4/11/2023: Reinforcement Learning I

Last time: **Bandits**

Agent at time t takes action A_t
receives reward R_t

No persistent world state - every time step is independent
- Each user of website independent

Today: Actions affect world state
- Choosing classes
  - Action: take a class
  - Rewards: Enjoyment
  - State: Satisfy more preregs

  e.g. A class may be good to take later but bad to take now
- Robotics
  - Actions: motor torques
  - Reward: complete a task
  - State: Position of robot, other objects

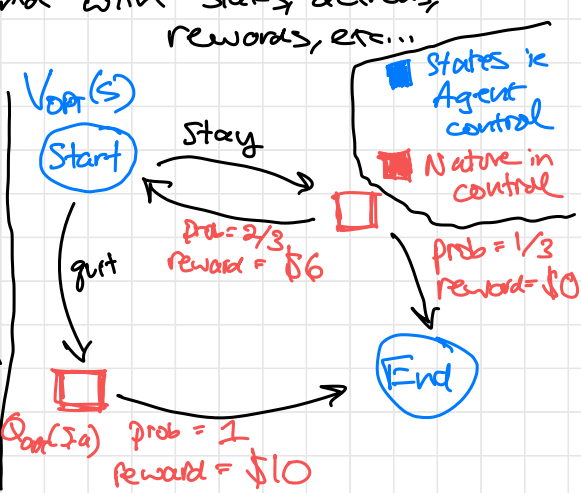
  State transitions are noisy / random
- Video games

Reinforcement Learning Problems

Markov Decision Processes (MDP):
Formal description of a world with states, actions, rewards, etc...

An example MDP:

At each time:
- Player can stay or quit
- If quit: get \$10 & game ends
- If stay:
  - prob. 1/3, get \$0 & end
  - prob 2/3, get \$6 & continues



Formal ingredients of MDP

- Set of States (e.g. possible positions of robot)
- Starting state s_{start} (or distribution over states)
- Actions(s): Possible actions at state s
- T(s, a, s'): Probability of going from state s to state s' after taking action a ("transition") (e.g. T(start, stay, End) = 1/3)
- Reward(s, a, s'): Reward when going from s to s' by taking action a (unknown in RL)
- IsEnd(s): Is this an end state

Given an MDP, what is optimal agent behavior?

Policy: A strategy that agent can use
Formally: mapping $\pi(s) \rightarrow$ action $\in$ Actions(s)
(s: current state; action: chosen action)

The **Value** $V_\pi(s)$ for policy π in state s is
Expected (discounted) sum of rewards starting at s, running π

Discounting: Future rewards are less valuable than rewards now
- At any timestep you could die

We introduce a discount factor $\gamma \in [0, 1]$
(probability of survival at each timestep)
e.g. $\gamma = .99$

If we get a sequence of rewards $r_1, r_2, r_3, \ldots$
Discounted sum of rewards = $r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots$

The optimal value $V_{opt}(s)$ is maximum possible value at state s for any policy

V_{opt} is characterized by recursive formulas:

$$V_{opt}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} Q_{opt}(s,a) & \text{else} \end{cases}$$

Expected optimal value after taking action a in state s

$$Q_{opt}(s,a) = \sum_{s'} T(s,a,s') \left[\text{Reward}(s,a,s') + \gamma V_{opt}(s') \right]$$

"Q-value"

Prob of transitioning to s'

Reward now (no discount)

discounted future rewards starting at s'

optimal policy: $\pi^*(s) = \underset{a \in \text{Actions}(s)}{\text{argmax}} Q_{opt}(s,a)$

Takeaway: If we can estimate $Q_{opt}(s,a)$ for all s, a we can immediately find the optimal policy

---

Announcements

- HW3 due today
- HW4 out Thurs
- Midterms reports back
- Thurs: Final project expectations

Reinforcement Learning

- Believe the world is some MDP
- Don't know $T(s,a,s')$ or $\text{Reward}(s,a,s')$

A simple RL problem:

At each time:

Agent stays or quits

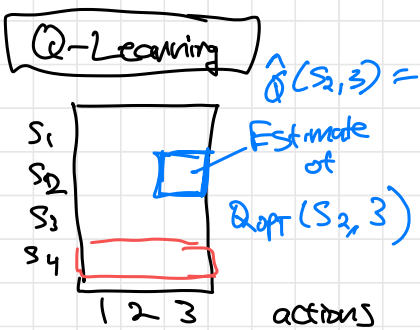
- If stay: Get some reward & new state
- If quit: Get some reward & new state

RL: Agent has to try many actions in many states to learn what to do

For episode = 1, 2, 3, ...
  $s_1 \leftarrow s_{start}$ // or sample from distribution over start state
  for $t = 1, 2, \ldots$
    - Agent chooses action $a_t = \pi_{act}(s_t)$
      (π_{act}: policy we act with during learning)
    - Agent receives:
      - Reward r_t
      - New state s_{t+1}
    - Update agent's parameters // How?

Do the Q-learning update →

Today's RL Algorithm: Q-Learning: Directly learn $Q_{opt}(s,a)$



"parameters" are table of size #Actions × #States

called $Q(s,a)$

$$Q_{opt}(s,a) = \sum_{s'} T(s,a,s') \cdot [\text{Reward}(s,a,s') + \gamma V(s')]$$

where $V_{opt}(s') = \begin{cases} 0 & \text{if IsEnd}(s') \\ \max_{a \in \text{Actions}(s')} Q_{opt}(s',a) \end{cases}$

In RL we have data of the form:

$s_1, a_1, r_1, s_2, a_2, r_2, s_3, \ldots$

start | take action | get reward | new state | take action | get reward | new state ...

1 "example" and "example"

Q-learning: Every time we see (s, a, r, s'):

"Nudge" $\hat{Q}(s,a)$ based on this observation

$$\hat{Q}(s,a) \leftarrow (1-\eta)\hat{Q}(s,a) + \eta\left(r + \gamma \hat{V}(s')\right)$$

- $\hat{Q}(s,a)$: our estimate of $Q_{opt}(s,a)$
- η: learning rate (e.g. 0.1)
- r: observed reward
- $\hat{V}(s')$: observed next state
- $r + \gamma \hat{V}(s')$: one "sample" for $Q_{opt}(s,a)$

where $\hat{V}(s') = 0$ if IsEnd(s'), else $\max_{a \in Actions(s)} \hat{Q}(s',a)$

Finally: what π_{act} to act with?

Obvious candidate: $\pi(s) = \text{argmax}_{a \in Actions(s)} \hat{Q}(s,a)$ — Pure exploitation strategy

RL requires exploration of actions & states

Solution: ε-Greedy

Policy during learning π_{Act} = {with prob $1-\varepsilon$, do $\text{argmax}_a \hat{Q}(s,a)$; with prob ε, do random action in Actions(s)}

- Explores different actions
- New actions land us in new states

Full picture:

- Training: Do Q-Learning, use ε-greedy with $\varepsilon = 0.1$
  Balances exploration & exploitation
- Testing: Act with $\varepsilon = 0$