Lecture 17
Reinforcement Learning

Extending learning situation

basic learning scenarios differ according the feedback available to the learner

- supervised learning: correct output available
- unsupervised learning: no feedback, need to construct measure of good output
- reinforcement learning: scalar feedback, possibly temporally delayed

general RL set-up:

- choose actions that can change state of environment
- reward function defines quality of action sequence

Reinforcement learning

RL difficult, with many sources of complexity

- actions: outcomes may be unknown and/or nondeterministic
- reward/utility: may be unknown and/or delayed
- states: can be partially observable

our approach:

1. Markov chains – dynamical systems that evolve (change state) over time
2. introduce decisions – maximize (known) reward based on Markov chain model
3. decision-making when reward not known

Markov Chain

Markov chains consist of:

- finite number of states
- possibly empty set of absorbing states
- probabilistic transition function from state to state

Let S be the set of states, T_{ij} be the probability of going from state $i \in S$ to state $j \in S$ in a unit of time.

For all $i \in S$, $\sum_{j \in S} T_{ij} = 1$

Markov property – probability in a state depends only on previous state & transition function

Algorithm:

1. initial state s_0 chosen according to distribution π_i
2. state at time $t (t \geq 1)$ chosen from distribution $T_{s_{t-1}}$.

Markov Decision Processes

Markov chains let us reason about behavior of dynamical systems where no decisions take place – can figure out probability of each outcome

For decision-making – need to decide how *useful* each outcome is

Example - prefer states with higher probability of reaching your goal

decision theory: maximize expected utility (value)

each time step t, agent:

- senses state s_t
- chooses and performs action a_t
- receives reward $r_t = r(s_t, a_t)$: reward function $r()$
- moves to next state $s_{t+1} = \delta(s_t, a_t)$: state transition function $\delta()$

MDP formulation

tuple: $\langle S, A, \delta, r, \gamma, h \rangle$

- S: set of states
- A: set of actions
- δ: transition function $\delta(s, a) = s'$
- r: reward function $r(s, a) = 3$
- γ: discount factor
- h: planning horizon

Goal: find policy π that maximizes expected accumulated future rewards $V^{\pi}(s_t)$, obtained by following π from state s_t:

$$\begin{aligned} V^{\pi}(s_t) &\equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \\ &= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned}$$

alternative return is average: $\frac{1}{t+1}(r_0 + \ldots + r_t)$

MDP assumptions

Markov property: state transition and reward functions depend only on current state and action, not on any earlier states or actions

model can be non-deterministic (state transitions, reward); for now assume deterministic, also assume actions, states finite

γ constant between 0 and 1: determines relative value of future versus immediate rewards (discounts future exponentially)

policy: what action a_t to choose in state s_t: $\pi(s_t) = a_t$ (not fixed sequence of actions – conditional plan)

value function $V(s)$: measures accumulated future rewards (value) from state s (or can be associated with state-action pair)

Basic Problems

Planning: Given complete MDP as input, compute strategy with optimal expected return

Learning: Only have access to experience in the MDP, learn a near-optimal strategy

We will focus on learning (planning discussed along the way)

Reinforcement Learning

aim: compute optimal policy (best action to take from any state)

$$\pi^*(s) = \arg\max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

computed based on optimal value function $V^*(s)$ (value associated with following π^* from state s) – Bellman's equation:

$$V^*(s) = \max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

can be computed if reward $r()$ and state transition functions $\delta()$ known, but in RL these not known: must learn an optimal policy for an unknown environment, thru exploration

trial consists of sequence of state-action pairs (s_t, a_t)

assume that only environmental feedback available is sequence of rewards obtained during trial

RL and dynamic programming: typically DP assumes state-transition, reward functions known; RL learns these along with values/policies

Reinforcement Learning: Examples

many natural problems have structure required to be RL problems:

1. game playing: know win/lose, but not specific moves (TD-gammon)
2. control: for traffic lights, can measure delay of cars, but not how to decrease it
3. robot juggling
4. robot path planning: can tell distance traveled, not how to minimize

Adaptive House

house of future:

- stereo lowers volume when phone rings
- VCR records shows of interest automatically
- temperature adjusted just before come home
- lighting fits mood - reading, sleeping, lounging

utility function trades off *comfort* and *efficiency* (save energy/money)

need to predict occupancy patterns; also depends on factors like outside temperature, time of day, activities

training signals: inhabitants use controls (light switches/dimmers; thermostats; stereo controls; TV remote; air vents) as normally would; house must monitor controls, sensors over time

input representations: sensor states; external variables (temperature, time, day of week)

output: control states, optimized by exploiting statistical regularities

reinforcement learning – system tries to conserve energy (keep lights low as possible), need to provide feedback, reinforcement by changing levels (turning up lights), so will adjust better next time