3010

ARTIFICIAL INTELLIGENCE

Lecture 5 A Brief Introduction to Reinforcement Learning

Masashi Shimbo

2019-05-23

Today's Agenda

Introduction

Review: LRTA*

Markov decision problem

Real-time dynamic programming

Q-learning

Function approximation

Summary

Outline

- Review of the online search algorithm LRTA* and its relation to asynchronous dynamic programming
- Extending LRTA* to stochastic state space leads to Real-Time Dynamic Programming, a model-based reinforcement learning method
- Q-learning, a popular model-free reinforcement learning method, is introduced
- By “model-free”, we mean the transition probability and incurred costs are not known—these have to be found by actually exploring the state space (i.e., trying out many different actions in different states)

Three types of machine learning tasks

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Supervised machine learning 教師付き機械学習

- Given a set of input-output pairs, find the relationship between input and output
- Typically, each input is assumed independent
  - ➥ Previous decisions made by an agent do not affect future input/output
- Sometimes not realistic—obtaining input-output pairs can be expensive

Unsupervised machine learning 教師なし機械学習

No supervision is provided; just a bunch of input objects are given

Find common patterns, or group them according to some similarity measure between objects

Examples:

Frequent pattern mining, clustering, etc.

Reinforcement learning 強化学習

- No explicit supervision (i.e., input-output pairs) is provided
- but “reward”/“cost” is provided as actions are taken
- An agent’s decision may affect not only immediate but also future outcome
- The agent’s goal is to maximize the expected total **future** reward/minimize the expected total **future** cost —not the *immediate* reward/cost
- Weaker form of supervision than supervised learning (strong explicit supervision), but stronger than unsupervised learning (no supervision at all)

Reward/cost 報酬/コスト

Also called **reinforcement signal**

1. a number (a real value)
2. can be negative;
   - negative reward is often called **penalty** or **cost**
3. A concept borrowed from behavioral psychology 行動主義心理学
   "Animals try to behave in a way that the reward they receive is maximized"
   - positive reward works for **reinforcing** the agent's tendency to take the actions that have led to its reception
   - negative reward (penalty) works for discourage the action

Reward/cost 報酬/コスト

Also called **reinforcement signal**

1. a number (a real value)
2. can be negative;
   - negative reward is often called **penalty** or **cost**
3. A concept borrowed from behavioral psychology 行動主義心理学
   "Animals try to behave in a way that the reward they receive is maximized"
   - positive reward works for **reinforcing** the agent's tendency to take the actions that have led to its reception
   - negative reward (penalty) works for discourage the action

Online search

The agent is not taught precisely which action to take at each time step, but instead a cost is incurred on taking an action

- Online search (like LRTA*) can be thought of as a very simple form of reinforcement learning

Today's Agenda

Introduction

Review: LRTA*

Markov decision problem

Real-time dynamic programming

Q-learning

Function approximation

Summary

LRTA*

v: current state of the agent

Repeat Steps 1 and 2 until v is a goal state:

1. Look-ahead and update

$$h[v] \leftarrow \min_{v' \in \mathrm{Succ}(v)} c(v, v') + h[v']$$

2. Action execution Move to a successor state that gives the minimum in the formula above

$$v \leftarrow \operatorname*{argmin}_{v' \in \mathrm{Succ}(v)} c(v, v') + h[v']$$

LRTA*: Repeated application

After the agent reaches a goal state, put it back to the initial state and repeat

Then, after running LRTA* for a sufficiently long time,

- h-values converge to the actual cost-to-go h^*:
  For every node v along the shortest path,

$$h[v] = h^*(v)$$

- All the moves made by the agent are "optimal":
  The agent eventually learns to move to successor v' along the optimal path, i.e.,

$$v' = \underset{v' \in \mathrm{Succ}(v)}{\mathrm{argmin}}\ c(v, v') + h^*(v')$$

  (We say the agent has learned an optimal **policy**)

LRTA*'s update formula: Where does it come from?

Update formula of LRTA*:

$$h[v] \leftarrow \min_{v' \in \mathrm{Succ}(v)} c(v, v') + h[v']$$

Bellman's equation for the shortest-path problem:

$$h^*(v) = \min_{v' \in \mathrm{Succ}(v)} c(v, v') + h^*(v')$$

Notice the similarity

Asynchronous dynamic programming

$$h[v] \leftarrow \min_{v' \in \mathrm{Succ}(v)} c(v, v') + h[v'] \qquad \text{(update formula of LRTA*)}$$

If this update formula is applied to every state $v \in V$ (not just the current state chosen at each iteration of the LRTA*) infinitely often, then h will converge to h^*; i.e.,

$$h[v] = h^*(v) \quad \text{for all } v \in V$$

after update is carried out a sufficiently large number of times on all states

- Of course, the number of states is often enormous, and updating all states infinitely often is impractical
- LRTA* combines "greedy" control (decision making/move) and value update so that it only updates the portion of the state space but still learns the optimal policy

Today's Agenda

Introduction

Review: LRTA*

Markov decision problem

Real-time dynamic programming

Q-learning

Function approximation

Summary

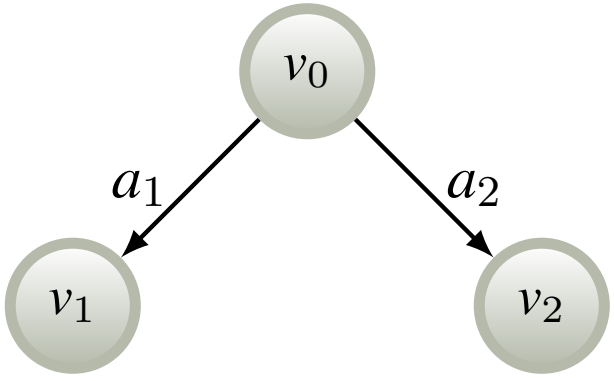
Stochastic state space 確率的な状態空間

Transition is not deterministic

Outcome of taking an action is not always the same

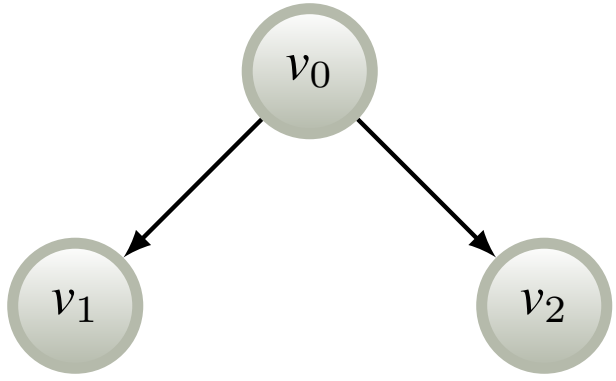
➥ An action cannot be identified with an edge anymore

deterministic state space



$$P(v_1 \mid v_0, a_1) = 1.0$$
$$P(v_2 \mid v_0, a_1) = 0.0$$
$$P(v_1 \mid v_0, a_2) = 0.0$$
$$P(v_2 \mid v_0, a_2) = 1.0$$

stochastic state space



$$P(v_1 \mid v_0, a_1) = 0.8$$
$$P(v_2 \mid v_0, a_1) = 0.2$$
$$P(v_1 \mid v_0, a_2) = 0.5$$
$$P(v_2 \mid v_0, a_2) = 0.5$$

Stochastic state space 確率的な状態空間

- For each state $v \in V$, a set of admissible actions $A(v)$ is available
- Taking an action $a \in A(v)$ leads to a state $v' \in \mathrm{Succ}(v)$ **with probability** $P(v' \mid v, a)$

  Note: $\sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v, a) = 1$
- Taking an "action" a in state v incurs a cost $c(v, a)$

This is a form of **Markov decision problem**—transition only depends on the current state, but not states visited earlier

Objective of agent in stochastic state space

- LRTA* finds an optimal policy in the deterministic state space
  - ➥ Optimal policy = decision rules that takes shortest route to the goal
- We want to design an agent architecture that achieves a similar objective in the stochastic state space
  - ➥ What is an “optimal” policy in the stochastic space? i.e., Taking an action does not always result in the desired destination state
  - ➥ “shortest path” does not make sense in stochastic space

Agent's objective

A "good" **policy** is the one that incurs small **expected total future cost**

- Objective of an agent is to find an **optimal policy**, which is a policy that minimizes the expected total future cost

Policy

A policy is a function from states to actions

$\pi : V \rightarrow A$

A policy determines which action should be taken at each state

Total future cost

Suppose we followed a policy π, and observed a state sequence $v_0, v_1, v_2 \ldots$ (an **episode**)

The total cost incurred in this episode is:

$$c(v_0, \pi(v_0)) + \gamma c(v_1, \pi(v_1)) + \gamma^2 c(v_2, \pi(v_2)) + \cdots$$
$$= \sum_{t=0}^{\infty} \gamma^t c(v_t, \pi(v_t))$$

- v_t: state of the agent at time t
- $c(v, a)$: cost incurred by taking action a at state v
- γ: problem-specific **discount factor** $0 < \gamma \leq 1$ (set to $\gamma < 1$ to ensure the summation always converges)

Expected total future cost and optimal policy

Expectation of total future cost starting from state v satisfies the following formula

$$U^{\pi}(v) = c(v, \pi(v)) + \gamma \sum_{v' \in \text{Succ}(v)} P(v' \mid v, \pi(v)) U^{\pi}(v')$$

Optimal expected total future cost $U^*(v)$ is

$$\begin{aligned} U^*(v) &= \min_{\pi} U^{\pi}(v) \\ &= \min_{\pi} \left[c(v, \pi(v)) + \gamma \sum_{v' \in \text{Succ}(v)} P(v' \mid v, \pi(v)) U^{\pi}(v') \right] \end{aligned}$$

A policy π^* that satisfies $U^{\pi^*}(v) = U^*(v)$ is called an optimal policy

It can be shown that:

- Minimum total future cost U^* satisfies

$$U^*(v) = \min_{a \in A(v)} \left[c(v,a) + \gamma \sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v, a) U^*(v') \right]$$

  (This is also called **Bellman's optimality equation**)

- Optimal policy π^* is the one that satisfies

$$\pi^*(v) = \operatorname*{argmin}_{a \in A(v)} \left[c(v,a) + \gamma \sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v, a) U^*(v') \right]$$

Bellman's equation in stochastic state space

when transition probability is known

$$U^*(v) = \min_{a \in A(v)} \left[c(v,a) + \gamma \sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v,a) U^*(v') \right]$$

Update formula for dynamic programming:

$U^*(v)$ can be found by (asynchronous) dynamic programming:

$$U[v] \leftarrow \min_{a \in A(v)} \left[c(v,a) + \gamma \sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v,a) U[v'] \right]$$

if initially $U[t] = 0$ for all goal states $t \in V$

Today's Agenda

Introduction

Review: LRTA*

Markov decision problem

Real-time dynamic programming

Q-learning

Function approximation

Summary

Real-time DP [Barto, Bradtke, Singh, 1995]

Interleaves planning and execution, like LRTA*

Update step (only for the current state v):

$$U[v] \leftarrow \min_{a \in A(v)} \left[c(v,a) + \gamma \sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v,a) U[v'] \right]$$

Act greedily according to the current U

Execute action a' that gives the minimum above:

$$a' = \operatorname*{argmin}_{a \in A(v)} \left[c(v,a) + \gamma \sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v,a) U[v'] \right]$$

If multiple actions give the minimum, choose among them randomly

Real-time DP: Property

Real-time DP will find optimal policy on the relevant states, if the following conditions are met:

- agent is reset to the initial state once it reaches a goal state
- $U[t] = 0$ for every goal state t
- All cost $c(v, a) > 0$ for every state v and action $a \in A(v)$
- $U[v]$ is initially non-overestimating for all state v; i.e., initially $U[v] \leq U^*(v)$ for all states v

Today's Agenda

Introduction

Review: LRTA*

Markov decision problem

Real-time dynamic programming

Q-learning

Function approximation

Summary

Model-free reinforcement learning

What if $P(v' \mid v, a)$ or $c(v, a)$ is not known? Can't we find optimal policy in this case?

It is possible, through trial and error (i.e., by actually trying every action in every state)

—but this makes it not easy to design an architecture like RTDP or LRTA* (i.e., that uses greedy action selection)

Model-free reinforcement learning

- One simple way is to first estimate the transition probability $P(v' \mid v, a)$, by actually trying every action a at every state v a sufficient number of times, observing how frequently the transition to each state v' occurs
  Then use these probability estimates with the value iteration procedure or real-time DP
- Another way is to directly estimate the Q**-values** when trying out every action at every state
  — (Temporal difference) Q-learning

Q-values

Define Q-value for state-action pair (v, a) as:

$$Q^*(v,a) = c(v,a) + \gamma \sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v,a) U^*(v') \qquad (*)$$

Plug (*) this into Bellman's equation:

$$\begin{aligned} U^*(v) &= \min_{a \in A(v)} \left[c(v,a) + \gamma \sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v,a) U^*(v') \right] \\ &= \min_{a \in A(v)} Q^*(v,a) \qquad (**) \end{aligned}$$

Plugging (**) back to (*) yields:

$$Q^*(v,a) = c(v,a) + \gamma \sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v,a) \overbrace{\min_{a' \in A(v')} Q^*(v',a')}^{U^*(v')}$$

Q-values

Thus, optimal Q-values must satisfy

$$Q^*(v,a) = c(v,a) + \gamma \sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v,a) \overbrace{\min_{a' \in A(v')} Q^*(v',a')}^{U^*(v')}$$

Again, we turn this into an update formula:

$$Q[v,a] \leftarrow c(v,a) + \gamma \sum_{v' \in \mathrm{Succ}(v)} P(v' \mid v,a) \min_{a' \in A(v')} Q[v',a']$$

We still have $P(v' \mid v,a)$ in the formula, but $Q[v,a]$ can be estimated by the method of **temporal difference** without estimating $P(v' \mid v,a)$

TD Q-Learning

Whenever an action a is executed in state v leading to state v' and receiving cost c, update $Q[v,a]$ by

$$Q[v,a] \leftarrow Q[v,a] + \beta(n(v,a)) \left(c + \gamma \min_{a' \in A(v')} Q[v',a'] - Q[v,a] \right)$$

where

- $\beta(n)$: learning rate function; $0 < \beta(n) \leq 1$ for all $n \geq 0$
- $n(v,a)$: number of times action a was executed in state v so far

Convergence of Q-learning

$Q[v, a]$ converges to $Q^*(v, a)$ if the following conditions are met:

- Q-values are updated infinitely often on all state-action pairs
- Learning rate function β satisfies $\sum_i \beta(i) = \infty$ and $\sum_i \beta(i)^2 < \infty$
- All costs are bounded; i.e., there exist a constant C such that $|c| < C$ holds for any cost c received

Today's Agenda

Introduction

Review: LRTA*

Markov decision problem

Real-time dynamic programming

Q-learning

Function approximation

Summary

Problem with tabular representation of parameters

Note that so far, we assumed that $U[v]$ and $Q[v, a]$ are represented in tabular form

- The table U has the size of number of states $|V|$
- The table for Q has the size of $|V| \times |A|$

These tables must be filled with values (=learning parameters)

Not efficient if the number of states is large

Function approximation

to reduce number of parameters

Represent state v as a vector of **features**

$$v = [x_1, x_2, \ldots, x_m]$$

Define $U[v]$ and $Q[v, a]$ as a parametric function of these features

$$U(v) = f(x_1, x_2, \ldots, x_m; \theta)$$

where θ is a set of parameters

Instead of learning $U[v]$, learn parameter θ

Example

For a node represented as a feature vector

$$v = [x_1, x_2, \ldots, x_m],$$

define $U(v)$ to be a linear function with parameters $\theta = [\theta_1, \ldots, \theta_m]$, i.e.,

$$U(v) = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_m$$

Then learn $\theta_1, \theta_2, \ldots, \theta_m$ instead of tabular representation for $U[v]$, using, e.g., gradient-based training

More complex functions can be modeled by, for example, artificial neural networks

If two states have many feature values in common, parameters adjusted for one state also influences the value for the other state.

➥ learning can be more efficient

However, choosing unsuitable function representation makes it difficult to learn U^* or Q^*

Today's Agenda

Introduction

Review: LRTA*

Markov decision problem

Real-time dynamic programming

Q-learning

Function approximation

Summary

Model-based reinforcement learning

RTDP, LRTA*, etc.

Applicable when model of the state space (i.e., $P(v' \mid v, a)$, $c(v, a)$) is available

Model-free reinforcement learning

Q-learning, SARSA, etc.

Applicable even if $P(v' \mid v, a)$, $c(v, a)$) are not available—agent has to find optimal policy by actually exploring the state space, trying every action in every state

- Useful when the model of the problem is not available, or if the problem is difficult to model
- Major problem is the learning inefficiency

Applications

Reinforcement learning has been used successfully in various tasks, most notably in game playing:

- TD-Gammon [Tesauro, 1992]
  combines temporal difference learning with function approximation using a neural network
- Alpha Go
  Q-learning and neural network-based function approximation (Deep Q-Networks)