Reinforcement learning (Chapter 21)
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?
Applications of reinforcement learning

• Backgammon


Applications of reinforcement learning

• AlphaGo

https://deepmind.com/research/alphago/
Applications of reinforcement learning

- **Learning a fast gait for Aibos**

  ![Initial gait](image1.jpg) ![Learned gait](image2.jpg)

  **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

V. Mnih et al., *Nature*, February 2015
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

• Object detection

J. Caicedo and S. Lazebnik, Active Object Localization with Deep Reinforcement Learning, ICCV 2015

Video
OpenAI Gym

A toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Go.

Read the launch blog post ›
View documentation ›
View on GitHub ›

https://gym.openai.com/
StarCraft Will Become the Next Big Playground for AI

Artificial intelligence will require key advances in order to play a video game filled with planning, guesswork, and bluffing.

by Will Knight  November 4, 2016

StarCraft, a hugely popular space-fiction-themed strategy computer game, will soon be accessible to advanced AI players. Blizzard Entertainment, the company behind the game, and Google DeepMind, a subsidiary of Alphabet focused on developing general-purpose artificial intelligence, announced the move at a games conference today.

Teaching computers to play StarCraft II expertly would be a significant milestone in artificial-intelligence research. Within the game, players must build bases, mine resources, and attack their opponents’ outposts. Mastering such a complex and sprawling game takes finely honed skills, strategic acumen, and a good dose of cunning. It is visually relatively complex, and players often cannot see what their opponent is up to. It should therefore be an ideal place for computers to make the next big leap in mimicking human intelligence.

https://www.technologyreview.com/s/602796/starcraft-will-become-the-next-big-playground-for-ai/
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 $s$
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:

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

• 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:

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

exploration function

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

\[
f(u, n) = \begin{cases} 
R^+ & \text{if } n < N_e \\
u & \text{otherwise}
\end{cases} \quad \text{(optimistic reward estimate)}
\]
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
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 and adjust the Q values towards the “local equilibrium”

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

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

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

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

- At each time step \( t \)
  - From current state \( s \), select an action \( a \):
    \[
    a = \arg \max_{a'} f(Q(s, a'), N(s, a'))
    \]
  - Get the successor state \( s' \)
  - Perform the TD update:
    \[
    Q(s, a) \leftarrow Q(s, a) + \alpha (R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a))
    \]
    - **Exploration function**
    - **Number of times we’ve taken action \( a' \) from state \( s \)**
    - **Learning rate**
      Should start at 1 and decay as \( O(1/t) \)
      e.g., \( \alpha(t) = 60/(59 + t) \)
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 $s$
  - Get the successor state $s'$
  - Perform the TD update:
    $$Q(s, a) \leftarrow Q(s, a) + \alpha (R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

TD Q-learning result

Q-VALUES AFTER 1000 EPISODES

Source: Berkeley CS188
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