인공지능

19차시 : Reinforcement Learning

서울대학교 컴퓨터공학부
담당 교수 : 장병탁
Seoul National University
Byoung-Tak Zhang
Introduction

- How an agent can learn from success and failure, from reward and punishment.

- **Supervised Learning (Previous lectures)**
  - Methods for learning functions, logical theories, and probability models from examples
  - A supervised learning agent needs to be told the correct move for each position it encounters, but such feedback is seldom available.

- **Reinforcement Learning (This lecture)**
  - RL agents learn from feedbacks about what is good and what is bad, not what is correct. How agents can learn what to do in the absence of labeled examples of what to do
  - **Passive** reinforcement learning
    - Fixed policy, learns utility function, (optionally) learns transition model
  - **Active** reinforcement learning
    - Learns policy, learns utility function, (optionally) learns transition model
Motivating Examples: Pole Balancing Control & Games

Figure 21.9 Setup for the problem of balancing a long pole on top of a moving cart. The cart can be jerked left or right by a controller that observes $x$, $\theta$, $\dot{x}$, and $\dot{\theta}$.
Outline (Lecture 19)

19.1 Introduction .................................................. 5
19.2 Passive Reinforcement Learning ............... 7
19.3 Active Reinforcement Learning .................. 17
19.4 Generalization in RL ...................................... 23
19.5 Policy Search .................................................. 26
19.6 Applications of Reinforcement Learning .... 28
Summary ................................................................. 30
Homework ............................................................... 31
1.1 Reinforcement learning (RL)

- Supervised learning agents need to be told what is the correct move for each position it encounters, but such feedback is seldom available.

- Without some feedback about what is good and what is bad, the agent will have no grounds for deciding which move to make.

- Reinforcement learning agents learn from this kind of feedbacks called reward or reinforcement.
  - Positive rewards: pleasure and food
  - Negative rewards: pain and hunger

- Reinforcement learning solves the Markov decision processes (MDPs) without having any prior knowledge about the transition model and the reward function. The goal is to find an optimal policy, i.e. a policy that maximizes the expected total reward, from the observed rewards.
  - Passive learning, active learning, exploration
2) Three designs of the agent

» **Utility-based agent**
  - **Learns a utility function on states**, $U^\pi(s)$, and uses it to select actions that maximize the expected outcome utility.
  - Must have a model $P(s' | s, a)$ of the environment.

» **Q-learning**
  - **Learns an action-utility function** $Q(s, a)$, giving the expected utility of taking a given action in a given state. Compares the expected utilities for its available choices without needing to know their outcomes.
  - **Does not need a model** $P(s' | s, a)$ of the environment.
  - *Q-learning agents cannot look ahead* because they do not know where their actions lead.

» **Reflex agent**
  - **Learns a policy** $\pi(s)$ that maps directly from states to actions.
19.2 Passive Reinforcement Learning (1/10)

1) Passive reinforcement learning

➢ Task: To learn the utilities of states (or state-action pairs)
  ▶ Learn how good the policy is

➢ The agent’s policy $\pi$ is fixed, and estimate $U^\pi(s)$

➢ In state $s$, it always executes the action $\pi(s)$

➢ Similar to the policy evaluation task (part of the policy iteration)
  ▶ But passive learning agent does not know (not given)
    ➢ Transition model $P(s' | s, a)$: Specifies the probability of reaching state $s'$ from state $s$ after doing action $a$
    ➢ Reward function $R(s)$: Specifies the reward for each state
1) Passive reinforcement learning

- The agent executes a set of trials in the environment using its policy ($\pi$).
- In each trial, the agent starts in state (1,1) and experiences a sequence of state transitions until it reaches one of the terminal state (4,2) or (4,3).

Figure 21.1 (a) A policy $\pi$ for the 4 x 3 world; this policy happens to be optimal with rewards of $R(s) = -0.04$ in the nonterminal states and no discounting. (b) The utilities of the states in the 4 x 3 world, given policy $\pi$. 

$$(1, 1) \to (1, 2) \to (1, 3) \to (2, 3) \to (3, 3) \to (4, 3) +1$$

$$(1, 1) \to (1, 2) \to (1, 3) \to (2, 3) \to (3, 3) \to (3, 4)$$

$$(1, 1) \to (2, 1) \to (3, 1) \to (3, 2) \to (4, 2)$$
2) Direct utility estimation

The idea is that the utility of a state is the expected total reward from that state onward (called the expected reward-to-go). Each trial provides a sample of this quantity for each state visited:

- (1, 1) $\Rightarrow$ 0.72
- (1, 2) $\Rightarrow$ 0.76, 0.84
- (1, 3) $\Rightarrow$ 0.80, 0.88

An instance of supervised learning where each example has the state as input and the observed reward-to-go as output.

Direct utility estimation, however, misses important source of information, i.e. the utilities of states are not independent.

The utility of each state equals its own reward plus the expected utility of its successor states.

The utility values obey the Bellman equations for a fixed policy:

$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) U^\pi(s')$$
3) Adaptive dynamic programming (ADP)

- ADP agent takes advantage of the constraints among the utilities of states by learning the transition model that connects them and solving the corresponding MDP using a dynamic programming method.
  - For passive learning agent, plugging the learned transition model $P(s'|s, \pi(s))$ and observed rewards $R(s)$ into Bellman equations to calculate the utilities of the states.
  - As Bellman equations are linear, we can adopt the modified policy iteration using a simplified value iteration process to update the utility estimates after each change to the learned model:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$
19.2 Passive Reinforcement Learning (5/10)

3) Adaptive dynamic programming (ADP)

```
function PASSIVE-ADP-AGENT(percept) returns an action
    inputs: percept, a percept indicating the current state \( s' \) and reward signal \( r' \)
    persistent: \( \pi \), a fixed policy
        \( mdp \), an MDP with model \( P \), rewards \( R \), discount \( \gamma \)
        \( U \), a table of utilities, initially empty
        \( N_{sa} \), a table of frequencies for state–action pairs, initially zero
        \( N_{s'|sa} \), a table of outcome frequencies given state–action pairs, initially zero
        \( s, a \), the previous state and action, initially null

    if \( s' \) is new then \( U[s'] \leftarrow r' \); \( R[s'] \leftarrow r' \)
    if \( s \) is not null then
        increment \( N_{sa}[s, a] \) and \( N_{s'|sa}[s', s, a] \)
        for each \( t \) such that \( N_{s'|sa}[t, s, a] \) is nonzero do
            \( P(t | s, a) \leftarrow N_{s'|sa}[t, s, a] / N_{sa}[s, a] \)
        \( U \leftarrow \text{POLICY-EVALUATION}(\pi, U, mdp) \)
    if \( s' \).TERMINAL? \( \text{then} \ s, a \leftarrow \text{null else} \ s, a \leftarrow s', \pi[s'] \)
    return \( a \)

\[
U_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U_i(s')
\]
```

Figure 21.2 A passive reinforcement learning agent based on adaptive dynamic programming. The POLICY-EVALUATION function solves the fixed-policy Bellman equations, as described on page 657.
3) Adaptive dynamic programming (ADP)

**Figure 21.3** The passive ADP learning curves for the $4 \times 3$ world, given the optimal policy shown in Figure 21.1. (a) The utility estimates for a selected subset of states, as a function of the number of trials. Notice the large changes occurring around the 78th trial—this is the first time that the agent falls into the $-1$ terminal state at $(4,2)$. (b) The root-mean-square error (see Appendix A) in the estimate for $U(1,1)$, averaged over 20 runs of 100 trials each.
3) Adaptive dynamic programming (ADP)

The algorithm PASSIVE–ADP–AGENT is using maximum-likelihood estimation to learn the transition model. Moreover, by choosing a policy based solely on the estimated model it is acting as if the model were correct. This is not necessarily a good idea.

Two mathematical approaches to choose policy

- **Bayesian reinforcement learning**: Assumes a prior probability $P(h)$ for each hypothesis $h$ about what the true model is; the posterior probability $P(h|e)$ is obtained in the usual way by Bayes’ rule given the observations to date

$$\pi^* = \arg \max_{\pi} \sum_h P(h | e) u_h^{\pi}$$

- **Robust control theory**: Allows for a set of possible models $\mathcal{H}$ and defines an optimal robust policy as one that gives the best outcome in the worst case over $\mathcal{H}$:

$$\pi^* = \arg \max_{\pi} \min_h u_h^{\pi}$$
19.2 Passive Reinforcement Learning (8/10)

4) Temporal difference RL (TD learning)

- ADP makes as many as it needs to restore consistency between the utility estimates $U$ and environment model $P$

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

- TD makes a single adjustment per observed transition. For each transition from $s$ to $s'$, update:

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

  - $\alpha$: learning rate, $\gamma$: discount rate

- Does not need a transition model, i.e. $\sum_{s'} P(s'|s, a)$, to perform its updates
4) Temporal difference RL (TD learning)

```plaintext
function PASSIVE-TD-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
  persistent: $\pi$, a fixed policy
    $U$, a table of utilities, initially empty
    $N_s$, a table of frequencies for states, initially zero
  $s, a, r, the previous state, action, and reward, initially null

  if $s'$ is new then $U[s'] \leftarrow r'$
  if $s$ is not null then
    increment $N_s[s]$
    $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
  if $s'.TERMINAL?$ then $s, a, r \leftarrow null$ else $s, a, r \leftarrow s', \pi[s'], r'$
  return $a$
```

**Figure 21.4** A passive reinforcement learning agent that learns utility estimates using temporal differences. The step-size function $\alpha(n)$ is chosen to ensure convergence, as described in the text.
4) Temporal difference RL (TD learning)

Figure 21.5  The TD learning curves for the $4 \times 3$ world. (a) The utility estimates for a selected subset of states, as a function of the number of trials. (b) The root-mean-square error in the estimate for $U(1, 1)$, averaged over 20 runs of 500 trials each. Only the first 100 trials are shown to enable comparison with Figure 21.3.
19.3 Active Reinforcement Learning (1/6)

1) Active reinforcement learning

- Passive learning agent has a fixed policy that determines its behavior, but an active agent must decide what actions to take.
- We start with the ADP agent and consider how it must be modified to handle this new freedom.
- The agent will need to learn a complete model with outcome probabilities for all actions, rather than the model for the fixed policy.
- Idea: simple learning mechanism in ADP + a choice of actions
- Two methods:
  - Active TD-learning = Active ADP
  - Q-learning = TD with action-utility value
1) Active reinforcement learning

**Figure 21.6** Performance of a greedy ADP agent that executes the action recommended by the optimal policy for the learned model. (a) RMS error in the utility estimates averaged over the nine nonterminal squares. (b) The suboptimal policy to which the greedy agent converges in this particular sequence of trials.
1) Active reinforcement learning

- The experimental results in the previous slide shows that greedy agents seldom converges to the optimal policy.
- Tradeoff between exploitation and exploration
  - **Exploitation**: to maximize its reward as reflected in the current utility estimates
  - **Exploration**: to maximize its long-term well-being
  - Pure exploitation risks getting stuck in a rut. Pure exploration to improve one’s knowledge is of no use if one never puts that knowledge into practice.
- There are two solutions,
  - **ε-greedy**: With fixed probability perform a random action
  - **Exploration**: assign a higher utility estimate to unexplored state-action pairs

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

**Exploration function** \( f(u, n) \)
- If \( n < N_e \), \( R^+ \) (optimistic estimate), \( u \) otherwise
1) Active reinforcement learning

Figure 21.7 Performance of the exploratory ADP agent. using $R^+ = 2$ and $N_e = 5$. (a) Utility estimates for selected states over time. (b) The RMS error in utility values and the associated policy loss.
2) Q-Learning (Learning an action–utility function)

Q-learning agent learns an action–utility function, or Q-function, giving the expected utility of taking a given action in a given state. Relation of Q values to utilities

\[ U(s) = \max_a Q(s, a) \]

The constraint equation that must hold at equilibrium when Q-values are correct:

\[ Q(s, a) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a') \]

This does, however, require that a model, \( P(s'|s, a) \), also be learned. The TD–approach, on the other hand, requires no model of state transition (model-free method). All it needs are the Q-values.

**TD Q-learning or Q-learning:**

\[ Q(s, a) \leftarrow Q(s, a) + \alpha (R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)) \quad : \text{off-policy} \]

**Cf. SARSA (State–Action–Reward–State–Action):**

\[ Q(s, a) \leftarrow Q(s, a) + \alpha (R(s) + \gamma Q(s', a') - Q(s, a)) \quad : \text{on-policy} \]
2) Q-Learning (Learning an action–utility function)

function Q-LEARNING-AGENT(percept) returns an action

inputs: percept, a percept indicating the current state \( s' \) and reward signal \( r' \)
persistent: \( Q \), a table of action values indexed by state and action, initially zero
\( N_{sa} \), a table of frequencies for state–action pairs, initially zero
\( s, a, r \), the previous state, action, and reward, initially null

if TERMINAL?(s) then \( Q[s, None] \leftarrow r' \)
if \( s \) is not null then
    increment \( N_{sa}[s, a] \)
    \( Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a]) \)
    \( s, a, r \leftarrow s', \arg\max_{a'} f(Q[s', a'], N_{sa}[s', a']), r' \)
return \( a \)

Figure 21.8 An exploratory Q-learning agent. It is an active learner that learns the value \( Q(s, a) \) of each action in each situation. It uses the same exploration function \( f \) as the exploratory ADP agent, but avoids having to learn the transition model because the Q-value of a state can be related directly to those of its neighbors.
So far, we have assumed that the utility functions $U(s)$ and $Q(s, a)$ are represented by lookup table. However, this works only for small problems.

We can generalize the utility functions by using function approximation techniques (parameterized representations of utility functions), such as neural networks or deep learning.

1) Using function approximation for utility

The error in online learning is defined as the squared difference of the predicted total reward $\hat{U}_\theta(s)$ and the actual total $u_j(s)$:

$$E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2/2$$

$u_j(s)$: the observed total reward from state $s$ onward in the $j$th trial

$\theta$: parameters

$\frac{\partial E_j}{\partial \theta_i}$: the rate of change of the error w.r.t. each parameter $\theta_i$

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha (u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

Widrow-Hoff rule
Delta rule
We can apply the (parameterized) function approximation idea to TD and Q learning

2) Reinforcement learning with function approximation

- **TD learning**

  \[
  \theta_i \leftarrow \theta_i + \alpha(R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}
  \]

  \(\theta\): parameters

- **Q-learning**

  \[
  \theta_i \leftarrow \theta_i + \alpha(R(s) + \gamma \max_{a'} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)) \frac{\partial \hat{Q}_\theta(s, a)}{\partial \theta_i}
  \]
3) Deep Q-Network (DQN)

- AlphaGo used deep learning (deep networks) for function approximation of Q functions
- Q-learning: \( Q(s_t, a_t) = Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \)
- DQN loss: \[
L = \frac{1}{2} [r + \gamma \max_a Q(s', a) - Q(s, a)]^2
\]
19.5 Policy Search (1/2)

- We can use **parameterized representation of** $\pi$ **that have far fewer parameters than there are states in the state space.**

1) **Parameterized policy representation**

- **1. Policy representation** in terms of $Q$ functions
  
  \[ \pi(s) = \max_a \hat{Q}_\theta(s, a) \]

- **2. Stochastic policy representation** with softmax
  
  \[ \pi_\theta(s, a) = \frac{e^{\hat{Q}_\theta(s, a)}}{\sum_{a'} e^{\hat{Q}_\theta(s, a')}} \]

- For a **stochastic policy** $\pi_\theta(s, a)$
  
  - $\rho(\theta)$ **policy value**: expected reward–to–go where $\pi_\theta$ is executed
  
  - $\nabla_\theta \rho(\theta)$ **policy gradient vector**
2) Policy gradient

- Derive this estimate $\nabla_\theta \rho(\theta)$ for the simple case of a nonsequential environment in which the reward $R(a)$ is obtained immediately after action $a$ in the start state $s_0$.

- **Policy value** $\rho(\theta)$ is just the expected value of the reward and the **policy gradient** $\nabla_\theta \rho(\theta)$ is:

\[
\nabla_\theta \rho(\theta) = \nabla_\theta \sum_a \pi_\theta(s_0, a) R(a) = \sum_a (\nabla_\theta \pi_\theta(s_0, a)) R(a)
\]

- **Monte Carlo approximation**: The summation can be approximated by samples generated from $\pi_\theta(s_0, a)$. With $N$ trials in all and the action taken on the $j$th trial is $a_j$:

\[
\nabla_\theta \rho(\theta) = \sum_a \pi_\theta(s_0, a) \cdot \frac{(\nabla_\theta \pi_\theta(s_0, a)) R(a)}{\pi_\theta(s_0, a)} \approx \frac{1}{N} \sum_{j=1}^{N} \frac{(\nabla_\theta \pi_\theta(s_0, a_j)) R(a_j)}{\pi_\theta(s_0, a_j)}
\]

- **For the sequential case** (Reinforce algorithm)

\[
\nabla_\theta \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} \frac{(\nabla_\theta \pi_\theta(s, a_j)) R_j(s)}{\pi_\theta(s, a_j)}
\]

$R_j(s)$: total reward received from state $s$ onwards in $j$th trial
19.6 Applications of Reinforcement Learning (1/2)

1) Applications of reinforcement learning

- Applications to game playing
  - Checker program written by Arthur Samuel (1959, 1967)
  - TD-GAMMON (Backgammon) program by Gerry Tesauro (1992)
  - AlphaGo (Deep Q-Net) program by Google DeepMind (2016)

- Applications to robot control
  - Cart-pole balancing problem (inverted pendulum) carried out by Michie and Chambers (1968)
  - Helicopter flight using policy search by Bagnell and Schneider (2001)
  - Deep RL for robotics and self-driving cars (2016~)
1) Applications of reinforcement learning

Figure 21.9  Setup for the problem of balancing a long pole on top of a moving cart. The cart can be jerked left or right by a controller that observes $x$, $\theta$, $\dot{x}$, and $\dot{\theta}$.

Figure 21.10  Superimposed time-lapse images of an autonomous helicopter performing a very difficult “nose-in circle” maneuver. The helicopter is under the control of a policy developed by the PEGASUS policy-search algorithm. A simulator model was developed by observing the effects of various control manipulations on the real helicopter; then the algorithm was run on the simulator model overnight. A variety of controllers were developed for different maneuvers. In all cases, performance far exceeded that of an expert human pilot using remote control. (Image courtesy of Andrew Ng.)
Direct utility estimation uses the total observed reward-to-go for a given state as direct evidence for learning its utility.

Adaptive dynamic programming (ADP) learns a model and a reward function from observations and then uses value or policy iteration to obtain the utilities or an optimal policy. ADP makes optimal use of the local constraints on utilities of states imposed through the neighborhood structure of the environment.

Temporal-difference (TD) methods update utility estimates to match those of successor states. They can be viewed as simple approximations to the ADP approach that can learn without requiring a transition model. Using a learned model to generate pseudo-experiences can, however, result in faster learning.

Action-utility functions, or Q-functions, can be learned by an ADP approach or a TD approach.

The temporal-difference signal can be used directly to update parameters in representations such as neural networks.

Policy-search methods operate directly on a representation of the policy, attempting to improve it based on observed performance.
Homework

» 21.4 (TD learning)

» 21.10 (Modeling human and animal behavior)
출처

사진


# 2 https://flickr.com/photos/prachatai/25708381781/in/photolist-FbVkDr-UH2u89-FaLbeK-F6L8p9-EP9vBd-ThxG5b-HH7PQt-FzNGdU-Eanobw-PCjLmb-2hcTxW2-2hcSH3K-2hcR1gi-2hcR1ix-eNizcR (Creative Commons License)