REINFORCEMENT LEARNING — THEORY AND PURPOSE

1. Introduction

Purpose of Project

Reinforcement Learning

Markov Decision Process. Markov decision processes is the formalism which allows us to reason about reinforcement learning in a mathematical way, and all standard reinforcement learning problems are formulated in a Markov decision process framework. In what follows we give a (semi) formal definition of a stationary Markov decision process. The building blocks of a Markov decision process consist of

(i) S a finite space of *states*
(ii) A a finite space of *actions*
(iii) $P_a(s, s')$ a transition probability function defined for all $(s, a, s') \in S \times A \times S$
(iv) $r(s, a)$ the immediate or expected immediate reward of taking action a in state s.
(v) $\gamma \in [0, 1]$, the discount factor (here always equal to 1).

We define $S_t \in S$ to be the sate of the process at time t; A_t is the (potentially random) action taken at time t, and R_t is the immediate reward at time t. The *history* H_t of the system up to time t is the random vector given by $(S_0, A_0, S_1, A_1, \ldots, S_{t-1}, A_{t-1}, S_t)$. An important concept going further is that of a policy: A policy $\pi = (\pi_0, \pi_1, \ldots)$ is a sequence of decision rules, where π_n, for each history of the process up to time n, determines a probability measure over A.

Now, we call $(S, A, P_\cdot(\cdot, \cdot), r(\cdot, \cdot), \gamma)$ a Markov decision process if

$$\begin{aligned}\mathbb{P}(S_t = s | S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \ldots, S_0 = s_0, A_0 = a_0)& \\ = \mathbb{P}(S_t = s | S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}) = P_a(s_{t-1}, s).& \quad (1)\end{aligned}$$

At first glance this expression makes sense, but there are subtle details which need to be addressed. For instance, we have said nothing about the process $\{A_t\}$ and therefore we should be careful defining conditional probabilities containing it. However, if $\{A_t\}$ is governed by a policy $\pi = (\pi_0, \pi_1, \ldots)$ then we can define the above probability. In order for the tuple to be a Markov decision process we formally require that (1) holds for all policies. This is still a bit vague, and for a thorough treatment see REFERENCE. We also require that R_{t+1}, given A_t and S_t, is independent of the history H_t.

The goal of reinforcement learning is to take a MDP and choose a policy which maximizes expected (discounted) rewards, and at first glance this can seem hopeless since policy decisions in general depends on the whole history of the chain. However, it can be shown that for finite stationary MDPs this is equivalent to maximizing rewards over policies which take into account only the current state of the process and which maps this to a single action, i.e. $\pi_n(H_n) = \pi(X_t) \in A$, see Reference. This is something that make the optimizing problem considerably easier, and allows the *value iteration* and *Q-learning* algorithms to converge to an optimal solution. Hence, from here on out we consider only policies $\pi : S \to A$.

Value Iteration. For a given policy π define the *value function*

$$V^\pi(s) = \mathbb{E}_\pi(\sum_{n=0}^{\infty} R_{t+1}|X_0 = s)$$

i.e. the expected reward obtained when following policy π. We say that a policy π^* is optimal if

$$V^{\pi^*}(s) \geq V^\pi(s) \qquad \forall s \text{ and policies } \pi.$$

It can be shown that for finite Markov decision processes such a policy exists uniquely. Under the optimal policy π^* let $V^*(s) = V^{\pi^*}(s)$. Say we know $V^*(s)$ for all s, can this be used to calculate an optimal policy? The answer is yes. Note by the Markov property of the process we must have

$$V^*(s) = r(s, \pi^*(s)) + \sum_{s'} P_{\pi^*}(s, s')V^*(s') = \max_{a\in A}(r(s, a) + \sum_{s'} P_a(s, s')V^*(s')).$$

Hence, $\pi^*(s)$ is given by the action a which achieves the above maximum. So if we can find a way to calculate the optimal value function we get the optimal strategy. In small state space the value function can actually be calculated but in large space approximation is the best one can hope for. Value iteration is an algorithm which iteratively updates the values of its estimates of V^*. It can be shown that the estimations converges towards the correct optimal values. The algorithm is given by

```
(i) Initiate $V(s)$.
(ii) Repeat
    For all $s \in S$: $V(s) \leftarrow \max_{a\in A}(r(s, a) + \sum_{s'} P_a(s, s')V(s'))$.
(iii) Stop when $V(s)$ has converged.
```

Finally, $\pi(s) = \arg\max_{a\in A}(r(s, a) + \sum_{s'} P_a(s, s')V(s')) \approx \pi^*(s)$.

Q-Learning. The big drawback with value iteration is that we need to know $P_a(s, s')$ and $r(s, a)$ a priori — Q-learning does not require this knowledge. Here we will assume that the underlying MDP is terminal, i.e. it has some final state-

In order to specify the algorithm we need some definitions. Define the state-action function as

$$Q^\pi(s, a) = r(s, a) + \sum_{s'} P_a(s, s')V^\pi(s')$$

that is the expected reward of starting in state s taking action a and then following the policy π. For terminal state we set $Q = 0$. Furthermore, define the optimal state-action function as

$$Q^*(s, a) = (r(s, a) + \sum_{s'} P_a(s, s')V^*(s'))$$

i.e. the expected reward of being in state s taking action a and then following the optimal policy π^*. With the same argument as with we see that if we can calculate Q^* we can calculate π^*. The following iterative algorithm is known as Q-learning, and it will produce estimates which converges to Q^*

```
(i) Initialize $Q(s, a)$ and learning rate $\alpha \in (0, 1)$
(ii) Repeat for each $N$ episodes:
    While $s_t$ not terminal: $Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha_t(r_t + \max_{a\in A} Q(s_{t+1}, a))$
```

Open Gym AI implementation of Black Jack

As a Markov Decision Process.

Results

Q-Learning.

Value Iteration.

Conclusion