MENG 3065 - MODULE 6

Artificial Intelligence: A Modern Approach
Chapter 22 Reinforcement Learning

HUMBER

WE ARE HUMBER

Outline

- An introduction to Reinforcement Learning
- Sequential Decision Problems
- Learning from Rewards
- Passive Reinforcement Learning
- Active Reinforcement Learning
- Examples of Reinforcement Learning
- Applications of Reinforcement Learning

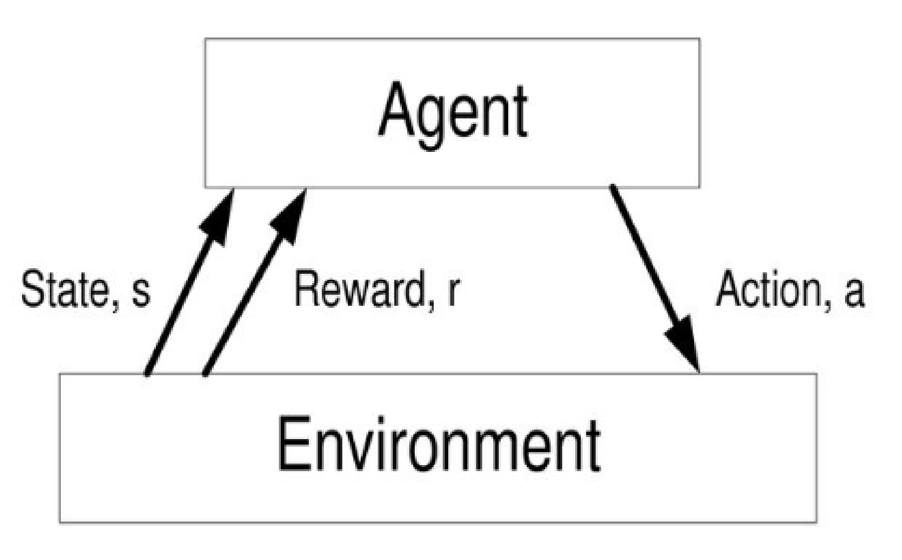
An introduction to Reinforcement Learning

https://youtu.be/JgvyzIkgxF0?si=pa4KFoz7DaVOzmc3

Understanding Reinforcement Learning

- Distinct from Supervised Learning: No direct supervision with "correct" answers
- Scalar Reward Signal: Feedback provided through a numerical reward
- Temporal Considerations: Involves a notion of "time" in terms of steps or moves
- Delayed Feedback: Feedback is not instantaneous but occurs after actions are taken
- Dynamic Impact of Actions: Agent's actions influence subsequent data received

Reinforcement Learning



- Environment
  - Provides the agent with the current state
  - Provides a reward at each time step
- Agent
  - Chooses an action at each time step, given the state
- Reward
  - A reward r is a scalar feedback signal
  - The goal of the agent is to select actions to maximize total future reward.

Key Components of an RL Agent

- Policy:
  - Describes the agent's behavioral strategy or decision-making function
- Value Function:
  - Calculates the expected future rewards associated with a given state
- Model:
  - Represents the agent's internal model of the external environment

Policy

- A policy defines the behavior of an agent by specifying the action it selects at a given state, often denoted by the symbol π.
- The policy function is a mapping that associates states with corresponding actions.
  - deterministic: $a = \pi(s)$
  - stochastic: $\pi(a|s) = P[A_t = a|S_t = s]$

Value Function

- A value function anticipates future rewards under a specific policy.
- A tool to assess the goodness or badness of states.
- Maps the states to their expected discounted returns to help evaluate the quality of states.
- The Q-function extends this concept by mapping pairs (states, actions) to their corresponding expected discounted returns

Model

- Predicts the next actions of the environment
  - predicts the next state given the current state and a specified action
  - predicts the next reward

Sequential Decision Problems

- **Markov decision process (MDP):** a sequential decision problem for a fully observable, stochastic environment

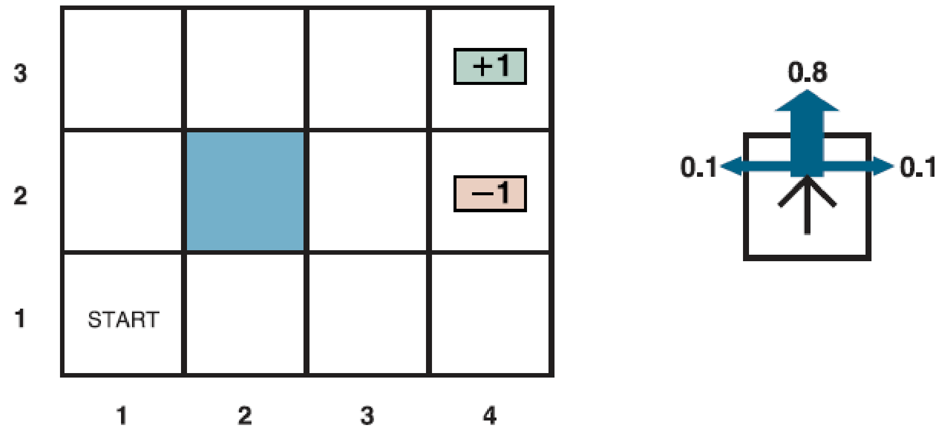
- MDP consists of:
  - a set of states (with an initial state s_0);
  - a set $\mathrm{ACTIONS}(s)$ of actions in each state;
  - a transition model $P(s \mid s, a)$; and
  - a reward function $R(s, a, s\ )$.

Sequential Decision Problems

- MDP solutions usually involve **dynamic programming** simplifying a problem by recursively breaking it into smaller pieces and remembering the optimal solutions to the pieces.

- A solution called **policy**.
  - specify what the agent should do for any state that the agent might reach
  - the quality of a policy is measured by the expected utility of possible environment histories generated
  - **optimal policy**: highest expected utility

Sequential Decision Problems



(a) (b)

a) A simple, stochastic 4x3 environment that presents the agent with a sequential decision problem.

(b) Illustration of the transition model of the environment: the “intended” outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. Transitions into the two terminal states have reward +1 and –1, respectively, and all other transitions have a reward of –0.04.

Sequential Decision Problems

- Utility of a state is the expected reward for the next transition plus the discounted utility of the next state, assuming that the agent chooses the optimal action

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma U(s')] \,.$$

- This is called the **Bellman equation**, after Richard Bellman (1957).
- **Action-utility function**, or **Q-function**: $Q(s, a)$
  - the expected utility of taking a given action in a given state.
  - related to utilities in the obvious way:

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

- The optimal policy can be extracted from the Q-function

$$\pi^*(s) = \operatorname{argmax}_{a} Q(s, a)$$

- The Q-function is in algorithms for solving MDPs

function Q-VALUE(mdp, s, a, U) **returns** a utility value
  return $\sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma U[s']]$

Sequential Decision Problems

	1	2	3	4
3	0.8516	0.9078	0.9578	+1
2	0.8016		0.7003	−1
1	0.7453	0.6953	0.6514	0.4279

- The utilities of the states in the 4X3 world with $\gamma = 1$ and $r = -0.04$ for transitions to nonterminal states.
- In this example: states are defined as (col#, row#)
- The Bellman equation for the state $(1, 1)$ is U(1,1) = max {r + γ * *U(s')*}

$$\begin{aligned}\max\{\, &[0.8(-0.04 + \gamma U(1,2)) + 0.1(-0.04 + \gamma U(2,1)) + 0.1(-0.04 + \gamma U(1,1))], && \text{up}\\ &[0.9(-0.04 + \gamma U(1,1)) + 0.1(-0.04 + \gamma U(1,2))], && \text{left}\\ &[0.9(-0.04 + \gamma U(1,1)) + 0.1(-0.04 + \gamma U(2,1))], && \text{down}\\ &[0.8(-0.04 + \gamma U(2,1)) + 0.1(-0.04 + \gamma U(1,2)) + 0.1(-0.04 + \gamma U(1,1))]\,\} && \text{right}\end{aligned}$$

Learning from Rewards

- Agent interacts with the world and periodically receives rewards (reinforcements)
- Varieties of approaches:
- **Model-based reinforcement learning:** uses a transition model
  - Model may be initially unknown
  - Learns from observing effects of actions
  - Useful for state estimation
  - Learn a utility function $U(s)$,
- **Model-free reinforcement learning:** neither knows nor learns transition model
  - **Action-utility learning:** most common form **Q-learning**, where the agent learns a **Q-function**, or quality-function, $Q(s, a)$, denoting the sum of rewards from state s if action a is taken.
  - **Policy search:** learns a policy $\pi(s)$ that maps directly from states to actions.

Learning from Rewards

Passive Reinforcement Learning

- **Passive learning agent:** agent that learns the utility function U π(s)
  - Expected total discounted reward if policy π is executed beginning in state s
  - does not know the transition model P(s'| s, a),
  - executes a set of trials in the environment using its policy π. starts in state (1,1) and experiences a sequence of state transitions until it reaches one of the terminal states, (4,2) or (4,3).

$$(1,1) \xrightarrow[Up]{\text{-.04}} (1,2) \xrightarrow[Up]{\text{-.04}} (1,3) \xrightarrow[Right]{\text{-.04}} (1,2) \xrightarrow[Up]{\text{-.04}} (1,3) \xrightarrow[Right]{\text{-.04}} (2,3) \xrightarrow[Right]{\text{-.04}} (3,3) \xrightarrow[Right]{\text{+1}} (4,3)$$

$$(1,1) \xrightarrow[Up]{\text{-.04}} (1,2) \xrightarrow[Up]{\text{-.04}} (1,3) \xrightarrow[Right]{\text{-.04}} (2,3) \xrightarrow[Right]{\text{-.04}} (3,3) \xrightarrow[Right]{\text{-.04}} (3,2) \xrightarrow[Up]{\text{-.04}} (3,3) \xrightarrow[Right]{\text{+1}} (4,3)$$

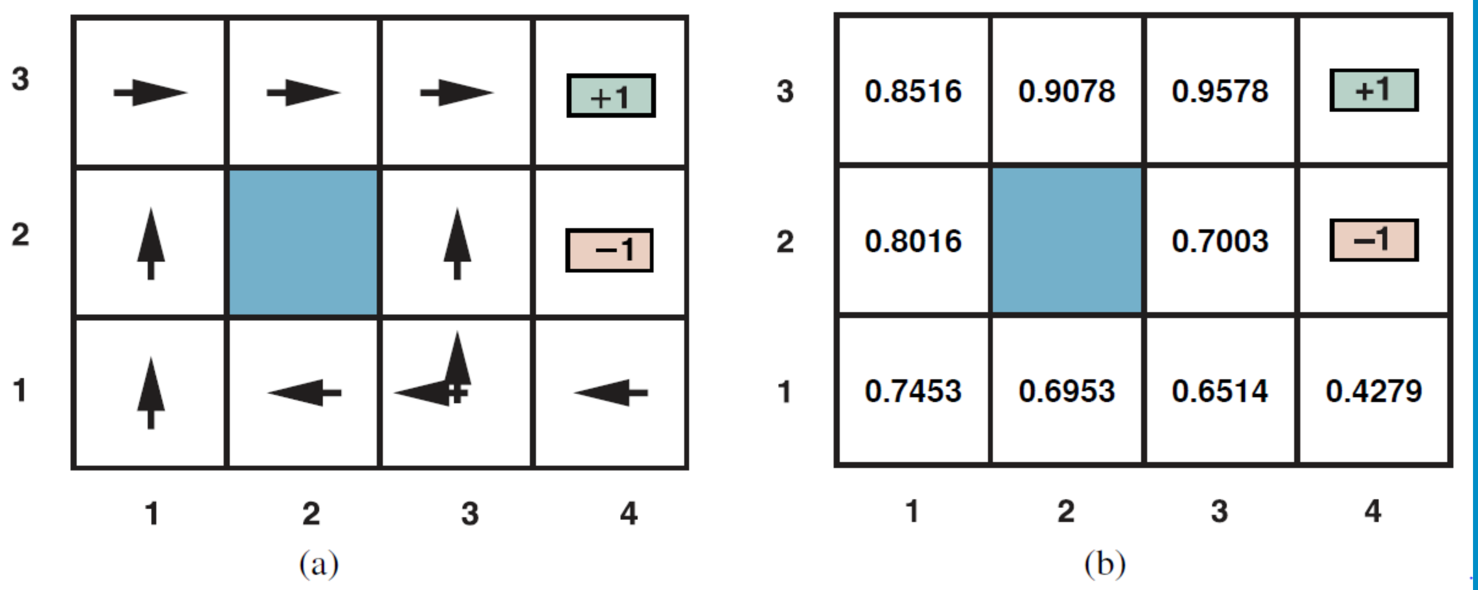
$$(1,1) \xrightarrow[Up]{\text{-.04}} (1,2) \xrightarrow[Up]{\text{-.04}} (1,3) \xrightarrow[Right]{\text{-.04}} (2,3) \xrightarrow[Right]{\text{-.04}} (3,3) \xrightarrow[Right]{\text{-.04}} (3,2) \xrightarrow[Up]{\text{-1}} (4,2)$$

  - Expected utility

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right],$$

Learning from Rewards

Passive Reinforcement Learning



(a) The optimal policies for the stochastic environment with $R(s, a, s^t) = 0.04$ for transitions between nonterminal states. There are two policies because in state (3,1) both *Left* and *Up* are optimal.
(b) The utilities of the states in the 4×3 world, given policy

Learning from Rewards

Direct utility estimation

- utility of a state is defined as the expected total reward from that state onward (reward-to-go)

- at the end of each sequence, the algorithm calculates the observed reward-to-go for each state and updates the estimated utility

- reduced reinforcement learning to a standard supervised learning problem in which each example is a (*state*, *reward-to-go*) pair.

- The utility of a state is determined by the reward and the expected utility of the successor states

$$U_i(s) = \sum_{s'} P(s' \mid s, \pi_i(s)) [R(s, \pi_i(s), s') + \gamma U_i(s')] .$$

Learning from Rewards

Adaptive dynamic programming (ADP)

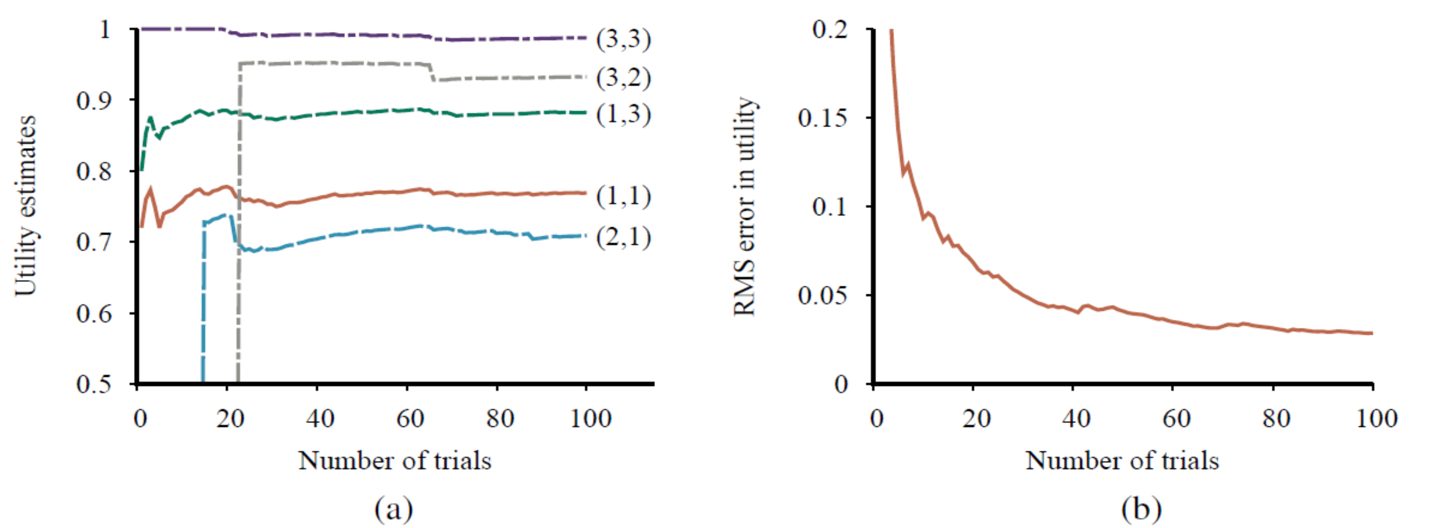
- Agent takes advantage of the constraints among the utilities of states by learning the transition model that connects them

- Solving the corresponding Markov decision process using dynamic programming

```
function PASSIVE-ADP-LEARNER(percept) returns an action
  inputs: percept, a percept indicating the current state s′ and reward signal r
  persistent: π, a fixed policy
              mdp, an MDP with model P, rewards R, actions A, discount γ
              U, a table of utilities for states, initially empty
              N_s′|s,a, a table of outcome count vectors indexed by state and action, initially zero
              s, a, the previous state and action, initially null

  if s′ is new then U[s′] ← 0
  if s is not null then
    increment N_s′|s,a[s, a][s’]
    R[s, a, s′] ← r
    add a to A[s]
    P(· | s, a) ← NORMALIZE(N_s′|s,a[s, a])
    U ← POLICYEVALUATION(π, U, mdp)
    s, a ← s′, π[s′]
    return a
```

Learning from Rewards



The passive ADP learning curves for the 4 x 3 world.

(a) The utility estimates for a selected subset of states, as a function of the number of trials. Notice that it takes 14 and 23 trials respectively before the rarely visited states (2,1) and (3,2) "discover" that they connect to the +1 exit state at (4,3).

(b) The root-mean-square error in the estimate for *U* (1, 1), averaged over 50 runs of 100 trials each.

Learning from Rewards

Temporal-difference learning

- use the observed transitions to adjust the utilities of the observed states so that they agree with the constraint equations.

Temporal-difference (TD) equation

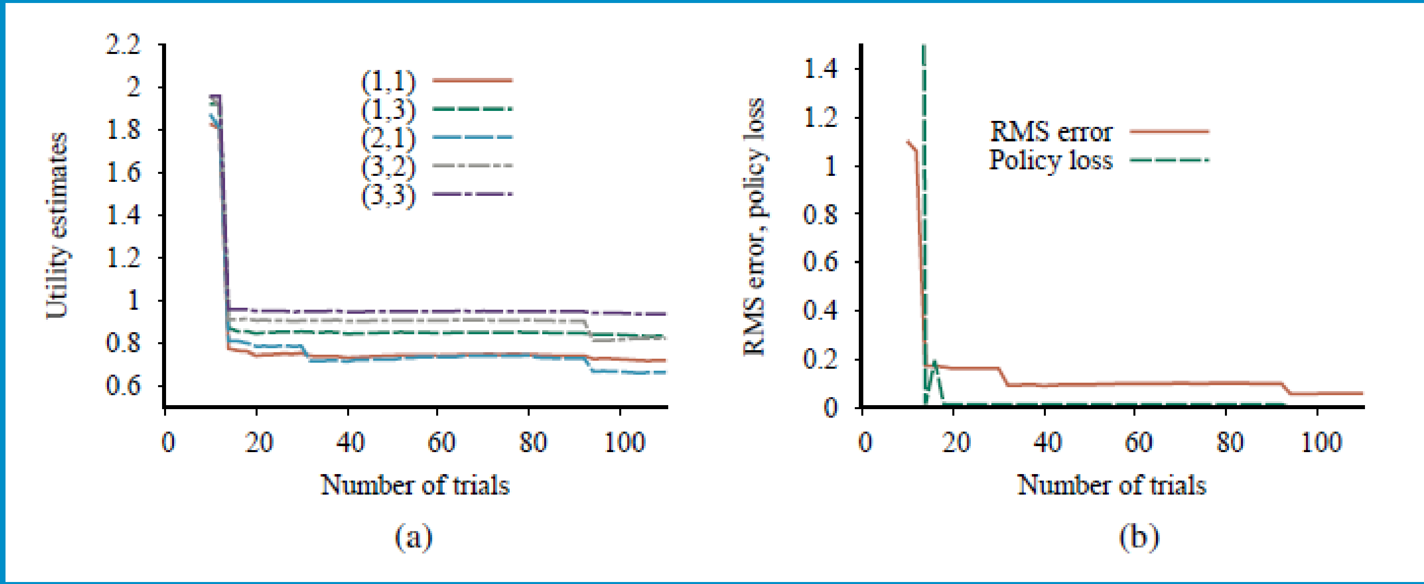
$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha[R(s,\pi(s),s') + \gamma U^{\pi}(s') - U^{\pi}(s)] .$$

- α is the **learning rate** parameter.
- uses the difference in utilities between successive states (and thus successive times)
- adjusting the utility estimates toward the ideal equilibrium that holds locally when the utility estimates are correct
- does not need a transition model to perform its updates.

Learning from Rewards

TD adjusts a state to agree with its *observed* successor	ADP adjusts the state to agree with all of the successors that might occur, weighted by their probabilities
TD makes a single adjustment per observed transition	ADP makes as many as it needs to restore consistency between the utility estimates *U* and the transition model *P*.
For each observed transition, the TD agent can generate a large number of imaginary transitions	can generate more efficient versions of ADP by directly approximating the algorithms for value iteration or policy iteration.

Learning from Rewards



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.

Active Reinforcement Learning

- Passive agent has a fixed policy that determines its behavior
- Active learning agent gets to decide

Exploration

- Refers to the agent's strategy of seeking information about the environment by trying different actions.
- A purely greedy agent sticks to its policy and does not explore alternative actions, potentially missing the optimal route.
    - Also termed as a greedy agent: greedily takes the action that it believes
    - Sometimes this approach pays off and sometimes it does not
    - Overlook that actions do more than provide rewards (Actions provide information in the form of percepts in the resulting states)
- Not greedy in immediate next move but **Greedy in Limit of infinite exploration (GLIE)**
    - GLIE scheme must try each action in each state an unbounded number of times
    - to avoid having a finite probability that an optimal action is missed.

Active Reinforcement Learning

Safe Exploration

- Many actions are **irreversible**
  - no subsequent sequence of actions can restore the state to what it was before the irreversible action was taken
  - Worst case: agent enters **absorbing state** (no actions have any effect/rewards)

- choose a policy that works reasonably well for the whole range of models that have a reasonable chance of being the true model
  - even if the policy happens to be suboptimal for the maximum-likelihood model.

  - **Three mathematical approaches:**

  i) Bayesian reinforcement learning
  ii) Exploration POMDP
  iii) Robust control theory

Active Reinforcement Learning

Temporal-difference Q-learning

- **Q-learning** method avoids the need for a model by learning an action-utility function $Q(s, a)$ instead of a utility function $U(s)$.
- derive a model-free TD update for the Q-values

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

- No transition model
- **SARSA** (for state, action, reward, state, action).
  - updates with the Q-value of the action a' that is actually taken:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma\, Q(s',a') - Q(s,a)] ,$$

Active Reinforcement Learning

```
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, the previous state and action, initially null

  if s is not null then
      increment N_sa[s, a]
      Q[s, a] ← Q[s, a] + α(N_sa[s, a])(r + γ max_a′ Q[s′, a′] − Q[s, a])
  s, a ← s′, argmax_a′ f(Q[s′, a′], N_sa[s′, a′])
  return a
```

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.

Applications of Reinforcement Learning

Applications in game playing

- deep Q-network (DQN) system, the first modern deep RL system by DeepMind
- DQN was trained separately on each of 49 different Atari video games
- learned to drive simulated race cars, shoot alien spaceships, and bounce balls with paddles
- DeepMind's ALPHAGO beat the best human players

Application to robot control

- cart–pole balancing problem, also known as the inverted pendulum
- radio-controlled helicopter flight
    - used policy search over large MDPs
    - often combined with imitation learning and inverse RL given observations of a human expert pilot

Example1 - Policy Based agent

- Pong:
  - https://www.youtube.com/watch?v=YOW8m2YGtRg
  - Policy function:
    - Input: pixel intensities
    - Output: velocity of paddle

Example2 - Q-learning

- Google DeepMind's Deep Q-learning playing Atari Breakout
  - https://www.youtube.com/watch?v=V1eYniJ0Rnk
  - Model Input:
    - Pixel intensities recorded over the last 4 time steps.
  - Model Output:
    - Anticipated future reward for each potential action.

Applications of Reinforcement Learning

(a) (b)

(a) 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 the cart's position x and velocity $\dot{x}$, as well as the pole's angle ϑ and rate of change of angle $\dot{\vartheta}$.

(b) Six superimposed time-lapse images of a single 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 (Ng *et al.*, 2003). 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.)

Summary

- A **model-based reinforcement learning** has a transition model $P(s^t \mid s, a)$ for the environment and learns a utility function $U(s)$.
- *A* **model-free reinforcement learning** agent may learn an action-utility function $Q(s, a)$ or a policy $\pi(s)$.
- Utility learning approach:
  - Direct utility estimation
  - Adaptive dynamic programming (ADP)
  - Temporal-difference (TD)
- Reward shaping and hierarchical reinforcement learning are helpful for learning complex behaviors
- Policy-search methods operate directly on a representation of the policy, attempting to improve it based on observed performance
- Apprenticeship learning through observation of expert behavior can be an effective solution when a correct reward function is hard to specify.