

Artificial Intelligence

Markovian Decision Processes

Daniel Polani

Scenario: sequence of decisions where

1. each decision may lead randomly to different outcomes
2. each decision is connected with a reward
3. rewards cumulate to total utility
4. rewards may be delayed

Relevance for Social Intelligence: adaptive models for social interaction

Example

Given: n -armed bandit problem. Each trial of some bandit arm, costs $1\mathcal{L}$.

Payoff: distributions with differing mean and variance per arm (finite and unknown)

Seeking: strategy with maximum payoff.

Temporal conditions:

- limited
- unlimited
- weighted

time

Exploration/Exploitation Conflict

Problem: finding balance between exploration and exploitation leads to conflict

“Greedy” Strategy: short-term gain — remember “tragedy of the commons”

Note: in the following we will ignore the intricacies of the exploration/exploitation dilemma and use only simple strategies for their control

Assumption: reward for an action a is random variable with mean $U^*(a)$.

Choose: sequence of actions $a_1, a_2, \dots, a_t, \dots$. Then obtain estimate for $U^*(a)$ via

$$U_t(a) = \frac{r_{t_1^a} + \dots + r_{t_{k_a}^a}}{k_a}$$

where the r at the times $t_{i=1, \dots, k_a}^a$ which are the times between 1 and t where action a is chosen.

Initialization: $U_0(a)$ is initially set to any value, e.g. 0.

“Greedy” Strategy: choose action

$$a^* = \operatorname{argmax}_{a \in \mathcal{A}} U_t(a)$$

Alternative: ϵ -Greedy — with probability $1 - \epsilon$, choose greedy, with probability ϵ random action. Advantage: More exploration

Incremental Estimates

Remark (Incremental Computation of $U_t(a)$): consider t actions having been made and the action at time $t + 1$ to be $a_{t+1} = a$. Then

$$\begin{aligned} U_{t+1}(a) &= \frac{1}{k_a + 1} \sum_{i=1}^{k_a+1} r_{t_i^a} \quad (\text{note: } t_{k_a+1}^a = t + 1) \\ &= \underbrace{U_t(a)}_{\text{old value}} + \underbrace{\frac{1}{k_a + 1}}_{t\text{-dependent factor}} \cdot \underbrace{(r_{t+1} - U_t(a))}_{\text{deviation from target value}} \end{aligned}$$

Notes

Notes:

1. need only to store $U_t(a)$ and k_a
2. the update $w_{t+1} = w_t + \alpha(t)(x_{t+1} - w_t)$ is very common in learning systems
3. $U_{t+1}(a') = U_t(a')$ for $a' \neq a_{t+1}$

Nonstationary Environments: give more importance to recent values, e.g.

$$U_{t+1}(a = a_{t+1}) = U_t(a) + \underbrace{\alpha}_{\text{e.g. constant in } [0, 1]} [r_{t+1} - U_t(a)]$$

Def. (state): a full description of the current situation, agent and world

Def. (policy): A *policy* π_t at a time t is a conditional probability that an agent in a state s chooses an action a :

$$P(a_t = a \mid s_t = s) = \pi_t(s, a)$$

Agent: at a time step t it has access to

- current state s_t
- reward just obtained r_t
- current policy π_t

From This: calculate current action choice a_t and follow policy π_{t+1} .

Markovian Decision Process

Note:

- full access to current state
- border between agent and environment given by *absolute control*, not by limitation of knowledge

Note: goals of agent specified by rewards r_t

Goal: long-term maximization of cumulated rewards

Example

Examples: reward structure as follows

1. robot is supposed to learn to move: $r_t = 1$ if step ahead
2. maze: 0 per step, 1 for a step outside the maze
3. maze: -1 per step in the maze

Objective

Question: have sequence of rewards $r_{t+1}, r_{t+2}, r_{t+3}, \dots$
 What do we want to maximize?

In General: want to maximize total payoff R_t

2 Cases:

Episodic Tasks: tasks with natural end time T :

$$R_t = r_{t+1} + \dots + r_T$$

Unlimited Tasks: " $T \rightarrow \infty$ " require *discounting*

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \end{aligned}$$

with $\gamma \in [0, 1)$

Value Function

Value Function: a measure how good it is for an agent to be in a certain state. This depends on future actions (more precisely, on policy)

$$V^\pi(s) = \mathbf{E}_\pi(R_t | s_t = s) = \mathbf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right)$$

Q-Function: a measure how good it is for an agent to be in a state and picking a certain action (again depends on the policy in the following states)

$$Q^\pi(s, a) = \mathbf{E}_\pi(R_t | s_t = s, a_t = a) = \mathbf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right)$$

Bellmann Equation

Theorem: with

$$\mathcal{P}_{ss'}^a := P(s_{t+1} = s' | s_t = s, a_t = a)$$

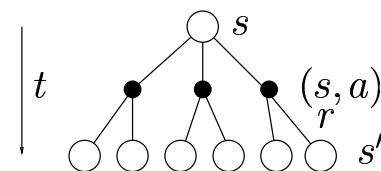
$$\mathcal{R}_{ss'}^a := \mathbf{E}(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s')$$

one has

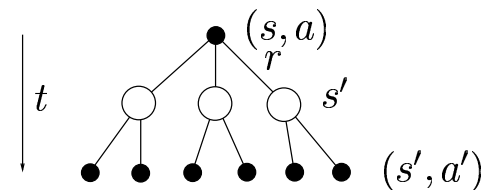
$$V^\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(s, a) \sum_{s' \in \text{Succ}(s)} \mathcal{P}_{ss'}^a (\mathcal{R}_{ss'}^a + \gamma V^\pi(s'))$$

Backup Diagrams

Backup Diagram for V:



Backup Diagram for Q:



Comparison of Policies: we say that $\pi \geq \pi'$ (π is at least as good as π') if for all states s we have $V^\pi(s) \geq V^{\pi'}(s)$.

Theorem: there exists always an *optimal* policy π^* , i.e. $\pi^* \geq \pi$ for all policies π .

Note: an optimal policy is not necessarily unique! But V^{π^*} is the same for all optimal policies. Therefore write V^* for optimal value function.

Analogously: define $Q^*(s, a)$

Remark: one has

$$Q^*(s, a) = \mathbf{E}(r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a)$$

Bellmann's Optimality Equation: one has

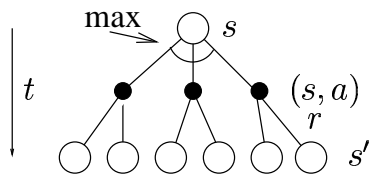
$$V^*(s) = \max_a \mathbf{E}(r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a)$$

Analogously:

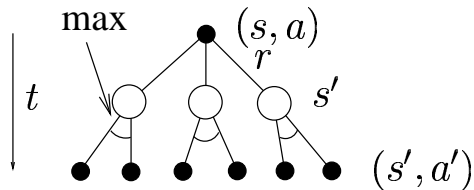
$$Q^*(s, a) = \mathbf{E} \left(r_{t+1} + \gamma \max_{a' \in \mathcal{A}(s_{t+1})} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a \right)$$

Backup Diagrams

For V^* :



For Q^* :



Learning Methods

Methods:

- dynamic programming
- value iteration
- Q-learning

Q-Learning: update rule given by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Theorem (Watkins): if all (s, a) are being update often enough, Q converges towards Q^* , independently of policy π .

Advantages:

- off-policy
- does not require explicit averaging — done implicitly as you go
- no model required

Reinforcement Learning: uses Q -Learning as central model — many variants and improvements exist.

General Remarks

Reinforcement Learning: learning from *delayed* rewards

In Particular: Q -Learning requires

- no model of dynamics
- only immediate backup
- but: state must have

Markov Property: the result of an action must only depend on a current state variable which must be known to the agent. In particular

1. it must not depend on some “memory” effects unseen by the agent
2. it must not depend on the history of the agent or the world

Gambler's Problem

Scenario:

- playing red/black
- starting and exiting with given amount