Problem Solving with AI Techniques
Reinforcement Learning

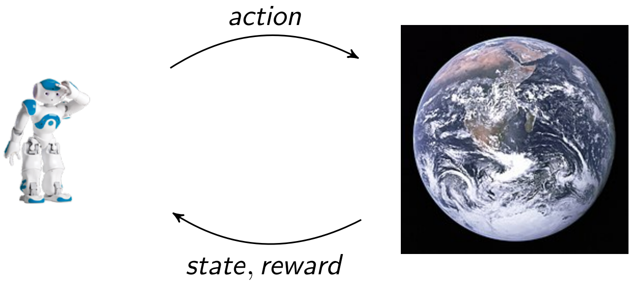
Paul Weng

UM-SJTU Joint Institute

VE593, Fall 2018

What is RL?

- General framework for learning from interactions

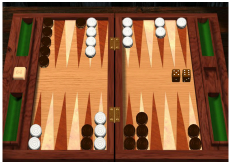


- Inspired by animal/human learning
  - dopamine-based learning
  - animal training
- MDP, but unknown model
- Contextual MAB with action-independent transition

Applications

- Video games, e.g., breakout

- Board game, e.g.,
  Backgammon, Go, Chess, Shogi



- Robotic control

Types of Problems

- Infinite horizon
  - Ergodic MDP
- Repeated problems:
  - Finite horizon: episodic problem
  - Goal oriented problem

Two families of approches:

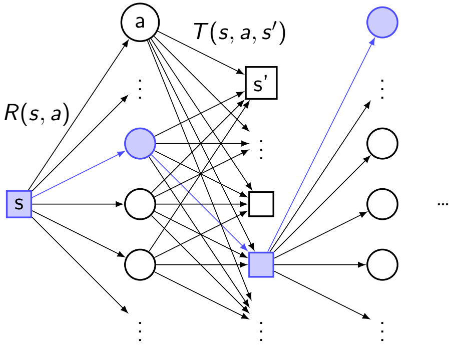
- Model-based RL
  - Indirect approach
- Model-free RL
  - Value-based approaches
  - Policy search

Model-based RL

- Learn model first
  - Estimate $T(s, a, \cdot)$ for all s, a
  - Estimate $R(s, a)$ for all s, a
- Find optimal policy using learned model
- Issue: Hard to learn model (in terms of sample, space requirement...)
- As learned model is an approximation, optimal policy in learned model is probably not optimal in true environment
- Maybe a good approach for multi-task RL

Monte Carlo Sampling: Principle

- Goal: $\max_\pi \mathbb{E}_\pi[\sum_t \gamma^t R(S_t, A_t) \mid S_0 = s]$
- Monte Carlo approach: Sample many trajectories



- MCTS in expectimax tree

Monte Carlo Sampling: Incremental Estimation

- Assume we can sample $n+1$ times from $T(s, \pi(s), \cdot)$

$$
\begin{aligned}
\hat{v}^{\pi}_{n+1}(s) &= \frac{v_1 + \ldots v_{n+1}}{n+1} \\
&= \frac{n}{n+1}\hat{v}^{\pi}_{n}(s) + \frac{1}{n+1}v_{n+1} \\
&= \hat{v}^{\pi}_{n}(s) + \frac{1}{n+1}(v_{n+1} - \hat{v}^{\pi}_{n}(s)) \\
&\sim \hat{v}^{\pi}_{n}(s) + \alpha(v_{n+1} - \hat{v}^{\pi}_{n}(s))
\end{aligned}
$$

- Property: no bias, but high variance
- Issue: May be inefficient, as it doesn't exploit the problem structure

Boostrapping

- Recall

$$\begin{aligned} v^\pi(s) &= \mathbb{E}_\pi[\sum_t \gamma^t R(S_t, A_t) \mid S_0 = s] \\ &= \mathbb{E}[\sum_t \gamma^t R(S_t, A_t) \mid S_0 = s, \pi(S_t) = A_t, S_{t+1} \sim T(S_t, A_t, \cdot)] \\ &= R(s, \pi(s)) + \gamma \mathbb{E}_{S' \sim T(s,\pi(s),\cdot)}[v^\pi(S')] \\ v^*(s) &= \max_\pi \mathbb{E}_\pi[\sum_t \gamma^t R(S_t, A_t) \mid S_0 = s] \\ &= \max_a R(s, a) + \gamma \mathbb{E}_{S' \sim T(s,a,\cdot)}[v^*(S')] \end{aligned}$$

- Idea: Use current estimates to avoid sampling whole trajectories

$$\hat{v}^\pi(s) \leftarrow R(s, \pi(s)) + \gamma \mathbb{E}_{S' \sim T(s,\pi(s),\cdot)}[\hat{v}^\pi(S')]$$
$$\hat{v}^*(s) \leftarrow \max_a R(s, a) + \gamma \mathbb{E}_{S' \sim T(s,a,\cdot)}[\hat{v}^*(S')]$$

Temporal Difference

- Incremental updates:

$$\hat{v}^{\pi}_{n+1}(s) = \hat{v}^{\pi}_{n}(s) + \alpha\big(r_{n+1} + \gamma \hat{v}^{\pi}_{n}(s') - \hat{v}^{\pi}_{n}(s)\big)$$

- TD error: $r_{n+1} + \gamma \hat{v}^{\pi}_{n}(s') - \hat{v}^{\pi}_{n}(s)$

- TD(0): learns the value function of a policy
- can be extended to the Q function:

$$\hat{Q}^{\pi}_{n+1}(s,a) = \hat{Q}^{\pi}_{n}(s,a) + \alpha\big(r_{n+1} + \gamma \hat{Q}^{\pi}_{n}(s',a') - \hat{Q}^{\pi}_{n}(s,a)\big)$$

- Why use the Q function?
- Property: biased, but low variance

Sarsa: Algorithm

- Idea: Estimate $Q^\pi(s, a)$ for all s, a and improve π

```
1  initialize $Q(s, a)$
2  for each episode do
3      initialize $s$
4      choose $a$ in $s$ using $Q$ and possibly some randomness
5      repeat
6          observe $s', r$ after applying $a$ in $s$
7          choose $a'$ in $s'$ using $Q$ and possibly some randomness
8          $Q(s, a) \leftarrow Q(s, a) + \alpha\big(r + \gamma Q(s', a') - Q(s, a)\big)$
9          $s \leftarrow s'$; $a \leftarrow a'$
10     until s is terminal;
```

Exploration Policy

- Greedy policy: $\pi(s) = \arg\max_a Q(s, a)$
- ε-greedy policy: choose $\arg\max_a Q(s, a)$ with probability $1 - \varepsilon$ and a random action otherwise
- softmax policy: choose a with probability $\frac{\exp \frac{Q(s,a)}{\tau}}{\sum_{a'} \exp \frac{Q(s,a)}{\tau}}$

Sarsa: Convergence

- Convergence to optimal policy and optimal Q-function if:
  - π converges to the greedy policy
  - But, all pairs s, a are visited infinitely often

- Example: choose π as $\frac{1}{t}$-greedy policy

Q-learning: Algorithm

- Idea: Estimate $Q^*(s, a)$ for all s, a

```
initialize $Q(s, a)$
**for** *each episode* **do**
    initialize $s$
    **repeat**
        choose $a$ in $s$ using $Q$ and possibly some randomness
        observe $s', r$ after applying $a$ in $s$
        $Q(s, a) \leftarrow Q(s, a) + \alpha\big(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big)$
        $s \leftarrow s'$
    **until** *$s$ is terminal*;
```

Q-learning: Convergence

- Convergence to optimal policy and optimal Q-function if:
  - all pairs s, a are visited infinitely often
  - learning rates satisfy: $\forall s, a$

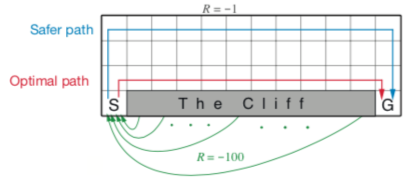
$$\sum_t \alpha_t(s, a) = \infty \qquad \sum_t \alpha_t^2(s, a) < \infty$$

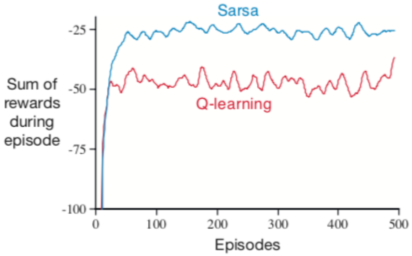
- In practice, $\forall s, a, t, \alpha_t(s, a) = \alpha$, which helps in non-stationary environments

Sarsa vs Q-learning

- Behavior policy: policy used in environment
- Target policy: policy whose value (or Q) function is learned
- Off-policy vs on-policy algorithms
- Online performance is preferred $\Rightarrow$ Sarsa
- Learning optimal Q-function is important, no need to control behavior policy $\Rightarrow$ Q-learning

Example: Cliff Walking





from Sutton and Barto

Issues

- Limitation: These methods assume finite MDPs
- Curse of dimensionality
- Need of compact representations
- We'll focus on methods based on function approximation

Linear Approximation for Value Functions

- Assume we want to approximate $v^\pi : \mathcal{S} \to \mathcal{A}$
- Basis Functions: $\boldsymbol{\phi} = (\phi_1, \phi_2, \dots, \phi_K)^\intercal$ where $\phi_k : \mathcal{S} \to \mathbb{R}$
- Goal: We want to find $\boldsymbol{w}$ such that $v_{\boldsymbol{w}} = \boldsymbol{w}^\intercal \phi$ is close to v^π
- e.g., $\arg\min_{\boldsymbol{w}} \mathbb{E}_\mu[(v^\pi(S) - v_{\boldsymbol{w}}(S))^2 \mid S \sim \mu]$
- Idea: Optimize its empirical version $\frac{1}{T}\sum_t (v_{\boldsymbol{w}}(s_t) - v^\pi(s_t))^2$ with stochastic gradient descent:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \nabla_{\boldsymbol{w}} v_{\boldsymbol{w}}(s_t)\big(v^\pi(S) - v_{\boldsymbol{w}}(S)\big)$$

- Issue: v^π not known $\Rightarrow$ boostrapping
- Linear TD(0): $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \boldsymbol{\phi}(s_t)\big(r_t + \gamma \boldsymbol{w}^\intercal \phi(s_{t+1}) - \boldsymbol{w}^\intercal \phi(s_t)\big)$

Linear Approximation for Q-Functions

- Previous approach can be extended to Q-functions
- Basis Functions: $\boldsymbol{\phi} = (\phi_1, \phi_2, \dots, \phi_K)^\intercal$ where $\phi_k : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- Linear Sarsa:
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \boldsymbol{\phi}(s_t, a_t)\big(r_t + \gamma \boldsymbol{w}^\intercal \phi(s_{t+1}, a_{t+1}) - \boldsymbol{w}^\intercal \phi(s_t, a_t)\big)$
- Linear Q-learning:
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \boldsymbol{\phi}(s_t, a_t)\big(r_t + \gamma \max_a \boldsymbol{w}^\intercal \phi(s_{t+1}, a) - \boldsymbol{w}^\intercal \phi(s_t, a_t)\big)$

Discussions

- Strictly speaking, previous method is not a gradient method, because of unknown target
- Other formulations of objective functions are possible, see Sutton and Barto's book
- Issue: How to define the basis functions in ϕ?
- Previous approach could be used with non-linear function approximators, however it may be unstable
- What if the action space is continuous?

Policy Search

- Issue: Previous methods do not work with continuous action space
- Idea: Try to find a good (possibly stochastic) policy directly
- Possible methods: Local (stochastic) search, metaheuristics...
- Policy gradient: gradient descent/ascent-based method
- All those methods may converge to a local optimum

Policy Gradient

- Assumptions: Differentiable parametrized policy $\pi_\theta(s, a)$
- Objective function:

$$J(\theta) = \mathbb{E}[\sum_t \gamma^t R(S_t, A_t) | S_0 \sim \mu, A_t \sim \pi_\theta(S_t, \cdot), S_{t+1} \sim T(S_t, A_t, \cdot)]$$

- Gradient update: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
- Likelihood trick: $\nabla_\theta \pi_\theta(s, a) = \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)$
- Policy Gradient Theorem: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$
- Reinforce: Approximate the expectations by sampling

Examples of Parametrized Policy

- Softmax policy
  - Define feature functions $\boldsymbol{\phi}(s, a) = (\phi_1(s, a), \dots, \phi_K(s, a))$
  - $\pi_{\boldsymbol{\theta}}(s, a) \propto e^{\boldsymbol{\phi}(s,a)^\intercal \boldsymbol{\theta}}$
  - $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s, a) = \boldsymbol{\phi}(s, a) - \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\boldsymbol{\phi}(s, \cdot)]$

- Gaussian policy
  - Define feature functions $\boldsymbol{\phi}(s) = (\phi_1(s), \dots, \phi_K(s))$
  - $\pi_{\boldsymbol{\theta}}(s, \cdot) = \mathcal{N}(\boldsymbol{\phi}(s)^\intercal \boldsymbol{\theta}, \sigma^2)$
  - $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s, a) = \frac{(a - \boldsymbol{\phi}(s)^\intercal \boldsymbol{\theta})\boldsymbol{\phi}(s)}{\sigma^2}$