19. Reinforcement Learning

熊溪 Xi Xiong, 金力 Li Jin
xi.xiong@nyu.edu li.jin@sjtu.edu.cn
纽约大学 New York University
上海交通大学 Shanghai Jiao Tong University

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Outline

- Overview of Reinforcement Learning
- Markov Decision Processes
- Dynamic Programming
- Monte Carlo Methods
- Temporal-Difference Learning
- Approximate Solution Methods

References

- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 2018.
- Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 2017.
- David Silver. *Tutorial: Deep Reinforcement Learning*.

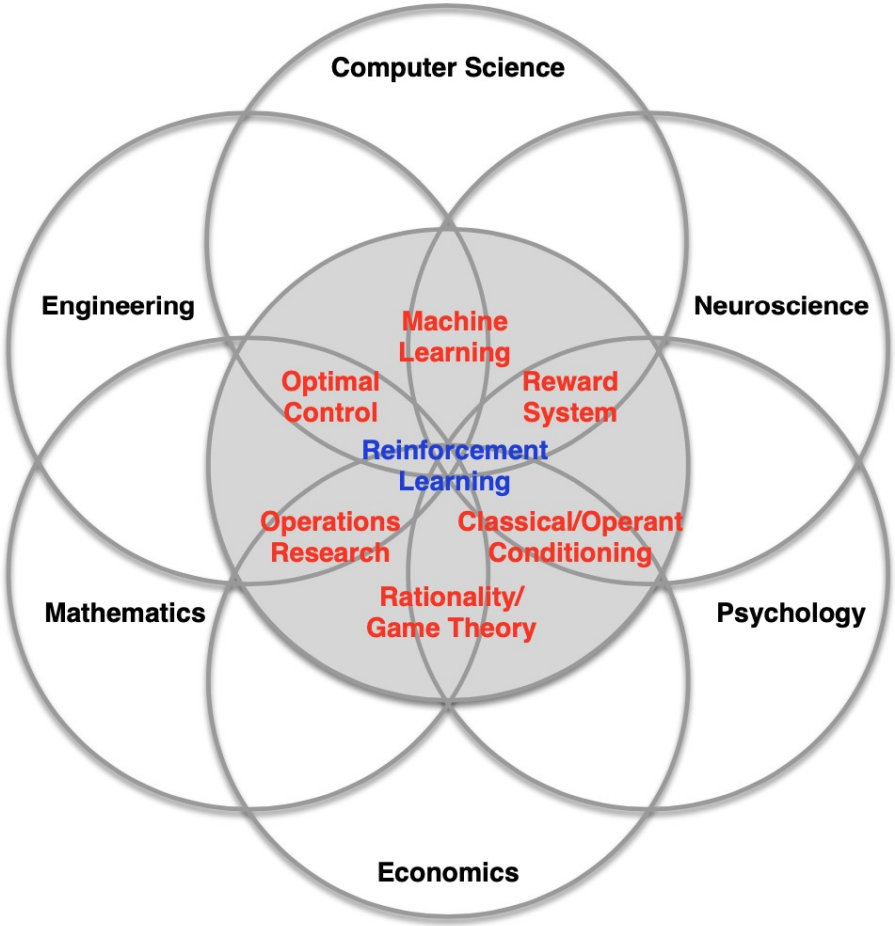
Outline

- Overview of Reinforcement Learning
- Markov Decision Processes
- Dynamic Programming
- Monte Carlo Methods
- Temporal-Difference Learning
- Approximate Solution Methods

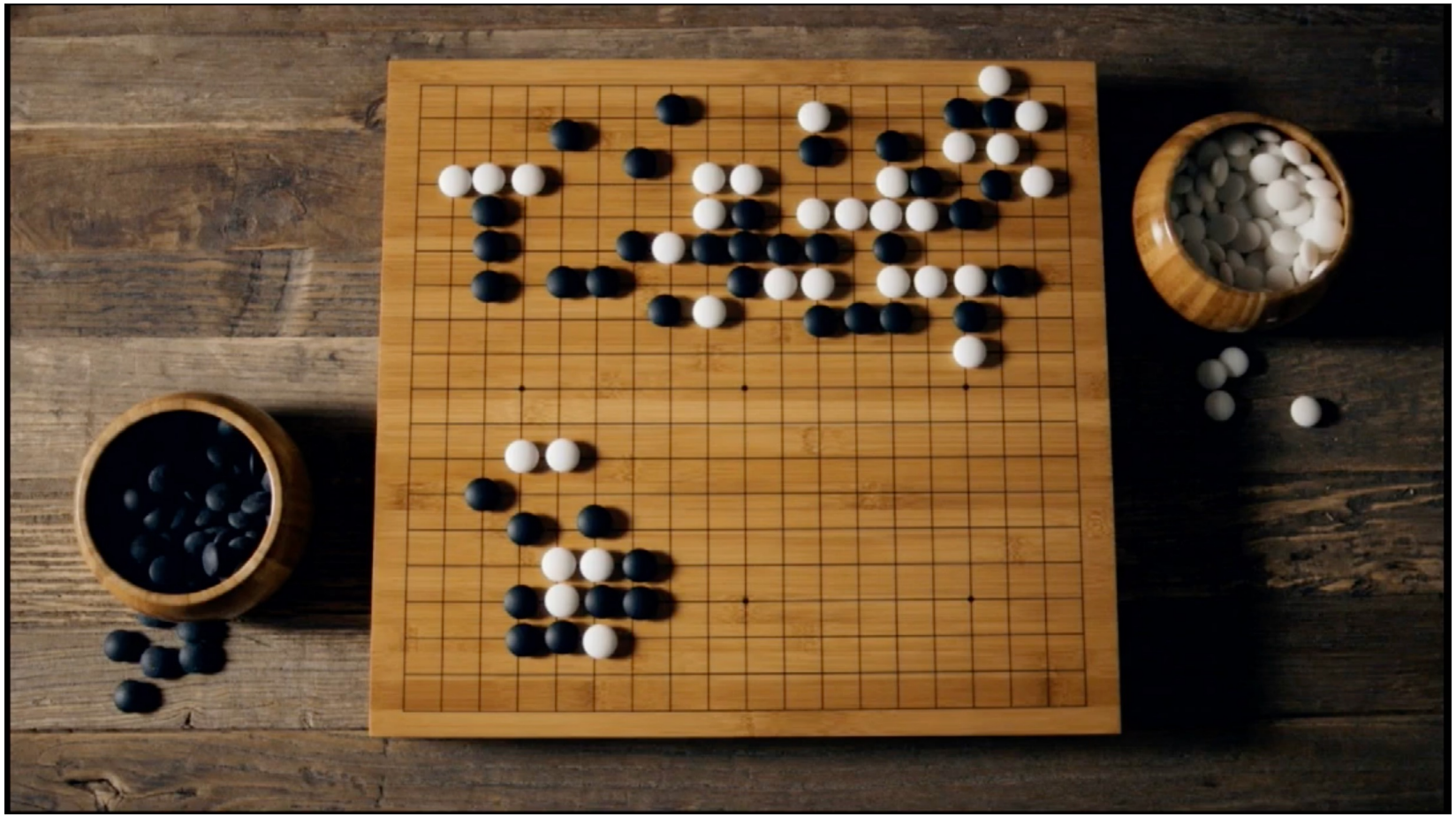
Overview of reinforcement learning

- Reinforcement learning (RL) is a general-purpose framework for decision-making.
- RL is for an agent to take actions under different situations to maximize future reward.
- RL belongs to machine learning.
- RL is also part of a larger trend in artificial intelligence back toward simple general.

Many faces of reinforcement learning



Milestones



AlphaGo and AlphaZero (DeepMind, 2016, 2017)

Milestones

AlphaStar (DeepMind, 2019)

Applications in smart cities

Autonomous vehicle

- Decision making using sensor data
- RL structure — end-to-end control

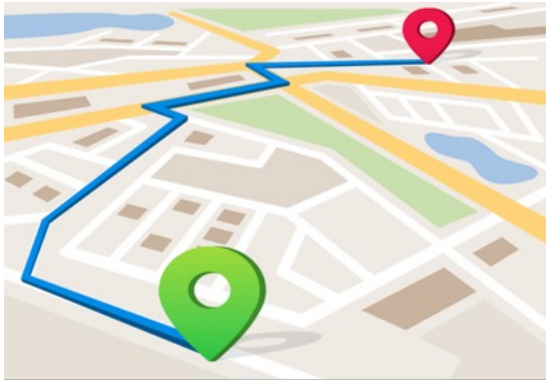
Unmanned aerial vehicles (UAV)

- Connected UAVs
- Multi-agent RL for cooperation

Applications in smart cities

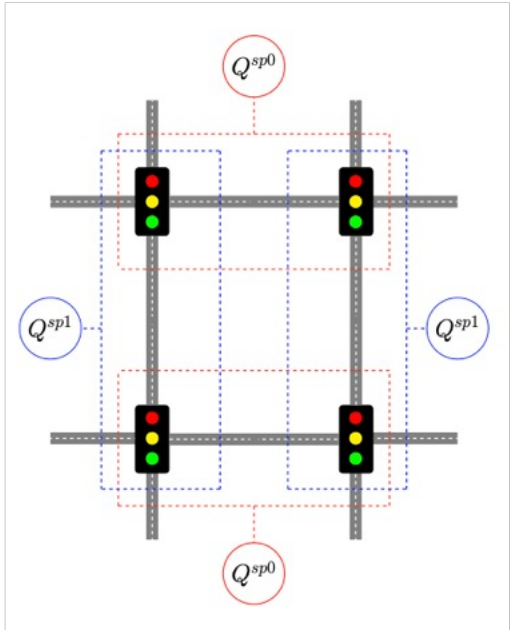
Dynamic routing

- Choose the optimal path
- Network level RL
- Objective — reduce travel time



Traffic light control

- Multi-agent RL for cooperation
- Reduce traffic congestion



Outline

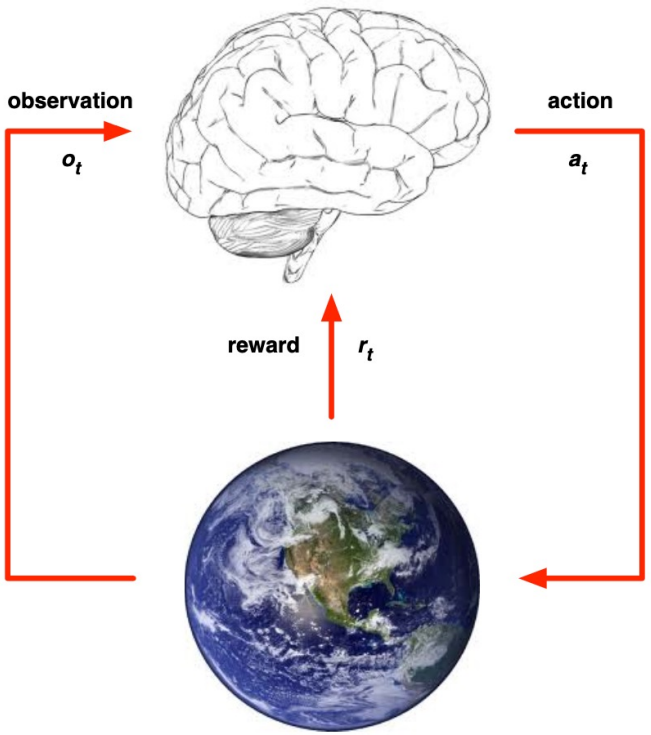
- Overview of Reinforcement Learning
- Markov Decision Processes
- Dynamic Programming
- Monte Carlo Methods
- Temporal-Difference Learning
- Approximate Solution Methods

Markov decision processes

- Markov decision process (MDP) is a discrete-time stochastic control process
- MDPs are a classical formalization of sequential decision making, which actions influence immediate rewards and subsequent situations (states, future rewards)
- MDPs are a mathematically idealized form of the reinforcement learning problem

Agent-environment interface

- At each step t the agent
  - Executes action a_t
  - Receives observation s_t
  - Receives scalar reward r_t
- The environment
  - Receives action a_t
  - Emits observation s_{t+1}
  - Emits scalar reward r_{t+1}
- MDP trajectory
  - Time sequence t = 0, 1, 2, …
  - Trajectory: $s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$



Dynamics

- We use function p to describe dynamics of MDP:

$$p(s', r|s, a) \coloneqq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

- Reward may or may not be random

- With a slight abuse of notations, state-transition probabilities

$$p(s'|s, a) \coloneqq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$
$$= \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

Reward

- Expected reward

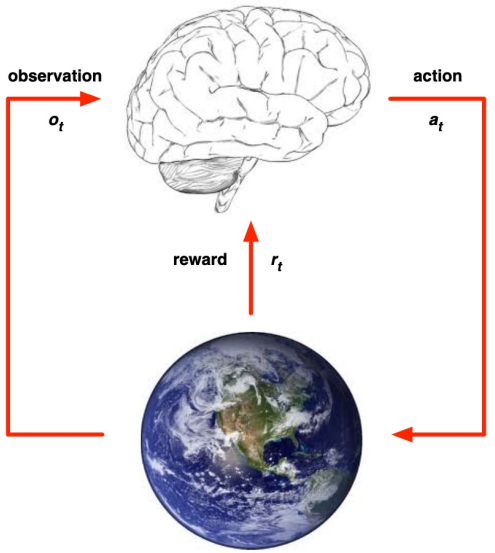
$$r(s,a) \coloneqq \mathrm{E}\{R_t | S_{t-1} = s, A_{t-1} = a\}$$
$$= \sum_{r \in \mathcal{R}} r \sum_{s' \in S} p(s', r | s, a)$$

- Three-argument function

$$r(s,a,s') \coloneqq \mathrm{E}\{R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'\}$$
$$= \sum_{r \in \mathcal{R}} r \frac{p(s', r | s, a)}{p(s' | s, a)}$$

Example: pick & place robot

- Robot arm
- Repetitive pick-and-place task
- State = position/velocity of linkages
- Action = voltages applied to each motor
- Reward = indicator function of success



Example: self-driving vehicle

- Agent: the computer driver
- Environment: this vehicle, other vehicles, roads, pedestrians…
- Action: apply acceleration/brake, rotate steering wheel
- State: vehicle position & angle & their time-derivatives
- Reward: safety, distance covered, fuel savings…

Goals & rewards

- Reward hypothesis:
  - That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).
- Design of reward: science & art
  - Success of an action: pick up an object, arrive at desired destination, collect required information…
  - Consuming less resource: time, fuel, electricity, computational load…
  - Avoid undesirable actions: hit an object, exceed allowable power level…

Returns & episodes

- Suppose a T-step decision process
- We are at step t
- **Return**:

$$G_t := R_{t+1} + R_{t+2} + \cdots + R_T$$

- If T is finite, the task has an **episode** of T steps.
- Task is **episodic**.
- If T is infinite, task is **continuing**.
- **Discounted return**:

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where $0 \leq \gamma \leq 1$ called **discount rate**.

- Reccurssive relation

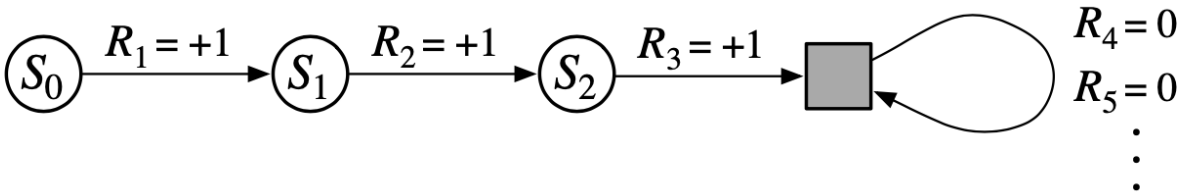
$$G_t = R_{t+1} + \gamma G_{t+1}$$

Unified notation for episodic & continuing tasks

- Introduce "absorbing" state with zero reward

$$G_t := \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

- Hence, it suffices to only consider episodic tasks.



Policies & value functions

- So far we have looked at the environment in which we will make decisions.
- But how to actually make the decisions?
- Recall: set of actions A
- **Policy** is a mapping $\pi: S \to A$ such that

$$\Pr\{A_t = a | S_t = s\} = \pi(a|s)$$

- **State-value function** is a mapping $v_\pi: S \to \mathbb{R}$

$$v_\pi(s) \coloneqq \mathrm{E}_\pi[G_t | S_t = s] = \mathrm{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right]$$

- Value function is for policy π
- Action-value function $q_\pi: S \times A \to \mathbb{R}$

$$q_\pi(s,a) \coloneqq \mathrm{E}_\pi[G_t | S_t = s, A_t = a]$$
$$= \mathrm{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]$$

Consistency condition

- By law of total expectation

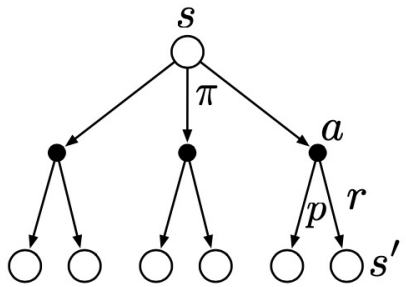
$$v_\pi(s) \coloneqq \mathrm{E}_\pi[G_t|S_t = s] = \mathrm{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma \mathrm{E}_\pi[G_{t+1}|S_{t+1} = s']\right]$$

$$= \sum_a \pi(a|s) \sum_{s'r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

Bellman equation

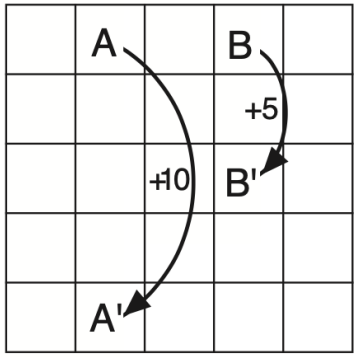
- $v_\pi(s) = \sum_a \left(\pi(a|s) \sum'_{s'r} p(s', r|s, a)[r + \gamma v_\pi(s')]\right)$
- Called the **Bellman equation for π**
- Expresses relationship between value of a state and values of successor states.
- Value of start state = value of expected next state + expected reward for one step.
- Backup diagram: transfer value information back to a state from successors

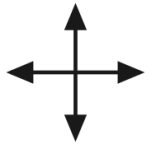


Backup diagram for v_π

Example: gridworld

- Rectangular gridworld representation of a finite MDP.
- The cells of the grid correspond to the states of the environment.
- At each cell, four actions are possible: north, south, east, and west, which deterministically cause the agent to move one cell in the respective direction on the grid.



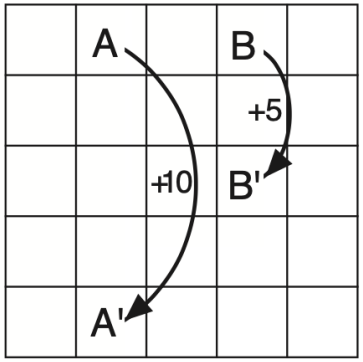


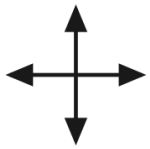
Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Example: gridworld

- Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1.
- Other actions result in a reward of 0, except those that move the agent out of the special states A and B.
- From state A, all four actions yield a reward of +10 and take the agent to A′. From state B, all actions yield a reward of +5 and take the agent to B′.



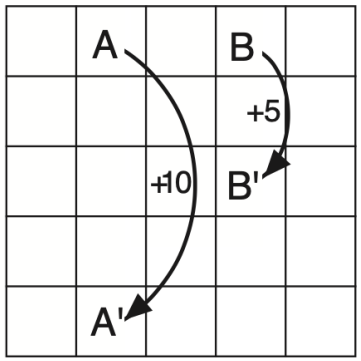


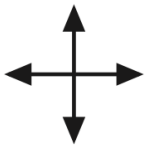
Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Example: gridworld

- Suppose the agent selects all four actions with equal probability in all states.
- Right figure shows the value function, v_π, for this policy, for the discounted reward case with $\gamma = 0.9$.
- This value function was computed by solving the Bellman equations for v_π.





Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Optimal policies & optimal value functions

- Value functions defines a partial ordering over policies
- π_1 better than π_2 if $v_{\pi_1}(s) > v_{\pi_2}(s)$ for all s
- **Optimal policy** π_* is such that

$$v_{\pi_*}(s) \geq v_{\pi}(s) \text{ for all s and all } \pi$$

- **Optimal state-value function**

$$v_*(s) = v_{\pi_*}(s) = \max_{\pi} v_{\pi}(s) \text{ for all s}$$

- **Optimal action-value function**

$$q_*(s,a) = q_{\pi_*}(s,a) = \max_{\pi} q_{\pi}(s,a) \text{ for all s and all a}$$

- Relation between optimal value functions

$$q_*(s,a) = \mathrm{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

Bellman optimality equation

- For v_*

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right].
\end{aligned}
$$

- For q_*

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right].
\end{aligned}
$$

Implementation concerns

- Explicitly solving Bellman optimality equation provides one route to finding an optimal policy.
- However, this is rarely directly useful: akin to an exhaustive search, looking ahead at all possibilities, computing their probabilities of occurrence and their desirabilities in terms of expected rewards.
- This solution relies on assumptions rarely true in practice:
  - accurate knowledge of dynamics of environment;
  - enough computational resources to complete the computation of the solution;
  - Markov property.

Optimality & approximation

- Accurate knowledge of dynamics of environment
  - How to get transition probabilities?
  - Are there enough data to estimate transition probabilities?
  - What if some states were never reached or some actions were never taken in certain states?
- Enough computational resources to complete the computation of the solution (**tabular** case)
  - How many states?
  - How many state-action pairs?
  - How many Bellman equations?
- Markov property
  - Is the environment stationary?
  - What if the environment gradually drifts over time?

Reinforcement learning (RL)

- A class of approaches to decision making in incomplete-information and stochastic settings.
- Core concepts:
  - Dynamic programming: perfect prior knowledge of system dynamics (transition probabilities)
  - Monte Carlo Methods: learning expected rewards and returns through averaging experiences
  - Temporal-difference learning: combination of the above two

Outline

- Overview of Reinforcement Learning
- Markov Decision Processes
- Dynamic Programming
- Monte Carlo Methods
- Temporal-Difference Learning
- Approximate Solution Methods

Dynamic programming (DP)

- A collection of algorithms that can be used to compute optimal policies for MDPs
- Pre-requisite: perfect knowledge of model
- Finds optimal policy heuristically rather than analytically
- Theoretically important: provides an essential foundation for RL ideas
- Practically of limited utility

Policy evaluation: find v_π

- Recall Bellman equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

- Analytical solution may be hard
- Iterative method: for given policy π, consider a sequence of value estimates $v_0(s), v_1(s), \ldots$
- Iterative policy evaluation

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'r} p(s', r|s, a)[r + \gamma v_k(s')]$$

- Convergence $\lim_{k\to\infty} v_k(s) = v_\pi(s)$ is theoretically guaranteed.

Policy improvement: find v_*

- Suppose a deterministic policy $\pi(s)$
- State value $v_\pi(s)$, action value $q_\pi(s,a)$

Theorem (Policy improvement)

Given a policy π, if there exists another policy π' such that

$$q_\pi(s,\pi'(s)) \geq v_\pi(s) \qquad \forall s$$

then π' must be no worse than π.

- Greedy policy improvement

$$\pi'(s) = \text{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

- After doing this for each state s, we obtain a better policy π'
- Can be extended to probabilistic policies.

Policy iteration: find π_*

- Initialize $\pi_0, v_{\pi_0,0}$
- Policy evaluation $v_{\pi_0,1}, v_{\pi_0,2}, \ldots, v_{\pi_0,n}$
- Policy improvement π_1
- Iterate until convergence
- Must give optimal policy and optimal values functions in finite numbers of iterations for finite MDP.

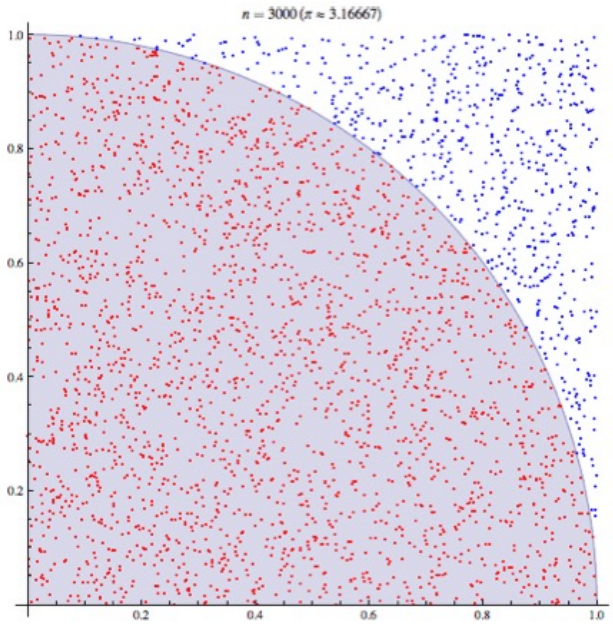
$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

Outline

- Overview of Reinforcement Learning
- Markov Decision Processes
- Dynamic Programming
- Monte Carlo Methods
- Temporal-Difference Learning
- Approximate Solution Methods

Monte Carlo (MC) methods

- No prior knowledge of system dynamics
- Only know states and rewards
- Core: use observed average return to estimate value



Monte Carlo prediction for v_π

- Input: policy π to be evaluated
- Initialize:
  - Arbitrary $V_0(s)$ for all s
- Loop: for kth episode of length T
  - Following π generate $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
  - Let t_s be the first-visit time to state s
  - Compute $Return_k(s) = R_{t_s+1} + \gamma R_{t_s+1} + \cdots + \gamma^{T-t_s+1} R_T$
  - Update $V_k(s) = \frac{1}{k} \sum_k Return_k(s)$
  - V_k not affected by V_{k-1}
- Convergence is theoretically guaranteed.

$$\lim_{k \to \infty} V_k(s) = v_\pi(s) \qquad \text{a.s.}$$

Monte Carlo prediction for q_π

- Input: policy π to be evaluated
- Initialize:
  - Arbitrary $Q_0(s,a)$ for all s,a
- Loop: for kth episode of length T
  - Following π generate $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
  - Let $t_{s,a}$ be the first-visit time to state s when a is taken
  - Compute $Return_k(s,a) = R_{t_{s,a}+1} + \cdots + \gamma^{T-t_{s,a}+1} R_T$
  - Update $Q_k(s,a) = \frac{1}{k}\sum_k Return_k(s,a)$
- Convergence is theoretically guaranteed.

$$\lim_{k\to\infty} Q_k(s,a) = q_\pi(s,a) \qquad a.s.$$

Monte Carlo control: policy improvement

- Initialize:
  - policy π to be improved
  - Arbitrary $Q_0(s,a)$ for all s,a
- Loop: for kth episode of length T
  - Following π generate $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
  - Let $t_{s,a}$ be the first-visit time to state s when a is taken
  - Compute $Return_k(s,a) = R_{t_{s,a}+1} + \cdots + \gamma^{T-t_{s,a}+1} R_T$
  - Update $Q_k(s,a) = \frac{1}{k}\sum_k Return_k(s,a)$
- Improve policy

$$\pi'(s) = \text{argmax}_a Q_K(s,a)$$

- Then iterate policy to obtain optimal solution

Outline

- Overview of Reinforcement Learning
- Markov Decision Processes
- Dynamic Programming
- Monte Carlo Methods
- Temporal-Difference Learning
- Approximate Solution Methods

Temporal-Difference (TD) learning

- Combination of DP & MC ideas
- Like MC: update value from experience
- Like DP: make decision without waiting for a final outcome
- TD prediction
- Sarsa
- Q-learning

TD prediction for v_π

- Initialize:
  - Policy π to be evaluated
  - Parameter: step size α
  - Value function $V(s)$ for all s
- Loop: for time 1, 2, … T in an episode
  - Following π generate $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
  - At the kth time step:

$$V(S_k) = V(S_k) + \alpha[R_k + \gamma V(S_{k+1}) - V(S_k)]$$

  - Values of other states unchanged
- At end of loop $V(s) \approx v_\pi(s)$
- Compared to DP: integration of observed rewards
- Compared to MC: fewer episodes needed

Sarsa: On-policy TD control

- Initialize
  - Parameters step size α
  - Action value estimate $Q(s,a)$
- Loop within an episode
  - Suppose S_t, take action $A_t = \max_a Q(S_t, a)$, get R_{t+1}, S_{t+1}
  - Take action $A_{t+1} = \max_a Q(S_{t+1}, a)$
  - Update
    $Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S, a)]$
- At end of loop $Q(s,a) \approx q_*(s,a)$

Q-learning: off-policy TD control

- Initialize
  - Parameters step size α
  - Action value estimate $Q(s,a)$
- Loop within an episode
  - Suppose S_t, take action $A_t = \max_a Q(S_t,a)$, get R_{t+1}, S_{t+1}
  - Update
    $Q(S_t,A_t) = Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S,a)]$
- At end of loop $Q(s,a) \approx q_*(s,a)$

Summary

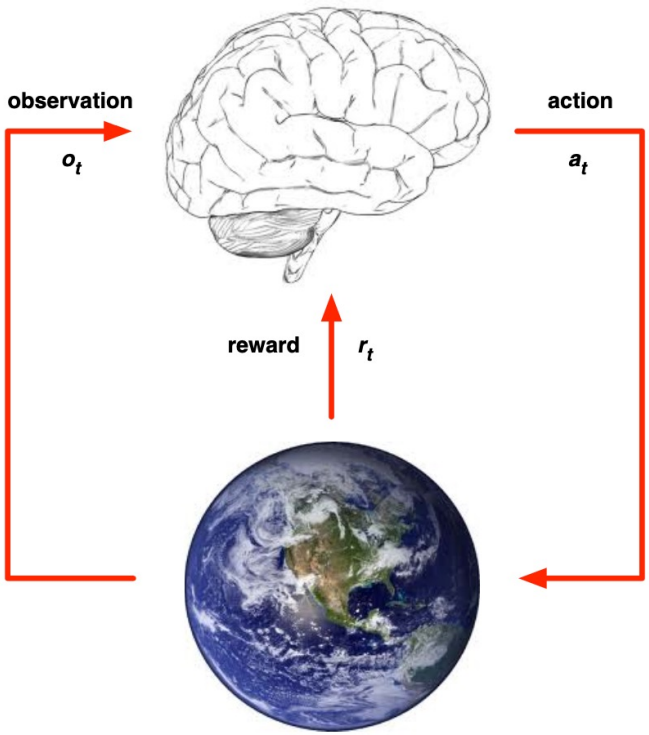
- Markov decision process
  - State, action, rewards, policy
  - state value & action value
  - Bellman equation
  - Bellman optimality equation
- Dynamic programming
  - Policy evaluation, improvement, iteration
- Monte Carlo methods
  - Average experience
- Temporal-difference learning

Outline

- Overview of Reinforcement Learning
- Markov Decision Processes
- Dynamic Programming
- Monte Carlo Methods
- Temporal-Difference Learning
- Approximate Solution Methods

Reinforcement learning

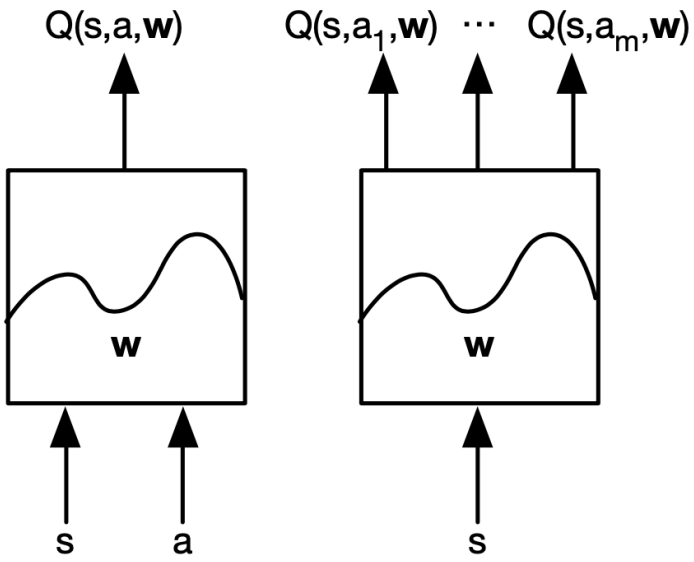
- At each step t the agent
  - Executes action a_t
  - Receives observation o_t
  - Receives scalar reward r_t
- The environment
  - Receives action a_t
  - Emits observation o_{t+1}
  - Emits scalar reward r_{t+1}



Deep reinforcement learning

- Represent value function by Q-network with weight w

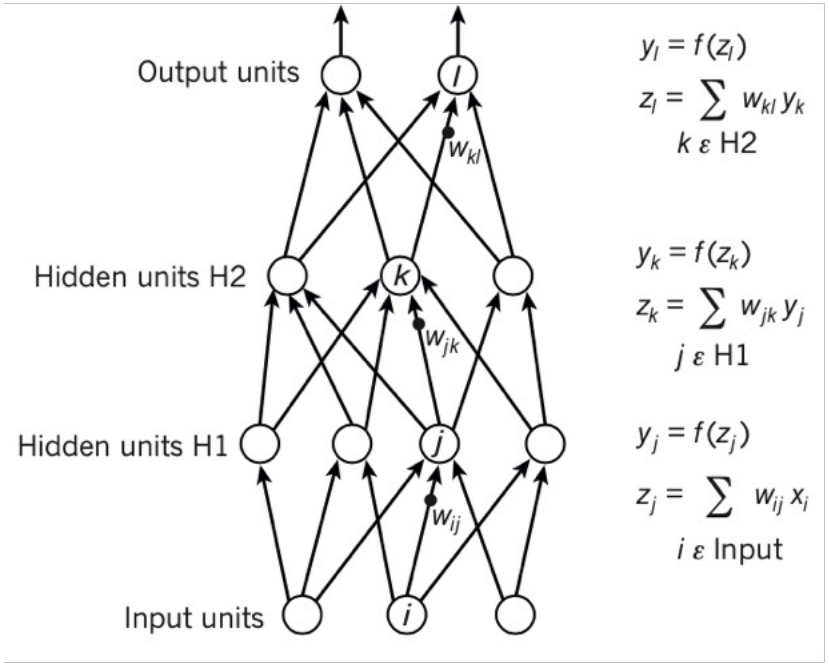
$$Q(s, a, w) \approx Q^*(s, a)$$

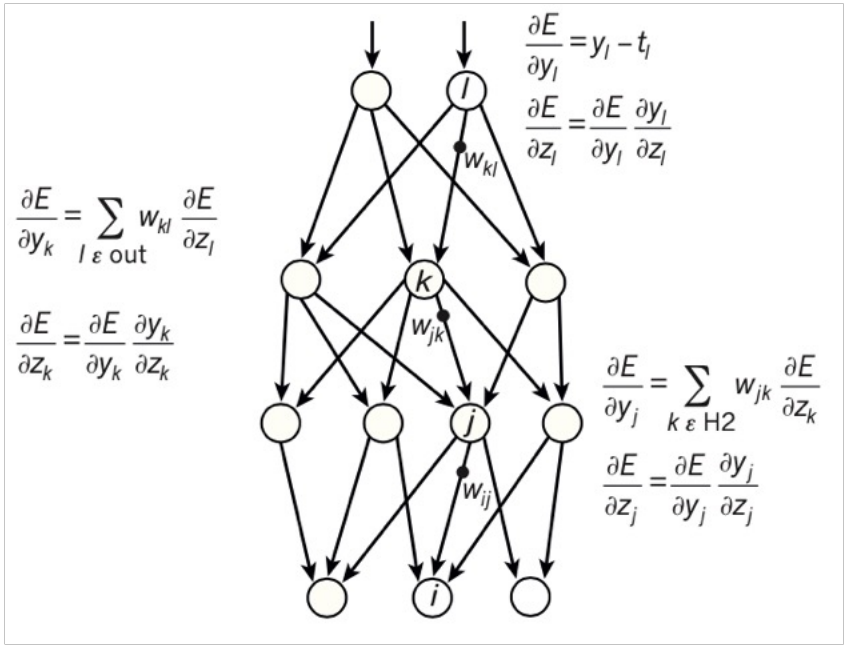


Deep learning

- A deep neural network is typically composed of
  - Linear transformations
    $$h_{k+1} = W h_k$$
  - Non-linear activation functions
    $$h_{k+2} = f(h_{k+1})$$
  - A loss function on the output: e.g.
    - Mean-squared error $l = \|y^* - y\|^2$
    - Log likelihood $l = \log \mathbb{P}(y, y^*)$

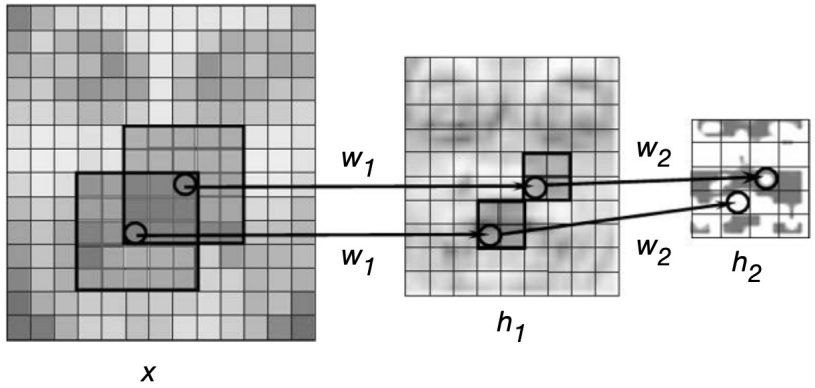
Multilayer NNs and backpropagation



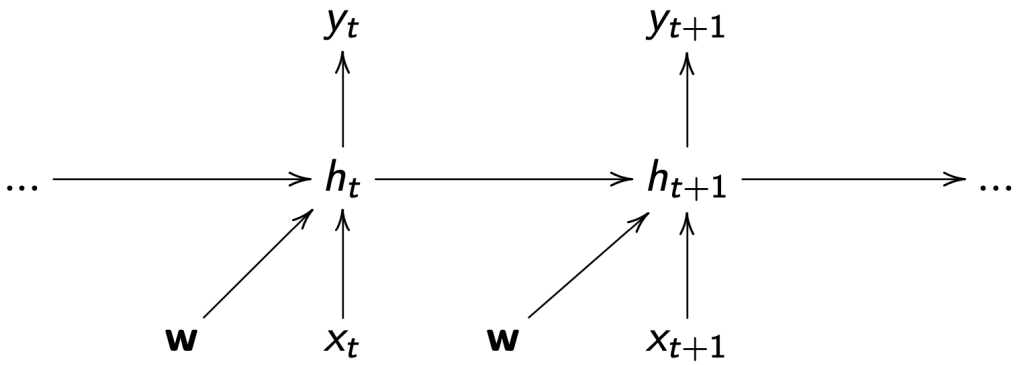


Weight sharing

- Convolutional neural network shares weight between local regions



- Recurrent neural network shares weight between time-steps



Value-based RL: Deep Q-Networks

- Optimal Q values should obey Bellman equation:

$$Q^*(s,a) = \mathrm{E}_{s'}\left[r + \gamma \underset{a'}{max} Q^*(s',a') | s,a\right]$$

- Treat right-hand side $r + \gamma \underset{a'}{max} Q^*(s',a',\boldsymbol{w})$ as a target
- Minimize MSE loss by stochastic gradient descent

$$L = \left(r + \gamma \underset{a'}{max} Q(s',a',\boldsymbol{w}^-) - Q(s,a,\boldsymbol{w})\right)^2$$

- Converges to Q^* using lookup representation
- A simple but impressive approach, it has discovered a new way to take action!

DQN in Atari

- DQN paper

Human-level control through deep reinforcement learning

www.nature.com/articles/nature14236

Policy-based rl

- Represent policy by deep network with weights $\boldsymbol{u}$

$$a = \pi(s, \boldsymbol{u})$$

- Define objective function as total discounted rewards

$$L(\boldsymbol{u}) = \mathrm{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots | \pi(\cdot, \boldsymbol{u})]$$

- The gradient of a deterministic policy $a = \pi(s)$ is given by

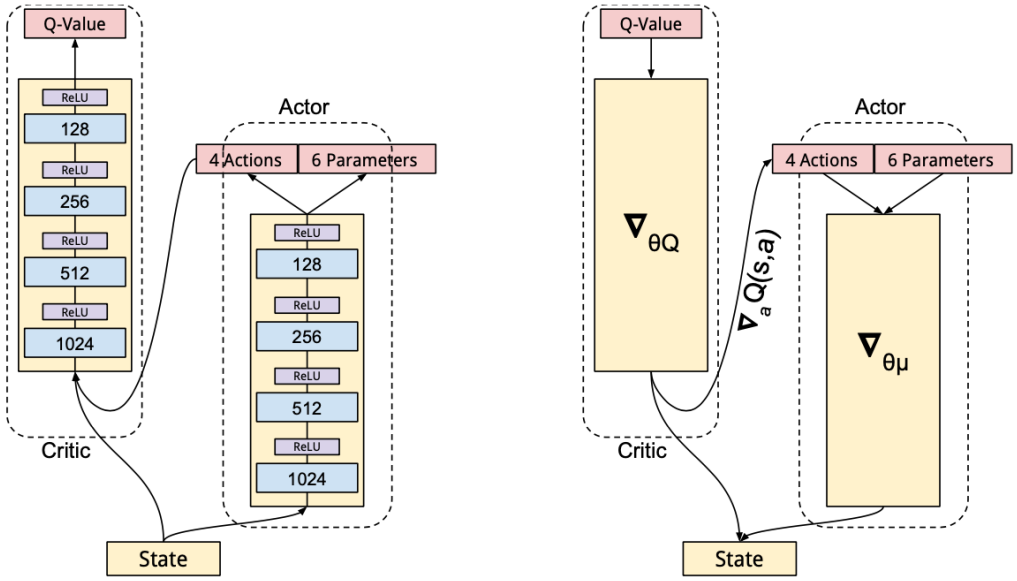
$$\frac{\partial L(\boldsymbol{u})}{\partial \boldsymbol{u}} = \mathrm{E}\left[\frac{\partial Q^{\pi}(s,a)}{\partial a}\frac{\partial a}{\partial \boldsymbol{u}}\right]$$

- If a is continuous and Q is differentiable

Deep deterministic policy gradient

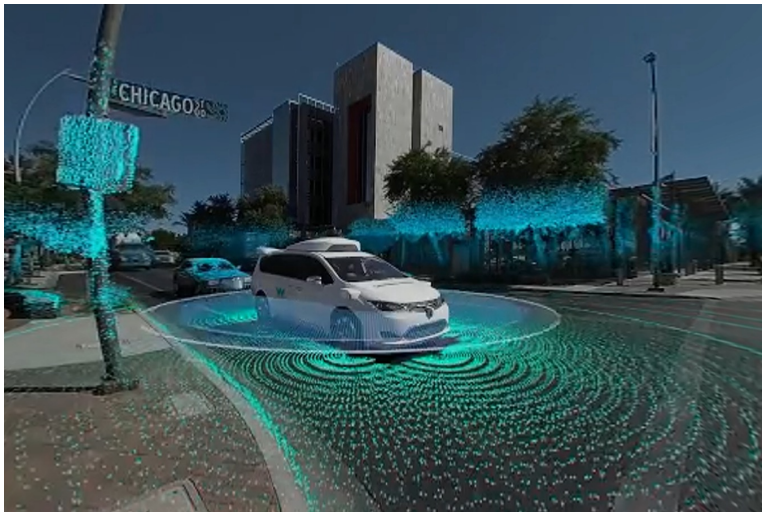
- Deep RL for continuous actions
- Estimate value function $Q(s,a,\boldsymbol{w}) \approx Q^{\pi}(s,a)$
- Update policy parameters $\boldsymbol{u}$ by

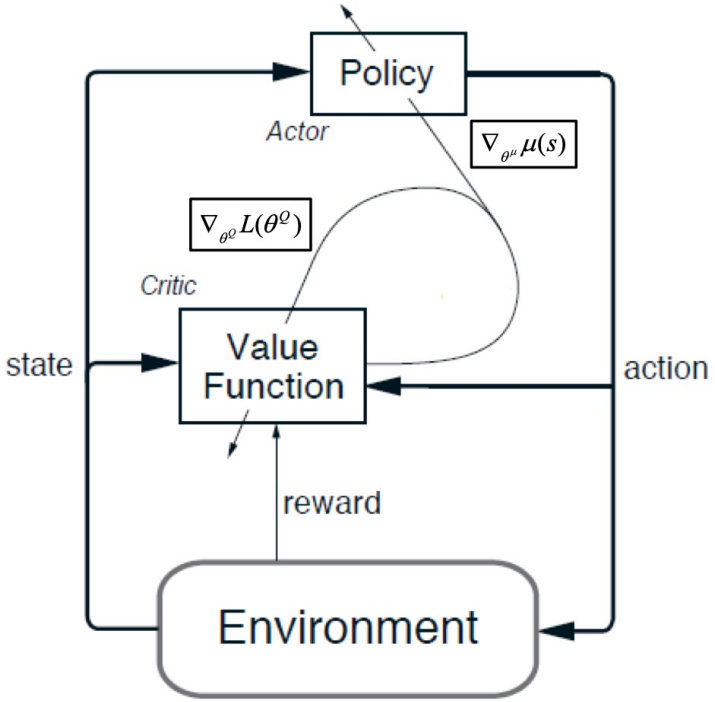
$$\frac{\partial L}{\partial \boldsymbol{u}} = \frac{\partial Q(s,a,\boldsymbol{w})}{\partial a}\frac{\partial a}{\partial \boldsymbol{u}}$$



Example: autonomous vehicles

- End-to-end control for autonomous vehicles





Multi-agent RL

- Unmanned aerial vehicles (UAV)
- Game of Go
- Poker games
- Team-battle video games: StarCraft II, Dota




Q & A