Machine Learning in Robotics
Lecture 11: Introduction to Reinforcement Learning

Prof. Dongheui Lee

Institute of Automatic Control Engineering
Technische Universität München

dhlee@tum.de

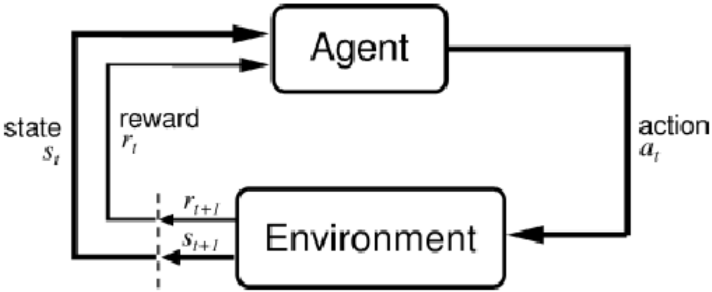
Reinforcement Learning

Learning of a behavior without explicit information about correct actions

- Between supervised and unsupervised learning
- No training patterns, but **rewards**
- Inspired by principles of human and animal learning
- Mild assumptions on the process to be controlled
- A control strategy can be learned from scratch

Architecture

The agent-environment interaction in reinforcement learning



The Environment

- The environment contains the process to be controlled
- Markov Decision Process (MDP): The environment is modeled by an MDP which is tuple $(S, A, \{P_{sa}\}, \gamma, R)$
  - S is a set of **states**
  - A is a set of **actions**
  - P_{sa} are the **state transition probabilities**.
  - $\gamma \in [0, 1)$ is the **discount factor**.
  - $R : S \times A \mapsto \mathbb{R}$ is the **reward function** (Rewards can also be a function of state S only and in that case $R : S \mapsto \mathbb{R}$).

Markov Process, HMM, Markov Decision Process

Task for the Agent

Find a behavior which maximizes the expected total reward

For how long should we consider?

Finite Horizon

$$\max \left[\sum_{t=0}^{T} r_t \right]$$

Infinite Horizon

$$\max \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

γ is a discount factor $(0 \leq \gamma < 1)$

Reward function

The reward function controls which task should be solved

- Game (Checkers, chess)
  Reward only at end: +1 when winning, -1 when loosing
- Avoiding mistakes (pole balancing)
  Reward -1 at the end (when falling)
- Find a fast/short/cheap path to a goal
  Reward -1 at each step

Simplifying assumptions

- Discrete time
- Finite number of actions $a_i \in a_1, a_2, a_3, \ldots, a_n$
- Finite number of states $s_i \in s_1, s_2, s_3, \ldots, s_m$
- Environment is a stationary markov decision process
- Reward r only depends on s

Policy and Value function

- Policy
  Policy provides a mapping from states to action.
  $$\pi(s) \mapsto a$$
- Value Function
  Expected total future reward when starting from s and following policy π

$$\begin{aligned} V^{\pi}(s) =& E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots | s_0 = s, \pi] \\ =& R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s') \qquad \text{(Bellman's equation)} \end{aligned}$$

Optimal Policy

An optimal policy is the the one which maximizes the value function

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

$$\pi^*(s) = arg \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

Classical problem: Grid world

- 11 states. Each **state** is represented by a position in the grid world.
- The agent **acts** deterministically by moving to other position. A={N,S,E,W}
- reward: $R(4,3) = 1$, $R(4,2) = -1$, $R(s) = -0.02$ for all other states
- transition probability: 0.8 for a planned state and 0.1 for the other adjacent two states.

Value Iteration

For each state s, initialize $V(s) := 0$.
Repeat until convergence
$\{$
For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$
$\}$

$V(s)$ can be updated in synchronous and asynchronous manner.

Policy Iteration

Initialize π randomly.
Repeat until convergence
$\{$
(a) Let $V := V^{\pi}$
(b) For each state s, let $\pi(s) := \arg\max_{a\in A} \sum_{s'} P_{sa}(s')V(s')$
$\}$

Step (a) can be calculated by solving linear equations (with equal number of equations and unknowns).

Monte-Carlo Method

Start at some random state.
Follow π, store the rewards and s_t.
When the goal is reached, update $V^{\pi}(s)$ estimation for all visited states with the future reward we actually received.

- Monte-Carlo method is suitable only for episodic tasks
- Learns incrementally from episode-by-episode but not step-by-step

Temporal Difference Learning

There are two estimates of the value of a state:

- Before: $V^{\pi}(s_t)$
- After: $R_{t+1} + \gamma V^{\pi}(s_{t+1})$

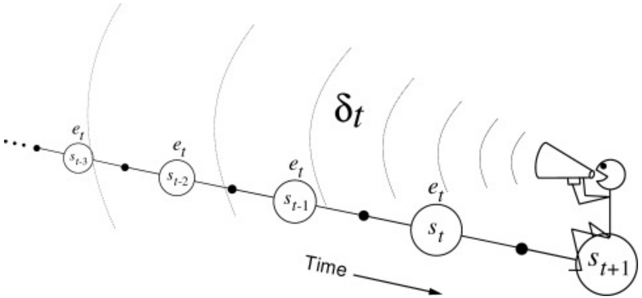
Temporal Difference Learning

Idea: The second estimate is better!

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha\left(R_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)\right)$$

- Learns considerably faster than the Monte-Carlo method
- Step by step learning.

Eligibility Trace



Q-Learning

Whenever reward r or next state s' cannot be predicted, we cannot calculate π even with a good estimate for V

$Q^{\pi}(s,a)$, *is the expected infinite-horizon discounted return for executing* a *in state* s *and thereafter following* π

$$Q^{\pi}(s,a) = E\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | s_t = s, a_t = a, \pi\right]$$

Q-Learning

Whenever reward r or next state s' cannot be predicted, we cannot calculate π even with a good estimate for V

$Q^{\pi}(s,a)$, is the expected infinite-horizon discounted return for executing a in state s and thereafter following π

$$Q^{\pi}(s,a) = E\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | s_t = s, a_t = a, \pi\right]$$

$$\pi(s) = arg\max_a Q(s,a)$$

$$V^*(s) = \max_a Q^*(s,a)$$

Q-Learning

Whenever reward r or next state s' cannot be predicted, we cannot calculate π even with a good estimate for V

$Q^\pi(s,a)$, *is the expected infinite-horizon discounted return for executing* a *in state* s *and thereafter following* π

$$Q^\pi(s,a) = E\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | s_t = s, a_t = a, \pi\right]$$

$$\pi(s) = arg\max_a Q(s,a)$$

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right]$$

Autonomous helicopter flight via RL

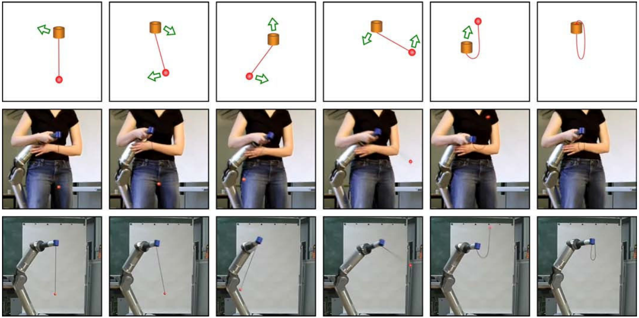
Value of each policy is calculated through Monte-Carlo method *PEGASUS*

method uses the observation that almost all computer simulations sample $s' \sim P_{sa}(.)$ by first calling a random number generator to get one (or more) random numbers p, and then calculating s' as some deterministic function of the input s, a and the random p Since the helicopters model is stochastic,

random number were fixed in advanced to evaluate different policies

HJ Kim, Michael I Jordan, Shankar Sastry, and Andrew Y Ng., *Autonomous helicopter flight via reinforcement learning*, In Advances in neural information processing systems, 2003.

Learning Motor Primitives using Reinforcement Learning



POWER is an Expectation Maximization based RL algorithm which does not require learning rate as a parameter:

$$\theta' = \theta + \frac{E\{\sum_{t=1}^{T} \varepsilon_t Q^{\pi}(s_t, a_t, t)\}}{E\{\sum_{t=1}^{T} Q^{\pi}(s_t, a_t, t)\}}$$ where ε_t is exploration term

Jens Kober and Jan Peters, *Learning motor primitives for robotics*, pp. 2112 - 2118, ICRA, 2009.

Reading Material

- Mitchell, Chapter 13
- Russell and Norvig, Artificial Intelligence: A Modern Approach, Chapter 21
- Sutton and Barto, Reinforcement Learning: An Introduction