Apprentissage par renforcement (3)

Bruno Bouzy
1 october 2013

This document consists in the 3rd part of the « Reinforcement Learning » chapter of the « Agent oriented learning » teaching unit of the Master MI computer course. It is based on part II chapters 5 and 6 of (Sutton & Barto 1998). The figures contained in this document are directly taken from the html version of (Sutton & Barto 1998). It concerns Monte-Carlo (MC) methods, temporal difference (TD) methods, action value methods such as Q-learning and Sarsa. The exemple of the grid world illustrates these methods.

Monte-Carlo

Introduction

The term Monte-Carlo (MC) consists in performing random simulations starting from a state we want to evaluate and to compute an empirical mean of the results. This method is suited when the transition probability function $P^a_{ss'}$ and the return function $R^a_{ss'}$ of the MDP are unknown. It is simple. The episodes must end. An episode is a simulation. The policy π is given. We look at evaluating the states with V^π, then we look at evaluating Q^π, . finally we look at how to improve a policy with MC control.

Monte-Carlo Evaluation of a policy

π is given and we look for an approximation of V^π. We store V(s) for every states. At the end of an episode, we update the mean value of the states encountered. Pseudo-code is given by figure 19.

Initialize:
  $\pi \leftarrow$ policy to be evaluated
  $V \leftarrow$ an arbitrary state-value function
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
  (a) Generate an episode using π
  (b) For each state s appearing in the episode:
    $R \leftarrow$ return following the first occurrence of s
    Append R to $Returns(s)$
    $V(s) \leftarrow$ average$(Returns(s))$

Figure 19: MC evaluation of V^π.

A back-up diagram shows the state to be updated and the states and transitions contributing to the update. Figure 20 gives the back-up diagram of the MC method. We start in a given state and we perform a random simulation by using the policy to evaluate. The simulation ends up in a terminal state. The cumulated reward received during the simulation is used to update the starting state.

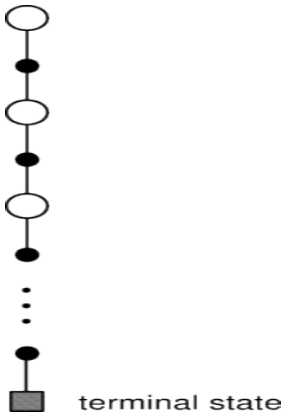

Figure 20: back-up diagram of MC estimation of V^{π}.

In this method, the computation of the value of a state does not depend of the computation of the value of a neighbouring state. To this extent, (Sutton & Barto) says the MC method does not « bootstrap ». The cost of MC evaluation depends on the length of episodes. We may focus on a part of the state space only. NB: if the policy and the environment are both deterministic, some states can remain unvisited. Therefore it important to introduce randmoness with the policy in such case.

Action value Monte Carlo estimation

It also possible to estimate $Q^{\pi}(s, a)$ for each state s and action a in a similar way.

Monte Carlo control

MC control means alternating policy improvement and policy evaluation, as shown by figure 21.

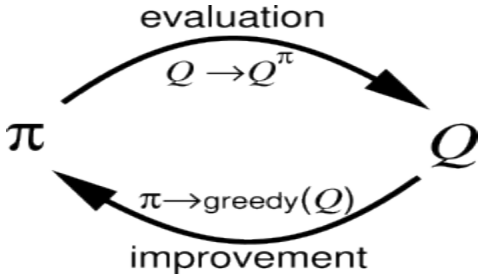

Figure 21: MC control.

Figure 22 shows an example of pseudo-code for MC control. Exploring starts mean that the first action is chosen randomly for a better exploration.

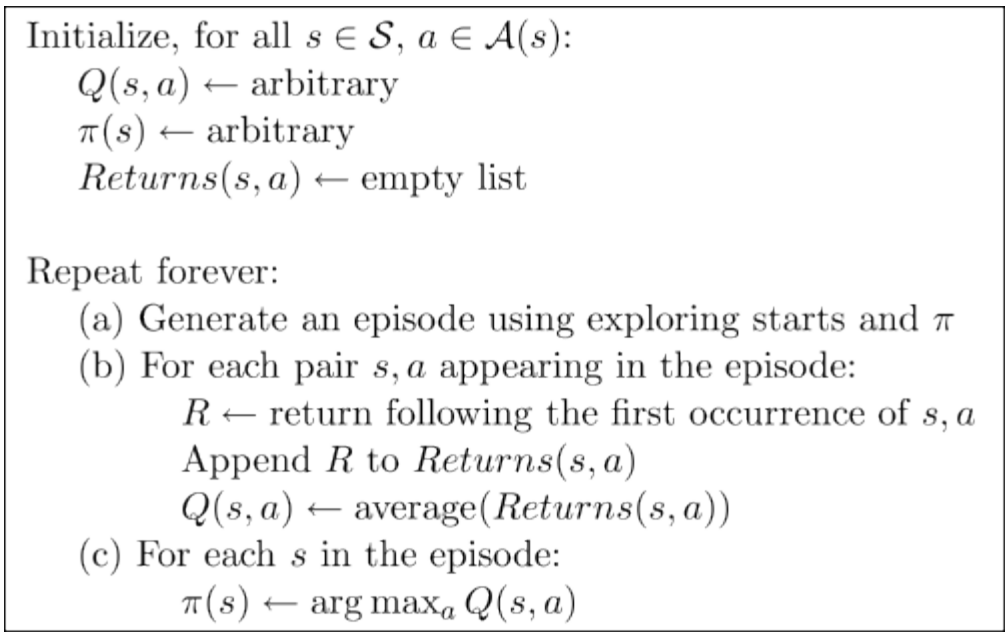

Figure 22: pseudo-code of MC control.

Monte-Carlo on the Grid World example:

Output of the GridWorld example for the MC control:

```
Monte Carlo Control: debut:
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
Monte Carlo Control: iteration 1:
  6.14   9.86   5.81   6.06   3.89
  3.80   4.33   3.41   3.10   2.58
  2.20   2.19   1.91   1.73   1.61
  1.35   1.28   1.16   1.08   1.06
  1.04   0.95   0.89   0.84   0.85
    >     <^>v    <       <^>v    <
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
   ^       ^       ^      <^       ^
Monte Carlo Control: iteration 2:
 19.66  21.87  19.66  16.51  14.88
 17.72  19.66  17.72  14.88  13.37
 15.96  17.72  15.96  13.37  12.01
 14.36  15.96  14.36  12.01  10.82
 12.87  14.36  12.87  11.20   9.73
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <
Monte Carlo Control: iteration 3:
 19.66  21.87  19.66  17.37  15.63
 17.72  19.66  17.72  15.96  14.22
 15.96  17.72  15.96  14.36  12.82
 14.36  15.96  14.36  12.87  11.55
 12.87  14.36  12.87  11.57  10.43
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
Monte Carlo Control: iteration 4:
 19.66  21.87  19.66  17.37  15.63
 17.72  19.66  17.72  15.96  14.36
 15.96  17.72  15.96  14.36  12.87
 14.36  15.96  14.36  12.87  11.57
 12.87  14.36  12.87  11.57  10.43
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
Monte Carlo Control, fin.
```

TD-learning

Introduction

Temporal difference (TD) method is central in RL. TD learns through the agent experience, like MC. Like MC, TD is suited when the transition probability function $P^a_{ss'}$ and the return function $R^a_{ss'}$ of the MDP are unknown. TD updates the value of states in function of the values of the neighbouring states, like in DP. In this meaning, TD bootstraps.

TD evaluation

For MC evaluation, the update rule can be written as:

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)] \qquad (30a)$$

TD uses this update rule :

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \qquad (30b)$$

The target of MC was R_t. Target of TD is $r_{t+1} + \gamma V(s_{t+1})$. Equations 13 and 15 can be written :

$$V^\pi(s) = E_\pi \{ r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s = s_t \} \qquad (31)$$

Figure 23 gives the pseudo-code of TD(0)[1].

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a; observe reward, r, and next state, s'
    $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
    $s \leftarrow s'$
  until s is terminal

Figure 23: TD(0) pour estimer V^π.

Figure 24 gives the back-up diagram of TD(0). The update of state s only depends on the unique actual following state. Notice the difference with DP in which the update rule includes all the neighbouring states.

Figure 24: back-up diagram of TD(0).

[1] TD has a parameter λ. TD(λ) is not the scope of this teachning unit. λ=0 corresponds to the simplest case.

TD on the Grid World example:

Output of the GridWorld example:

```
Temporal Difference Control: debut:
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
  <^>v    <^>v    <^>v    <^>v    <^>v
Temporal Difference Control: iteration 1:
  7.08  10.94   6.25   6.75   4.14
  4.01   4.64   3.64   3.35   2.83
  2.46   2.46   2.08   1.94   1.78
  1.47   1.42   1.32   1.23   1.21
  1.14   1.03   0.99   0.96   0.97
    >     <^>v    <       <^>v    <
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
Temporal Difference Control: iteration 2:
 21.98  24.42  21.98  18.45  16.60
 19.78  21.98  19.78  16.60  14.94
 17.80  19.78  17.80  14.94  13.45
 16.02  17.80  16.02  13.45  12.10
 14.42  16.02  14.42  12.10  10.89
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
Temporal Difference Control: iteration 3:
 21.98  24.42  21.98  19.42  17.48
 19.78  21.98  19.78  17.80  15.87
 17.80  19.78  17.80  16.02  14.35
 16.02  17.80  16.02  14.42  12.95
 14.42  16.02  14.42  12.98  11.67
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
Temporal Difference Control: iteration 4:
 21.98  24.42  21.98  19.42  17.48
 19.78  21.98  19.78  17.80  16.02
 17.80  19.78  17.80  16.02  14.42
 16.02  17.80  16.02  14.42  12.98
 14.42  16.02  14.42  12.98  11.68
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
Temporal Difference Control, fin.
```

Sarsa

Sarsa (StateActionRewardStateAction) is a control method that improve a policy and its action value function. Figure (32) starts on state s_t with action a_t , then it uses the reward r_{t+1}, next state s_{t+1} and its action a_{t+1}.

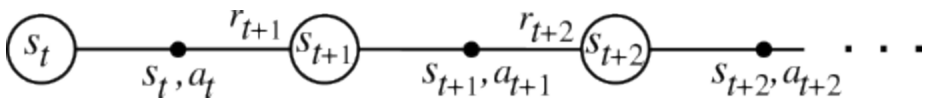

Figure 32

Sarsa updates Q with :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (32)$$

Figure (33) gives pseudo-code of Sarsa.

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Figure 33

The policy is implicitly represented by the Q values. Action a' corresponds to the Q value used for updating. To this extent, Sarsa is said to be an « online » method. Action choice is ε-greedy. It is possible to initialize Q values with high values to implicitly explore states less explored.

Q-learning

Q-learning (Watkins 1989) is similar to Sarsa. The difference with sarsa lies in the update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (33)$$

Figure (36) gives the pseudo-code of Q-learning and figure (37) gives the back-up diagram.

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$;
  until s is terminal

Figure 36: Q-learning

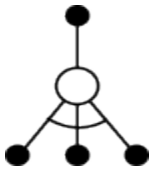

Figure 37: Back-up diagram of Q-Learning.

QL is also said to be an « on-line » method. However the actual action a' is not necessarily the optimale action that corresponds to the update. To this extent, QL is a slightly « offline » method.

Action choice is ε-greedy as for Sarsa.

Q Learning on the Grid World example:

```
Q Learning: debut:
20 20 22 20 |25 25 25 25 |22 20 20 20 |21 22 22 21 |20 18 18 18 |
18 21 20 18 |20 22 20 20 |20 20 18 18 |18 20 18 18 |18 18 16 16 |
17 19 18 17 |18 20 18 18 |18 18 16 16 |17 18 16 16 |16 17 15 15 |
15 17 16 15 |16 18 16 16 |17 17 15 15 |16 16 14 14 |15 16 14 14 |
14 16 15 14 |14 16 14 14 |15 16 14 14 |14 15 13 13 |14 14 13 12 |
Policy:
    >     <^>v    <         >     <
   ^       ^      <^       ^      <^
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
   ^       ^       ^       ^       ^
delta = 12.544 improve = 96 nb_pas_total = 10000
19 19 22 19 |24 24 24 24 |22 19 19 19 |20 20 20 20 |18 16 16 16 |
17 20 20 17 |19 22 19 19 |20 20 17 17 |18 18 16 16 |16 16 14 14 |
16 18 18 15 |17 20 17 16 |18 18 15 15 |16 16 14 14 |15 15 13 13 |
14 16 16 14 |15 18 15 15 |16 16 14 14 |14 15 13 13 |13 14 12 12 |
13 15 14 13 |14 16 14 14 |14 14 12 12 |13 13 12 11 |12 13 11 11 |
Policy:
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^       ^
   ^>      ^      <^       ^       ^
   ^       ^      <^       ^       ^
delta = 2.286 improve = 20 nb_pas_total = 20000
19 19 22 18 |24 24 24 24 |22 19 18 18 |19 19 19 19 |17 15 15 15 |
17 20 20 17 |18 22 18 18 |20 20 17 16 |18 17 15 15 |16 16 14 14 |
15 18 18 15 |16 20 17 16 |18 18 15 15 |16 16 13 13 |14 14 12 12 |
14 16 16 13 |15 18 15 15 |16 16 13 13 |14 14 12 12 |13 13 11 11 |
12 14 14 12 |13 16 13 14 |14 14 12 12 |13 13 11 11 |12 12 10 10 |
Policy:
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
delta = 1.058 improve = 16 nb_pas_total = 30000
19 19 22 18 |24 24 24 24 |22 19 18 18 |19 19 19 19 |17 15 15 15 |
17 20 20 16 |18 22 18 18 |20 20 16 16 |18 17 15 15 |16 16 14 13 |
15 18 18 15 |16 20 16 16 |18 18 15 14 |16 16 13 13 |14 14 12 12 |
13 16 16 13 |15 18 15 15 |16 16 13 13 |14 14 12 12 |13 13 11 11 |
12 14 14 12 |13 16 13 13 |14 14 12 12 |13 13 11 11 |12 12 10 10 |
Policy:
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
delta = 0.747 improve = 0 nb_pas_total = 40000
Q Learning: fin.
```

Rmax

Introduction

Rmax is a RL algorithm that builds and uses an environment model [Brafman & Tennenholtz 2002]. « Environment model » means the reward function R and the transition function P. The model is not correct at the beginning but it is refined as long as Rmax runs. When the environment model is updated, Rmax computes the optimal value fonction V* for this model with the value iteration algorithm. The environment model is initialized optimistically with Rmax, the maximal reward.

Rmax solves the exploitation – exploration dilemma. Because the initial rewards are optimistic, the optimal policy computed by value iteration leads the agent to explore the less explored states. Futhermore, since value iteration computes an optimal policy given the rewards, it is possible to say that Rmax optimally explores. Of course, the exploration is implicit.

Rmax can be applied to stochastic games (multi-agent MDP) and not only to single-agent MDP.

Algorithm

Init:
Input: policy length = T
Building the model M' made up with N states G_1, G_2, ..., G_N, plus one fictitious state G_0.
Each state owns:
- state value V.
- joint action matrix with, for each cell:
  - the reward initialized with R_{max}.
  - List of next states with, for each next state:
    - number of times the transition has been performed.
  - (list initialized with G_0, with one actual transition performed).
  - Boolean variable «known» or not, initialized with false.
  - Joint action value Q.

Execution
Repeat
- (C) Compute the length-T policy P starting on current state,
- Execute P during T timesteps.
- After each joint action and transition to a next state,
  - update the reward
  - increment the count on this transition.
  - If the count > threshold then
    - joint action is « known »
    - go to (C)

[Brafman & Tennenholtz 2002] proves that Rmax converges to the optimal policy.

Conclusion

This section sums up the properties of the RL methods and their ancestors. The methods are direct Bellmann equations solving, Dynamic programming (DP), Monte-Carlo (MC), Rmax, Temporal Differences (TD), QLearning (QL), Sarsa.

METHOD:	Direct solving	DP	Rmax	MC	TD	QL Sarsa
iterative ?	direct	iterative	iterative	iterative	iterative	iterative
State visit	ordered	ordered	Exper.	Exper.	Exper.	Exper.
Update frequ.	step	step	step	episode	step	step
Envir. Mod ?	yes	yes	built	No	No	No
reward ?	r	r	r	R	r	r
function:	V and/or Q	V and/or Q	V or Q	V and/or Q	V	Q
Kind of prob.	small	medium	big	Very big	Very big	Very big
off/on line?		«off»	«on»	«off»	«on»/«off»	«on»

Figure 41: Summary of the properties of the methods.

Conversely to direct Bellman equations solving, DP is an iterative method with « policy evaluation », « policy improvement », « policy iteration » and « value iteration ». Rmax, MC, TD and QL are iterative as well. Conversely to DP that sweeps space states in a given order, actual experiment conducts the order of updating states in RL methods. MC updates with a cumulated reward while the other method « bootstrap » i.e. they update the current state with the next states and the present reward. Direct solving, DP, Rmax and MC uses V or Q. TD uses V. QL and Sarsa use Q.

The interest of each method goes with the size of the problem to solve.: direct solving work for « small » problems. DP work for medium size problems. Rmax needs to have the environment model in memory. TD, QL or Sarsa work for « very big » problems.

Références

• R. Brafman, M. Tennenholtz, “Rmax – A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning”, J. of Machine Learning Research, pages 213-231, 2002.

• R. Coulom, “Apprentissage par renforcement utilisant les réseaux de neurones, avec des applications au contrôle moteur”, Ph.D. thesis, INPG, Grenoble, 2002.

• R. Sutton, A. Barto, “Reinforcement Learning”, MIT Press.

• R. Sutton, “Learning to Predict by the methods of Temporal Differences”, Machine Learning 3, pages 9-44, Kluwer, 1988.

• C. Watkins, “Learning from delayed rewards”, Ph.D. thesis, Cambridge University, 1989.

• C. Watkins, P. Dayan, “Q-learning”, Machine Learning, 8, pages 279-292, 1992.