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
- *value function* : what our learning agent is trying to optimize
- *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 (MDP's).

1.2 MDP : Markov decision processes

Markov's decision processes are composed by :

- a set of states : $\mathcal{S}$
- a set of actions : A
- 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

The Bellman equation

$$v_\pi(s) = R(s) + \gamma \sum_{s \in \mathcal{S}} P(s' \mid s, \pi(s)) v_\pi(s') \tag{1}$$

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \tag{2}$$

These are two formulations of the bellman equation used to compute optimal policies by iteration : (1) is from the MOOC and (2) is from Sutton's book [1]

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 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

$$q_\pi(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a] \tag{3}$$

And the value of a state s under a policy:

$$v_\pi(s) = \mathbb{E}[G_t \mid S_t = s] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^t R_{t+k+1} \mid S_t = s] \quad \forall s \in \mathcal{S} \tag{4}$$

1.2.2 Optimal policies and Optimal value function

For finite MDP's, 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.

Bellman optimality equations

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')] \tag{5}$$

$$q_*(s,a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{6}$$

For finite MDP's 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.

1.3 Solving MDP's 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.

We assume our MDP's are finite.

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 :

- $v_k(s) = \sum_{a \in A} \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)(r + \gamma v_k(s'))$ 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.

Algorithm 1: Iterative policy evaluation (in place)

```
Input  : policy to evaluate $\pi$
Output: $V \approx v_\pi$
1  Initialize $V = 0$
2  while $\Delta \geq \epsilon$ do
3      $\Delta = 0$
4      for $s \in \mathcal{S}$ do
5          $v = V(s)$
6          $V(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid a, s)(r + \gamma V(s'))$
7          $\Delta = \max(\Delta, \mid v - V(s) \mid)$
8      end
9  end
10 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 :

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

Algorithm 2: Policy improvement

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

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).

Algorithm 3: Policy Iteration

```
Input  : arbitrary policy, stopping criterion $\epsilon$
Output: estimation of the optimal policy and of its value function
1  Policy evaluation :
2  while $\Delta \geq \epsilon$ do
3      $\Delta = 0$
4      for $s \in \mathcal{S}$ do
5          $v = V(s)$
6          $V(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid a, s)(r + \gamma V(s'))$
7          $\Delta = \max(\Delta, \mid v - V(s) \mid)$
8      end
9  end
10 Policy improvement :
11 policy-stable = true
12 for $s \in \mathcal{S}$ do
13     old-action= $\pi(s)$
14     $\pi(s) = \arg\max_{a \in A} q_\pi(s,a)$
15     if old-action $\neq \pi(s)$ then
16         policy-stable = false
17     end
18 end
19 if policy-stable then
20     return $V \approx v^*$ and $\pi \approx \pi^*$
21 else
22     Go to Policy evaluation
23 end
```

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

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

$$q_\pi(s, \pi(s)) \geq v_{\pi'}(s) \tag{7}$$

then :

$$\pi \geq \pi' \tag{8}$$

Moreover if there is a strict inequality in all the states in 7 then there must be at least a strict inequality for one state in 8

Proof. The idea of the proof is to expand the q_π side until we get $v_{\pi'}(s)$ using :

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] = \\ &= \sum_{r,s'} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{9}$$

□

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 :

Algorithm 4: Value iteration

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

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 :

- 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.

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 and typed the end of the resume

References

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