Reinforcement Learning (2)

Bruno Bouzy

1 october 2013

This document is the second 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 chapter 4 of (Sutton & Barto 1998). The figures contained in this document are directly taken from the html version of (Sutton & Barto 1998) on the web. It deals with Dynamic Programming (DP) for RL.

Dynamic programming

Introduction

In this part we use Dynamic Programming (DP) to solve the Bellman equations of MDP. The general idea of DP is to use iterative algorithms that update the values of variables of an equation system to obtain approximate solutions of the equation system. DP is a general tool that helps solving various problems not necessarily related to MDP. In the case of MDP, DP helps to find approximate values of V and Q functions. This method assumes that the model of the environment (the P and R functions) is available. This part is important to understand the base of RL algorithms.

The key idea is to look at the Bellman equations:

$$V^*(s) = \max_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^*(s')] \quad (21)$$

$$Q^*(s, a) = \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma \max_{a'} Q^*(s', a')] \quad (22)$$

and to use them as affectations in an iterative algorithm.

We are going to see:

(1) how to evaluate a given policy with a required maximal error (**policy evaluation**),

(2) how to improve a policy with its evaluation (**policy improvement**),

(3) how to iterate (1) and (2) on a policy (**policy iteration**),

(4) how to gather (1) (2) (3) into a unique algorithm that finds V* the optimal value function of the optimal policy (**value iteration**).

Next, we mention asynchronous DP, generalized policy iteration, and the efficiency of this method.

1 Policy evaluation

Here, we assume a given policy π that we want to evaluate. The Bellman equations for V^π are:

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

$$= \sum_a \pi(s, a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^\pi(s')] \quad (15)$$

$\pi(s, a)$ is the probability to choose action a in state s. If the set S of states is finite with size |S|, then (15) is a system of |S| linear equations. Practically, when |S| forbids the direct solving of the system, DP proposes an iterative methode in which (15) becomes an **affectation**. We start with an initial V_0 and we compute V_{k+1} in function of V_k with:

$$V_{\kappa+1}(s) := \sum_a \pi(s, a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V_\kappa(s')] \quad (23)$$

V^π is a fixed point of 23. (V_k) converge to V^π when k goes to ∞. The algorithm based on 23 is named policy evaluation (figure 12).

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

Figure 12: Policy evaluation.

In order to write policy evaluation correctly (without taking the order in which the values are updated into account), we need 2 tables, one for the old values, one for the new values.The process that corresponds to the « for each » statement is called a sweep.

The Grid World Example

With $\theta = 0.01$, $\gamma = 0.9$ and 18 sweeps, policy evaluation gives the results of figure 12b:

3,31	*8,78*	*3,86*	*3,67*	*0,63*
1,5	*2,9*	*1,94*	*1,3*	*0,05*
0,03	*0,67*	*0,52*	*0,11*	*-0,65*
-0,98	*-0,47*	*-0,43*	*-0,69*	*-1,3*
-1,86	*-1,36*	*-1,27*	*-1,48*	*-2,04*

Figure 12b: Results of policy evaluation. Each number is the V value.

Exemple 2

A second example to illustrate policy evaluation is the 4x4 gridworld of figure 13. The upper left and the lower right cells are terminal states. The agent starts on an arbitrary cell. The actions are {up, down, left, right}. An action yields one next state: the adjacent cell in direction of the action, except if the target cell is outside the grid. In such a case, the agent stays on the same cell. The reward is -1 at any timestep. The agent uses the random policy with uniform probability.

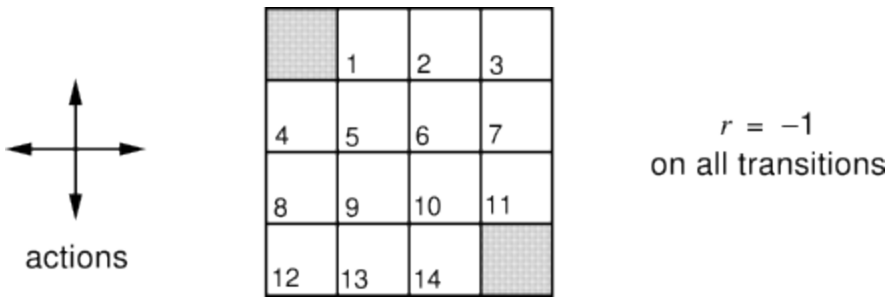

Figure 13: the 4x4 Grid world

The left column of figure 14 gives the sequence of values of V computed with policy evaluation.

Exercices

1) Program policy evaluation on the 4x4 gridwolrd example.

2) What are the equations for Q^{π} ?

3) What happens if cell 15 at the bottom right is non terminal state ?

2 Policy improvement

The next stage is to improve a policy with a given V function.

-- *masque*
--

Let us consider π' the greedy policy in Q^{π} :

$$\begin{aligned} \pi'(s) &= \arg\max_a Q^{\pi}(s, a) \\ &= \arg\max_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^{\pi}(s')] \end{aligned} \quad (27)$$

-- *masque*
--

The process giving the greedy policy in Q^{π} with policy π, is called policy improvement. If this process gives the policy itself, then the policy is optimal.

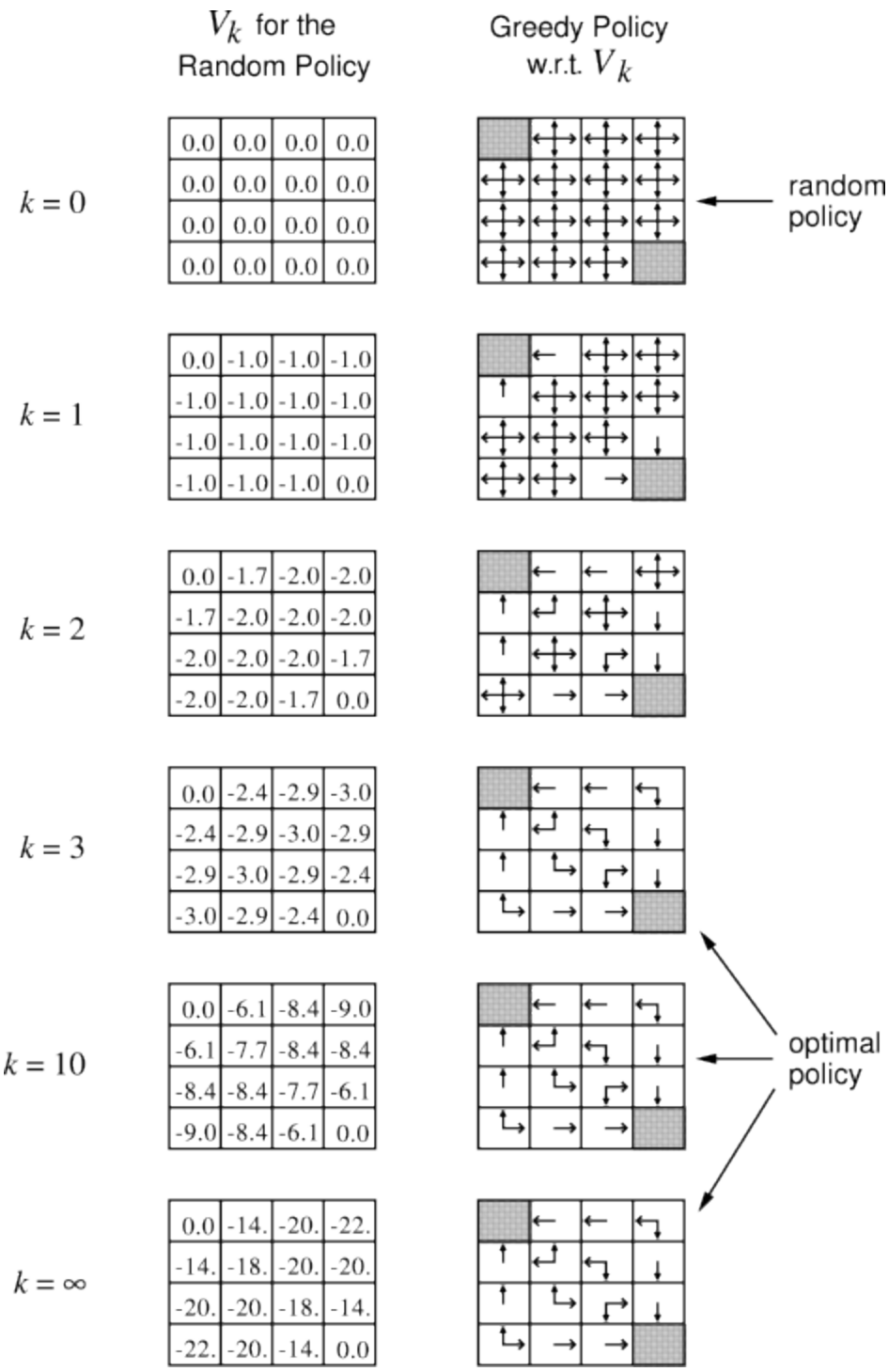

Figure 14: Convergence of policy evaluation on the 4x4 Grid World. The right column shows the policies associated to the V values on the left column.

3 Policy iteration

The policy π being evaluated with V^{π}, and improved to give policy π', it is possible to iterate this process several times, and obtain a series of improved policies and value functions. When the policy is not improved by an iteration, then it is the optimal policy. Because the MDP is finite, it has finite number of policies and this sequence of policies converge to the optimal policy in a finite times of iterations. Figure 15 gives the policy iteration algorithm.

1. Initialization
   $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left[\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s') \right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $b \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V(s') \right]$
   If $b \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop; else go to 2

Figure 15: The policy iteration algorithm.

Figure 14 shows that policy iteration would converge in one iteration on the grid world example (2).

Policy iteration output on the Grid World example 1

```
Policy Iteration: 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
Policy Iteration: iteration 1:
```

```
  3.38   9.03   4.12   3.96   0.88
  1.59   3.07   2.15   1.54   0.29
  0.17   0.85   0.73   0.34  -0.41
 -0.78  -0.26  -0.20  -0.45  -1.04
 -1.62  -1.13  -1.02  -1.22  -1.77
    >     <^>v    <       <^>v    <
   ^       ^       ^       ^      <
   ^       ^       ^       ^      <^
   ^       ^       ^       ^      <^
   ^       ^       ^       ^       ^
Policy Iteration: iteration 2:
 21.86  24.35  21.91  12.17  10.96
 19.68  21.91  19.72  10.96   9.86
 17.71  19.72  17.75   9.86   8.87
 15.94  17.75  15.97   8.87   7.99
 14.35  15.97  14.38   7.99   7.19
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
   ^>      ^      <^      <       <^
Policy Iteration: iteration 3:
 21.86  24.35  21.91  16.62  14.96
 19.68  21.91  19.72  17.75  14.72
 17.71  19.72  17.75  15.97  13.81
 15.94  17.75  15.97  14.38  12.68
 14.35  15.97  14.38  12.94  11.53
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <
   ^>      ^      <^      <^      <
   ^>      ^      <^      <^      <
Policy Iteration: iteration 4:
 21.86  24.35  21.91  16.62  14.96
 19.68  21.91  19.72  17.75  15.97
 17.71  19.72  17.75  15.97  14.38
 15.94  17.75  15.97  14.38  12.94
 14.35  15.97  14.38  12.94  11.64
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
Policy Iteration: iteration 5:
 21.86  24.35  21.91  16.62  14.96
 19.68  21.91  19.72  17.75  15.97
 17.71  19.72  17.75  15.97  14.38
 15.94  17.75  15.97  14.38  12.94
 14.35  15.97  14.38  12.94  11.64
    >     <^>v    <       <^>v    <
   ^>      ^      <^      <       <
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
   ^>      ^      <^      <^      <^
Policy Iteration, fin.
```

Value iteration

A drawback of policy evaluation is the number of iterations to evaluate a policy. The key idea of value iteration is to evaluate roughly the policy with one iteration only, and improve the policy with this rough evaluation. Furthermore, it is possible to aggregate policy evaluation and policy improvement is the same loop, making disappear the policy.

$$V^{\pi}(s) = \max_a E_{\pi} \{ r_{t+1} + \gamma \quad V^{\pi}(s_{t+1}) \mid s = s_t, a = a_t \} \\ = \max_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma \; V^{\pi}(s')] \qquad (29)$$

This gives pseudo-code of figure 16.

Initialize V arbitrarily, e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Figure 16: value iteration

Asynchronous Dynamic Programming

The drawback of DP is sweeping the whole state at each iteration. The asynchron DP principle is to sweep part of the space states. So as to converge, an asynchron DP algorithm must garantee to visit every states a number of times sufficiently high. This idea of not visiting the whole space sates at each sweep is reused in RL algorithms. In « generalised policy iteration », two processes are running simultaneously: one for evaluating policies, and one for improving policies. Figures 17 and 18 show graphically this idea.

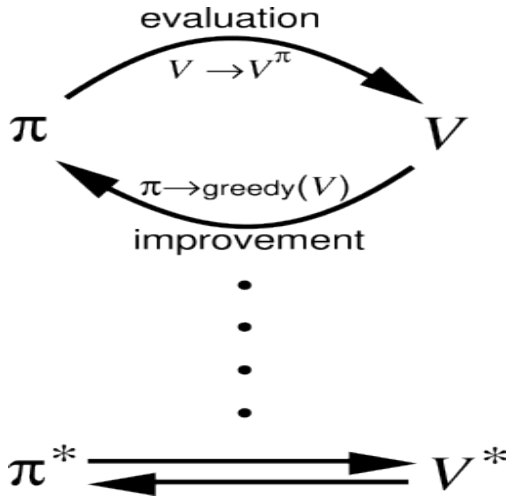

Figure 17: Generalized policy iteration.

6 Efficiency

Time used by DP to converge is known to be polynomial in the number n of states and in the number m of actions, although the number of deterministic policies is in m^n. To this extent DP is a very efficient method. However, DP is limited when the number of states is increasing exponentially in the number of variables of the problem.

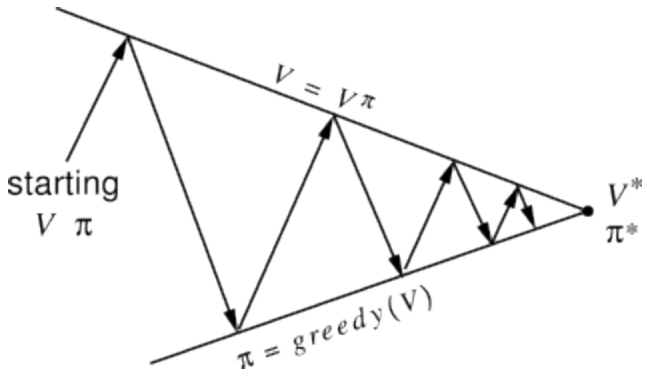

Figure 18: Generalized policy iteration (2).