a_prime

print('The final Q function is:\n', Q)
```

```
The final Q function is:
 [[ 41.15690117  25.13753528   0.        ]
 [ 68.68305032   0.          28.26528974]
 [127.83597797   0.           0.        ]]
```

Homeworks

Here we propose some additional exercises in Python for you. They are not mandatory, but they can be helpful to better understand the contents of the lecture.

Monte-Carlo Prediction and Control

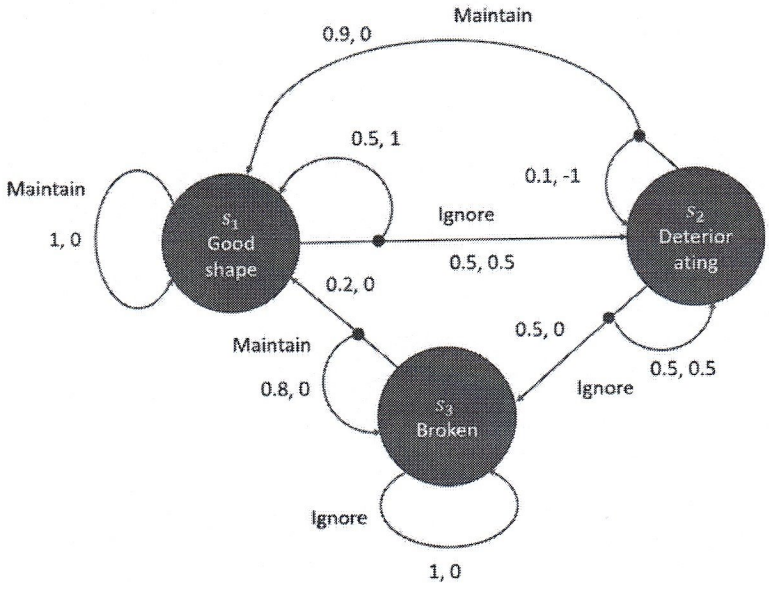
Try to solve the same prediction and control problems we have considered with Monte-Carlo value functions estimation instead of TD.

Off-Policy Monte-Carlo Prediction

Try to evaluate the *far-sighted* policy by sampling interactions with a different policy (hint, try with a uniform policy). How do you have to modify the Monte-Carlo update to get an unbiased estimate?

Environment Class Definition

Try to define the environment class for the MDP depicted below. Then, use the code we implemented above to (approximately) solve a prediction and control problem from samples interactions with this new environment.



8 Reinforcement Learning

8.1 Questions

Exercise 8.1

Tell if the following statements are true or false and provide the adequate motivations to your answer.

1. In RL we do not require to have the model of the environment;
2. In RL we do not represent the model of the environment;
3. We need to use data coming from the optimal policy if we want to learn it;
4. Since RL sequentially decide the action to play at each time point, we cannot use information provided by historical data;
5. We can manage continuous space with RL.

X Exercise 8.2

Tell if the following properties hold for MC or TD and motivate your answers.

1. Can be applied to infinite horizon ML;
2. Can be applied to indefinite horizon ML; (there is an horizon, we don't know where) we can't say "100 steps" but we know that at some point the process will end
3. Needs an entire episode;
4. Works step by step (online);
5. Applies bootstrap; — ~ update a guess over a guess (this is what we do in TD, in MC we don't guess anything)
6. Learning complexity depends on the dimension of the MDP; = the number of samples that we get depends on the dimension of the MDP
7. Learning complexity depends on the length of the episodes;
8. Solves the prediction problem;

9. Reuse the information learned from past learning steps;
10. Makes use of the Markov property of the MDP;
11. Has no bias;
12. Has some bias.

Exercise 8.3

Tell if the following statements are true or false and motivate your answers.

1. With MC estimation you can extract a number of samples for the value function equal to the length of the episode you consider for prediction;
2. Generally, every-visit estimation is better if you use a small amount of episodes;
3. Stochasticity in the rewards requires the use of a larger number of episode to have precise prediction of the MDP value in the case we use MC estimation;
4. MC estimation works better than TD if the problem is not Markovian.

Exercise 8.4

Tell if the following statements are true or false and motivate your answers.

1. To compute the value of a state TD uses an approach similar to the one used in the Policy Evaluation algorithm;
2. TD updates its prediction as soon as a new tuple (state, action, reward, next state) is available;
3. TD cannot be used in the case there is no terminal state in the original MDP;
4. Since with TD we use values computed by averaging, we introduce less variance in the estimation than MC.

Exercise 8.5

Evaluate the value for the MDP with three states $\mathcal{S} = \{A, B, C\}$ (C is terminal), two actions $\mathcal{A} = \{h, r\}$ given the policy π, given the following trajectories:

$$(A, h, 3) \to (B, r, 2) \to (B, h, 1) \to (C)$$
$$(A, h, 2) \to (A, h, 1) \to (C)$$
$$(B, r, 1) \to (A, h, 1) \to (C)$$

1. Can you tell without computing anything if by resorting to MC with every-visit and first-visit approach you will have different results?
2. Compute the values with the two aforementioned methods.
3. Assume to consider a discount factor $\gamma = 1$. Compute the values by resorting to TD? Assume to start from zero values for each state and $\alpha = 0.1$.

Exercise 8.6

Comment on the use of α in the stochastic approximation problem to estimate an average value:

$$\mu_i = (1 - \alpha_i)\mu_{i-1} + \alpha_i x_i$$

Is $\alpha_i = \frac{1}{i}$ a valid choice? Is $\alpha = \frac{1}{i^2}$ meaningful?

Exercise 8.7

Consider the following problems and tell when the optimal policy can be found by resorting to RL or DP techniques:

1. Maze Escape
2. Pole balancing problem
3. Ads displacement
4. Chess

Exercise 8.8

Tell if the following statements are true or false.

1. To converge to the optimal policy we can even use MC estimation and a greedy policy;
2. To ensure convergence we should ensure that all the states are visited during the learning process;

3. It is not possible to learn the optimal policy by running a different policy on an MDP;

4. Information gathered from previous experience can not be included in the RL learning process.

Provide adequate motivations for your answers.

Solutions

Answer of exercise 8.1

1. TRUE This is the difference from dynamic programming approaches;
2. FALSE In model based RL we do not have a known model of the environment, but we built an approximation of the model of the environment by basing on the data. This statement is true for model free RL approaches, which does not learn an MDP explicitly;
3. FALSE It is possible to learn also from the non optimal policies (off-policy RL);
4. FALSE For instance, in the case of model-free prediction we might use data coming from previously executed policies and process them as a single batch;
5. TRUE We might resort to function approximation if the state space is continuous or if the complexity of the problem does not allow us to solve the problem.

Answer of exercise 8.2

1. TD since it might operate even without having complete episodes;
2. MC since is able to use episodes with different length and TD since it might be updated after each transition;
3. MC (and TD(∞)) since it needs the overall reward at the end of the episode;
4. TD since it updates the value function after each instantaneous reward;
5. TD since it makes computes the estimates by using the information learned so far. This also applies to MC in its incremental version with $\alpha \neq \frac{1}{n}$, n being the number of states processed so far;
6. MC (first-visit) since you will get only one sample per state in each episode (assuming to have fewer states than transitions in an episode);
7. TD and MC (every-visit) since the more the episodes are long the more the sample we have;
8. MC and TD since both returns the value for each state given the observed policy;
9. TD since it uses a bootstrap strategy (and MC with $\alpha \neq \frac{1}{n}$);
10. TD since it explicitly uses it to compute the state value function;

11. MC (first-visit) since the computed value is an unbiased estimator of the state value function;

12. TD and MC (every-visit) since they make use of biased state values and dependent samples, respectively.

Answer of exercise 8.3

1. TRUE/FALSE With every-visit we have a sample per states in the episode, while if we consider first-visit we will count the occurrence of a state only once, thus, we have at most a number of samples equal to the number of states;

2. TRUE Even if it is a biased estimator for the expected value of a state, by resorting to every-visit MC we are gathering more samples, thus, we reduce the variance, which in the case we have only few samples is crucial;

3. TRUE The presence of stochasticity increases the variance of the estimates, thus, the estimated values are more and more uncertain;

4. TRUE MC does not use any information about the transition model of the MDP, thus no assumption on the transition model is considered when using MC. Instead, TD explicitly makes use of the Markovian property of MDPs.

Answer of exercise 8.4

1. TRUE The estimated return in the TD equation is nothing else but the right hand side of the Bellman expectation equation in the case we have a single action and we know the transition we perform;

2. TRUE It requires only a single transition to update the value of a state, while for instance MC needs to have a complete episode before updating the estimation of the values of the MDP;

3. FALSE Since its update are performed at each transition, we do not require to have finite episodes. This is different from MC for which we need to wait for the end of the episode;

4. TRUE By considering TD we are introducing some bias (in the estimates of the state values), but the fact that they are computed by averaging more transitions (i.e., we are considering multiple state, action, reward tuples) decreases the variance of the estimates. Moreover, we know that the estimate is consistent, thus, if we consider an infinite number of transitions it becomes unbiased.

Answer of exercise 8.5

if we have trajectories where every state is unique, it we do first visit or every-visit it doesn't matter (it is the same)

1. Since we have multiple instances of the same state in a single episode and their value is not the same, the two approach will provide different results.

2. Every-visit MC:

whenever we find a state in the trajectory we "create" a new trajectory:

V(A): (A,h,3) → (B,r,2) → (B,h,1) 6
(A,h,2) → (A,h,1) 3
(B,r,1) → (A,h,1) 1
(A,h,1) 1

$$V(A) = \frac{6+3+1+1}{4} = \frac{11}{4}$$
$$V(B) = \frac{3+1+2}{3} = 2$$

V(B): (A,h,3) → (B,r,2) → (B,h,1) 3
(A,h,2) → (A,h,1) 0
(B,r,1) → (A,h,1) 2
(B,h,1) 1

The fact that we "create" a new trajectory introduces a bias. However it's the reason why first-visit ≠ every-visit.

First-visit MC:

when we find a state in a trajectory we compute the value from there

V(A): (A,h,3) → (B,r,2) → (B,h,1) 6
(A,h,2) → (A,h,1) 3
(B,r,1) → (A,h,1) 1

$$V(A) = \frac{6+3+1}{3} = \frac{10}{3}$$
$$V(B) = \frac{3+2}{2} = \frac{5}{2}$$

V(B): (A,h,3) → (B,r,2) → (B,h,1) 3
(A,h,2) → (A,h,1) 0
(B,r,1) → (A,h,1) 2

3. TD:

(A,h,3) → (B,r,2) → (B,h,1)

$$\begin{cases} V(A) \leftarrow V(A) + 0.1(3 + V(B) - V(A)) = 0 + 0.1(3 + 0 - 0) = 0.3 \\ V(B) \leftarrow V(B) + 0.1(2 + V(B) - V(B)) = 0 + 0.1(2 + 0 - 0) = 0.2 \\ V(B) \leftarrow V(B) + 0.1(1 + V(C) - V(B)) = 0.2 + 0.1(1 + 0 - 0.2) = 0.28 \end{cases}$$

(A,h,2) → (A,h,1)

$$\begin{cases} V(A) \leftarrow V(A) + 0.1(2 + V(A) - V(A)) = 0.3 + 0.1(2 + 0.3 - 0.3) = 0.5 \\ V(A) \leftarrow V(A) + 0.1(1 + V(C) - V(A)) = 0.5 + 0.1(1 + 0 - 0.5) = 0.55 \end{cases}$$

(B,r,1) → (A,h,1)

$$\begin{cases} V(B) \leftarrow V(B) + 0.1(1 + V(A) - V(B)) = 0.28 + 0.1(1 + 0.55 - 0.28) = 0.407 \\ V(A) \leftarrow V(A) + 0.1(1 + V(C) - V(A)) = 0.55 + 0.1(1 + 0 - 0.55) = 0.595 \end{cases}$$

we end with 2 estimates < 1 when the aboves are all > 1. This is probably because we selected a learning rate too small to learn all with so few updates. (0.1? We should have at least 10 samples (10 updates))

because we have so few samples we are strongly conditioned on the initial guess (which in our case is pessimistic (0,0))

Answer of exercise 8.6

We know that if $\sum_{i \geq 0} \alpha_i = \infty$ and $\sum_{i \geq 0} \alpha_i^2 < \infty$ the estimator μ_i is consistent, thus, it converges to the real mean of the approximating problem when $i \to \infty$.

Thus the former choice $\alpha_i = \frac{1}{i}$ will provide a consistent estimator, while the latter $\alpha_i = \frac{1}{i^2}$ will be not guarantee to converge.

Answer of exercise 8.7

Any time you are able to use DP for solving a problem you might also consider to find the corresponding approximate solution with RL, thus, DP $\Rightarrow$ RL.

1. RL: we do not have the complete knowledge of the state we are in, thus we can not resort to DP.

2. DP and RL: we have complete information about the transition model and of the reward function. We might resort to RL if we consider continuous action space, which is usually difficult to handle with DP.

3. RL: in this case the transition is known (single state), but we do not know the reward associated to each action. In the specific, we might use MAB techniques to solve it.

4. RL: we have complete information, given a fixed strategy of the opponent, but usually we do not resort to DP for computational complexity issues.

Answer of exercise 8.8

1. FALSE If we use a greedy policy it might happen that we are not exploring some actions at all, while they are the optimal ones. This is especially true if the reward you gain from the states are stochastic.

2. TRUE The GLIE property requires that if we are considering an arbitrarily long learning process it will visit all the states an infinite number of times:

3. FALSE If we consider an off-policy learning method we are able to converge to the optimal policy even if we do not run the learned policy;

4. FALSE If we consider an off-policy learning method we might extract information about the optimal policy even if we consider data coming from sub-optimal ones.