CS 440/ECE448 Lecture 21: Reinforcement Learning

Slides by Svetlana Lazebnik, 11/2016
Modified by Mark Hasegawa-Johnson, 4/2018

By Nicolas P. Rougier - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=29327040
Reinforcement learning

• Regular MDP
  • Given:
    • Transition model $P(s' | s, a)$
    • Reward function $R(s)$
  • Find:
    • Policy $\pi(s)$

• Reinforcement learning
  • Transition model and reward function initially unknown
  • Still need to find the right policy
  • “Learn by doing”
Reinforcement learning: Basic scheme

• In each time step:
  • Take some action
  • Observe the outcome of the action: successor state and reward
  • Update some internal representation of the environment and policy
  • If you reach a terminal state, just start over (each pass through the environment is called a trial)

• Why is this called reinforcement learning?
Outline

• Applications of Reinforcement Learning
• Model-Based Reinforcement Learning
  • Estimate $P(s' | s, a)$ and $R(s)$
  • Exploration vs. Exploitation
• Model-Free Reinforcement Learning
  • Q-learning
  • Temporal Difference Learning
  • SARSA
• Function approximation; policy learning
Applications of reinforcement learning

**Spoken Dialog Systems (Litman et al., 2000)**

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GreetS</td>
<td>Welcome to NJFun. Please say an activity name or say 'list activities' for a list of activities I know about.</td>
</tr>
<tr>
<td>GreetU</td>
<td>Welcome to NJFun. How may I help you?</td>
</tr>
<tr>
<td>ReAsk 1 S</td>
<td>I know about amusement parks, aquariums, cruises, historic sites, museums, parks, theaters, wineries, and zoos. Please say an activity name from this list.</td>
</tr>
<tr>
<td>ReAsk 1M</td>
<td>Please tell me the activity type. You can also tell me the location and time.</td>
</tr>
</tbody>
</table>
Applications of reinforcement learning

• Learning a fast gait for Aibos

Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion
Nate Kohl and Peter Stone.
Applications of reinforcement learning

- Stanford autonomous helicopter

Pieter Abbeel et al.
Applications of reinforcement learning

• Playing Atari with deep reinforcement learning

Video

Applications of reinforcement learning

• **End-to-end training of deep visuomotor policies**

Fig. 1: Our method learns visuomotor policies that directly use camera image observations (left) to set motor torques on a PR2 robot (right).

[Video](#)  
Sergey Levine et al., Berkeley
Applications of reinforcement learning

- **Active object localization with deep reinforcement learning**

J. Caicedo and S. Lazebnik, ICCV 2015
Learning to Translate in Real Time with Neural Machine Translation

Graham Neubig, Kyunghyun Cho, Jiatao Gu, Victor O. K. Li

Figure 2: Illustration of the proposed framework: at each step, the NMT environment (left) computes a candidate translation. The recurrent agent (right) will the observation including the candidates and send back decisions—READ or WRITE.
Reinforcement learning strategies

• **Model-based**
  • Learn the model of the MDP (transition probabilities and rewards) and try to solve the MDP concurrently

• **Model-free**
  • Learn how to act without explicitly learning the transition probabilities $P(s' \mid s, a)$
  • **Q-learning**: learn an action-utility function $Q(s,a)$ that tells us the value of doing action $a$ in state $s$
Outline

• Applications of Reinforcement Learning

• Model-Based Reinforcement Learning
  • Estimate $P(s' | s, a)$ and $R(s)$
  • Exploration vs. Exploitation

• Model-Free Reinforcement Learning
  • Q-learning
  • Temporal Difference Learning
  • SARSA

• Function approximation; policy learning
Model-based reinforcement learning

• **Basic idea:** try to learn the model of the MDP (transition probabilities and rewards) and learn how to act (solve the MDP) simultaneously

• **Learning the model:**
  • Keep track of how many times state \( s' \) follows state \( s \) when you take action \( a \) and update the transition probability \( P(s' | s, a) \) according to the relative frequencies
  • Keep track of the rewards \( R(s) \)

• **Learning how to act:**
  • Estimate the utilities \( U(s) \) using Bellman’s equations
  • Choose the action that maximizes expected future utility:

\[
\pi^*(s) = \arg \max_{a \in A(s)} \sum_{s'} P(s'| s, a) U(s')
\]
Model-based reinforcement learning

• Learning how to act:
  • Estimate the utilities $U(s)$ using Bellman’s equations
  • Choose the action that maximizes expected future utility given the model of the environment we’ve experienced through our actions so far:

$$
\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' | s, a) U(s')
$$

• Is there any problem with this “greedy” approach?
Exploration vs. exploitation

• **Exploration**: take a new action with unknown consequences
  • Pros:
    • Get a more accurate model of the environment
    • Discover higher-reward states than the ones found so far
  • Cons:
    • When you’re exploring, you’re not maximizing your utility
    • Something bad might happen

• **Exploitation**: go with the best strategy found so far
  • Pros:
    • Maximize reward as reflected in the current utility estimates
    • Avoid bad stuff
  • Cons:
    • Might also prevent you from discovering the true optimal strategy
Incorporating exploration

• **Idea:** explore more in the beginning, become more and more greedy over time

• Standard (“greedy”) selection of optimal action:

$$a = \arg \max_{a' \in A(s)} \sum_{s'} P(s'|s,a')U(s')$$

• Modified strategy:

$$a = \arg \max_{a' \in A(s)} f \left( \sum_{s'} P(s'|s,a')U(s'), N(s,a') \right)$$

- Exploratory function
- Number of times we’ve taken action a’ in state s

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

• Applications of Reinforcement Learning
• Model-Based Reinforcement Learning
  • Estimate P(s’|s,a) and R(s)
  • Exploration vs. Exploitation
• Model-Free Reinforcement Learning
  • Q-learning
  • Temporal Difference Learning
  • SARSA
• Function approximation; policy learning
Model-free reinforcement learning

• **Idea:** learn how to act without explicitly learning the transition probabilities $P(s' \mid s, a)$

• **Q-learning:** learn an *action-utility function* $Q(s,a)$ that tells us the value of doing action $a$ in state $s$

• Relationship between Q-values and utilities:

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

• Selecting an action: $\pi^*(s) = \arg\max_a Q(s,a)$

• Compare with: $\pi^*(s) = \arg\max_a \sum_{s'} P(s'\mid s, a)U(s')$

  • With Q-values, don’t need to know the transition model to select the next action
TD Q-learning result

Q-VALUES AFTER 1000 EPISODES

Source: Berkeley CS188
Model-free reinforcement learning

• **Q-learning**: learn an action-utility function $Q(s,a)$ that tells us the value of doing action $a$ in state $s$

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

• Equilibrium constraint on Q values:

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

• What is the relationship between this constraint and the Bellman equation?

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')$$
Model-free reinforcement learning

- **Q-learning**: learn an action-utility function $Q(s, a)$ that tells us the value of doing action $a$ in state $s$

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

- Equilibrium constraint on Q values:

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

- Problem: we don’t know (and don’t want to learn) $P(s' | s, a)$
Temporal difference (TD) learning

• Equilibrium constraint on Q values:

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

• Temporal difference (TD) update:

• Pretend that the currently observed transition \((s, a, s')\) is the only possible outcome. Call this “local quality” as \(Q^{local}(s, a)\); it is computed using \(Q(s, a)\).

\[ Q^{local}(s, a) = R(s) + \gamma \max_{a'} Q(s', a') \]

• Then interpolate between \(Q(s, a)\) and \(Q^{local}(s, a)\) to compute \(Q^{new}(s, a)\).

\[ Q^{new}(s, a) = (1-\alpha)Q(s, a) + \alpha Q^{local}(s, a) \]
Temporal difference (TD) learning

• The interpolated form:

\[
Q_{local}^{\text{local}}(s, a) = R(s) + \gamma \max_{a'} Q(s', a') \\
Q^{\text{new}}(s, a) = (1 - \alpha)Q(s, a) + \alpha Q_{local}^{\text{local}}(s, a)
\]

• The temporal-difference form:

\[
Q_{local}^{\text{local}}(s, a) = R(s) + \gamma \max_{a'} Q(s', a') \\
Q^{\text{new}}(s, a) = Q(s, a) + \alpha \left( Q_{local}^{\text{local}}(s, a) - Q(s, a) \right)
\]

• The computationally efficient form (all calculations rolled into one):

\[
Q^{\text{new}}(s, a) = Q(s, a) + \alpha \left( R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)
\]
Temporal difference (TD) learning

• At each time step $t$
  • From current state $s$, select an action $a$:

$$a = \arg \max_{a'} f(Q(s, a'), N(s, a'))$$

  Exploration function
  Number of times we’ve taken action $a'$ from state $s$

• Observe the reward $r$, next state $s'$
• Perform the TD update:

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

$s \leftarrow s'$
Temporal difference (TD) learning

• At each time step t
  • From current state $s$, select an action $a$:
    $$a = \arg \max_{a'} f(Q(s, a'), N(s, a'))$$
    Exploration function  Number of times we’ve taken action $a'$ from state $s$

  • Observe the reward $r$, next state $s'$
  • Perform the TD update:
    $$Q(s, a) \leftarrow Q(s, a) + \alpha (R(s) + \gamma \max_a Q(s', a') - Q(s, a))$$
    $$s \leftarrow s'$$
Temporal difference (TD) learning

- At each time step $t$
  - From current state $s$, select an action $a$:
    \[ a = \arg \max_{a'} f(Q(s, a'), N(s, a')) \]
    - Exploration function
    - Number of times we’ve taken action $a'$ from state $s$
  - Observe the reward $r$, next state $s'$
  - Perform the TD update:
    \[ Q(s, a) \leftarrow Q(s, a) + \alpha (R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)) \]
    \[ s \leftarrow s' \]
    That’s not necessarily the action we will take next time...
SARSA: State-Action-Reward-State-Action

- Initialize: choose an initial state $s$, initial action $a$
- At each time step $t$
  - Observe the reward $r$, next state $s'$
  - From next state $s'$, select next action $a'$:
    \[ a' = \text{arg max}_{a'} f (Q(s', a'), N(s', a')) \]
  - Perform the TD update:
    \[ Q(s, a) \leftarrow Q(s, a) + \alpha (R(s) + \gamma Q(s', a') - Q(s, a)) \]
    \[ s \leftarrow s' \]

That is the action we will take next time...
Outline

• Applications of Reinforcement Learning
• Model-Based Reinforcement Learning
  • Estimate $P(s'|s,a)$ and $R(s)$
  • Exploration vs. Exploitation
• Model-Free Reinforcement Learning
  • Q-learning
  • Temporal Difference Learning
• Function approximation; policy learning
Function approximation

- So far, we’ve assumed a lookup table representation for utility function $U(s)$ or action-utility function $Q(s,a)$
- But what if the state space is really large or continuous?
- Alternative idea: approximate the utility function, e.g., as a weighted linear combination of features:

$$U(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots w_n f_n(s)$$

- RL algorithms can be modified to estimate these weights
- More generally, functions can be nonlinear (e.g., neural networks)
- Recall: features for designing evaluation functions in games
- Benefits:
  - Can handle very large state spaces (games), continuous state spaces (robot control)
  - Can generalize to previously unseen states
Other techniques

• **Policy search**: instead of getting the Q-values right, you simply need to get their ordering right
  - Write down the policy as a function of some parameters and adjust the parameters to improve the expected reward

• **Learning from imitation**: instead of an explicit reward function, you have expert demonstrations of the task to learn from