École polytechnique fédérale de Lausanne

Disopt

Semester project

Reinforcement learning and robot navigation

Student:
Charles Dufour

Supervisors:
Jonas Racine
Prof. Friedrich Eisenbrand

Contents

1 Theory

1.1 Introduction

Reinforcement learning
Reinforcement learning is learning what to do, how to map situations to actions so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. [1]

Some technical terms :

- *policy* : it is a mapping from the states to the actions
- *reward* : what our learning agent is trying to maximize
- *model* of the environment : the laws governing the environment

"Reinforcement learning methods specify how the agent's policy is changed as a result of its experience"[1]

The usual way to formulate the reinforcement learning problem from a mathematical point of view is by using what we call Markov's decision processes (MDPs).

1.2 MDP : Markov decision processes

Markov's decision processes are composed by :

- a set of states : $\mathcal{S} = \{s_0, s_1, \ldots\}$
- a set of actions : $A = \{a_1, a_2, \ldots\}$
- a transition function : $T(s, a, s') \sim Pr(s' \mid a, s) \quad s, s' \in \mathcal{S}$ which gives the state transition probabilities
- a reward function : $\mathcal{R} : \mathcal{S} \mapsto \mathbb{R}$
- The Markov property : the transitions only depends on the current state and action

When an agent is learning in a MDP, what it observes is a sequence of states, actions and rewards: suppose the agent is in the state s_0 and chooses action a_1 and then end up in state s_1 with reward r_1; then the sequence observed is of the form : $s_0, a_1, s_1, r_1, s_2, \ldots$

Markov decisions processes are usually represented with graph : the nodes are the states and the directed edges from a node Q are the actions an agent can choose to make while being in state Q, which will bring the agent in the state represented by the node at the endpoint of the edge as we can see in figure 1.2.

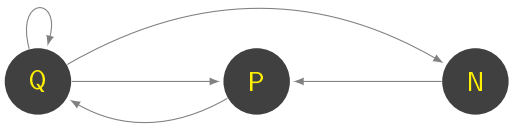

Figure 1: example of a graph representing a Markov Decision process

1.2.1 Policies and Value functions

A policy : $\pi : \mathcal{S} \mapsto A$ is a mapping from states to action. If we follow this policy (way of behaving) we can define the value function for a policy in order to compare them, which links a state and its expected reward if we follow this policy.

The discount factor γ helps making our learning agent more or less far-sighted : the greater γ the more "impact" will have a late reward on our reward sequence, hence making the agent more conscious about these actions.

We define the return as : $G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

And $\pi(a \mid s)$ is the probability that $A_t = a$ if $S_t = s$

Then we can define the value of taking action a in state s while following the policy π

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}[G_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a] \\ &= \sum_{r,s'} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{1}$$

And the value of a state s under a policy:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[G_t \mid S_t = s] \\ &= \mathbb{E}[\sum_{k=0}^{\infty} \gamma^t R_{t+k+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{r,s'} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{2}$$

1.2.2 Optimal policies and Optimal value function

For finite MDPs, the value function can define a partial order in the space of policies :

$$\pi \leq \pi' \Leftrightarrow \pi(s) \leq \pi'(s) \quad \forall s \in \mathcal{S}$$

An optimal policy is a policy which is greater or equal than any other policy. This is what we are interested to find.

Bellman optimality equations The optimal policy π^* has optimal value functions : v_* and q_*, which satisfy the relations below :

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')] \tag{3}$$

$$q_*(s,a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{4}$$

These are called the *Bellman optimality equations.*

For finite MDPs these equations have a unique solution. We can note that if we know v_* or q_* a greedy approach to define a policy (best in the short term) becomes a long-term optimal solution : indeed defining a policy being greedy in function of v_* implies that you go to the best state possible, the one with the bigger expected reward.

1.3 Solving MDPs with dynamic programming

In general we don't have all the information we need to compute the exact value of v_* or even if we have them, we don't have the computational power needed. We often use approximation of value-function instead.

From now on we assume our MDPs are finite, even if it is possible to extend everything to infinite MDPs if we are careful enough to avoid the problematic ones.

1.3.1 Policy iteration

This method uses two processes : the first one is policy evaluation : we compute the value function of a policy for all the states $s \in \mathcal{S}$; then we use policy improvement to get a better policy by acting greedy.

Policy evaluation We begin by setting arbitrary $v(s) \quad \forall s \in \mathcal{S}$. Then we update using one of the following update rules :

- $v_{k+1}(s) = \sum_{a \in A} \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)(r + \gamma v_k(s'))$ this is called "iterative policy evaluation".
- we can use the same update rule as before, but use new information as soon as it is available : this kind of upgrade algorithm are called in places.

From this we derive the algorithm 1.

Algorithm 1: Iterative policy evaluation (in place)

Input : policy to evaluate π
Output: $V \approx v_\pi$
Initialize $V = 0$
while $\Delta \geq \epsilon$ **do**
  $\Delta = 0$
  for $s \in \mathcal{S}$ **do**
    $v = V(s)$
    $V(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid a, s)[r + \gamma V(s')]$
    $\Delta = \max(\Delta, \mid v - V(s) \mid)$
  end
end
return $V \approx v_\pi$

Policy improvement Then we try to find a better policy : we try to determine whether we should change $\pi(s)$ to $a \neq \pi(s)$. In order to do so, we try to first select action a while being in state s and then following the policy. If the expected reward we get by doing this choice is better than the one we get by simply following our policy, we should improve.

Mathematically speaking we will compare the value of taking action a while being in the state s : $q_\pi(s, a)$ to the value of s: $v_\pi(s)$. Then we would greedily improve our policy this way.

The greedy update rule we use to improve our policy is from which we derive algorithm 2 :

$$\pi'(s) = \arg\max_{a \in A} q_\pi(s, a)$$

Algorithm 2: Policy improvement

Input : policy to improve π
Output: π' s.t : $\pi' \geq \pi$
for $s \in \mathcal{S}$ **do**
  $\pi'(s) = \arg\max_{a \in A} q_\pi(s, a)$
end
return π'

Policy Iteration By combining the two processes described before, we can derive an algorithm to sweep through all our states and upgrade our policy until the changes between each sweep is too small : it is controlled by a parameter ϵ and a parameter γ (which we talked about previously).This is the algorithm 3.

The reason why the algorithm 3 works is called the the *policy improvement theorem*:

Theorem 1. *Let π and π' be any pair of policies such that $\forall s \in \mathcal{S}$:*

$$q_\pi(s, \pi(s)) \geq v_{\pi'}(s) \tag{5}$$

then :

$$\pi \geq \pi' \tag{6}$$

Moreover if there is a strict inequality in all the states in 5 then there must be at least a strict inequality for one state in 6

Proof. The idea of the proof is to expand the q_π side until we get $v_{\pi'}(s)$ using Equation 1 in page 3.
Indeed we have :

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(a)] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1}) \mid S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})] \mid S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s] \\ &\leq \dots \\ &= v_{\pi'}(s) \end{aligned} \tag{7}$$

□

Algorithm 3: Policy Iteration

Input : arbitrary policy, stopping criterion ϵ
Output: estimation of the optimal policy and of its value function

```
Policy evaluation :
while Δ ≥ ε do
    Δ = 0
    for s ∈ S do
        v = V(s)
        V(s) = Σ_a π(a | s) Σ_{s',r} p(s', r | a, s)[r + γV(s')]
        Δ = max(Δ, | v − V(s) |)
    end
end
Policy improvement :
policy-stable = true
for s ∈ S do
    old-action= π(s)
    π(s) = arg max_{a∈A} q_π(s, a)
    if old-action ≠ π(s) then
        policy-stable = false
    end
end
if policy-stable then
    return V ≈ v* and  π ≈ π*
else
    Go to Policy evaluation
end
```

1.3.2 Value Iteration

Value iteration is another way of approximating an optimal policy : it combines in each of its sweep improvement and evaluation in algorithm 4.

Here the main difference is the max in the evaluation line.

1.3.3 Other types of DP method

There exists some others dynamic programming algorithm to solve these problems :

Algorithm 4: Value iteration

Input : policy
Output: estimate of optimal policy
Initialize V arbitrarily **while** $\Delta \geq \epsilon$ **do**
  $v = V(s)$
  $V(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')]$
  $\Delta = \max(\Delta, \mid v - V(s) \mid)$
end
return $\pi \approx \pi^* \; s.t \; : \pi(s) = arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')]$

- Asynchronous Dynamic programming : it doe not sweep amongst all the states at each iteration. They are in place iterative DP algorithms.
- General Policy Iteration (GPI): they are mixing the two components of policy iteration (evaluation and improvement) a little bit more than the algorithm we already saw.

1.3.4 Efficiency of the DP method

2 Journal

28.02.2018 finished reading chapter 2 Sutton's book about the `k-bandits problem` : implementation of simple algorithms of the book on jupyter notebook in `Rl-sandbox`

01.03.2018 initiated the Latex journal

04.03.2108 read the notebook about `numpy` and tried to go on with the lecture of the literature–> have to read again the example about the golf

finished the chapter 3

12.03.2018 read chapter three again and made a summary of it

Then tried to attack the street racer problem

15.03.2018 tried to understand exactly what the problem of the street racing was about and tried to define a real reward function after coding the matrices for each action Then went back to studying chapter 4 in order to implement the policy evaluation/improvement functions

16.03.2018 finished reading chapter 4 of Sutton's book continued the summary

17.03.2018 finished typing the résumé so that I could concentrate on the actual code. I'm not sure about including a subsubsection about the efficiency of the DP method but for now I'm just putting the title to remember .

I have a little problem of references for my first graph but I'll see to that later

$$v_\pi(s) = R(s) + \gamma \sum_{s \in \mathcal{S}} P(s' \mid s, \pi(s)) v_\pi(s') \tag{8}$$

Et je pense que cette equation traduit bien le cas non stochastic, est-ce que j'explique en plus ?

Then I tried to finish the street racing program but too many bugs appeared so I stop for today, and I'll try again tomorrow

21.03.2018 Finished design the matrices and the model for the racing car, but issues seem to appear , the algorithm doesn't seem to work

22.03.2018 fixed issues in the code : still strange things happening : the score get optimum too quickly....

References

[1] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA, 1st edition, 1998.