Active Reinforcement Learning

Virginia Tech CS4804

Outline

- Active reinforcement learning
  - Active adaptive dynamic programming
  - Q-learning
  - Policy Search

Passive Learning

- Recordings of agent running fixed policy
- Observe states, rewards, actions
  - Direct utility estimation
  - Adaptive dynamic programming (ADP)
  - Temporal-difference (TD) learning

Problems with Passive Reinforcement Learning

$$\pi(s) = \underset{a}{\operatorname{argmax}} P(s'|s,a) U(s')$$

Exploration

- Naive approach: randomly choose random action
  - shrink probability of random action over time
- Another approach: always act **greedy**, but overestimate rewards for unexplored states

$$U^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} P(s'|s,a)U^+(s'), N(s,a)\right)$$

- Another approach: always act **greedy**, but overestimate rewards for unexplored states

$$U^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} P(s'|s,a)U^+(s'), N(s,a)\right)$$

$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

Active TD-Learning

- Exactly the same as non-active TD learning

$$U(s) \leftarrow U(s) + \alpha(R(s) + \gamma U(s') - U(s))$$

$$U(s) = R(s) + \gamma \mathrm{E}_{s'}[U(s')]$$

- Still need estimates of transition probabilities

Action-Utility Functions

- Combine transition probabilities with utilities

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a} Q(s',a')$$

$$U(s) = \max_{a} Q(s,a)$$

$$U(s) = R(s) + \gamma \max_{a} \sum_{s'} P(s'|s,a) U(s')$$

Q-Learning

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_a Q(s',a')$$

$$U(s) = \max_a Q(s,a)$$

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) U(s')$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

$$U(s) \leftarrow U(s) + \alpha(R(s) + \gamma U(s') - U(s))$$

Approximate Q-Learning

$$\hat{Q}(s,a) := g(s,a,\boldsymbol{\theta}) := \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_d f_d(s,a)$$

$$\theta_i \leftarrow \theta_i + \alpha \left(R(s) + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a) \right) \frac{\partial g}{\partial \theta_i}$$

$$\theta_i \leftarrow \theta_i + \alpha \left(R(s) + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a) \right) f_i(s,a)$$

Loss Minimization for Parameterized Q

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Estimation error: $$\ell(s,s') = \ell\left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

Compute $\nabla_Q \ell(s,s')$ with backpropagation, do stochastic gradient descent.

Policy Search

- Instead, parameterize policy as a condition probability distribution

$$\pi_\theta(a \mid s)$$

- Tune parameters by approximating the **gradient** of policy
- Gradient descent
- If curious, read https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d

Summary

- Active ADP:
  - choose random actions or act greedy and overestimate utility
- Active TD: exactly the same as standard TD
- Q-learning: learn action-reward function directly, via TD learning
- Approximation uses derivative of utility function, easy if linear
- Policy search

Summary

	Direct Utility Estimation	ADP	TD Learning	Q-Learning	Policy Search (Gradient)
Learns p(s' I s, a)	No	Yes, by counting	No	No	No
Learns R(s)	No	Yes	No	No	No
Learns state utility (U)	U^{π}	optimal U	optimal U if active. U^{π} otherwise	No	No
Learns state-action utility (Q)	No	No	No	Yes	No
Learns π	No	Yes	No (need p)	Yes	Yes
Strategy	Average observed utility	Solve Bellman w/ estimated p & R	Adjust U based on change from s to s'	TD learning for Q instead of U	Fancy calculus