# Introduction to Reinforcement Learning

Hongning Wang
CS@THU

# Reinforcement learning

**Reinforcement learning (RL)** is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward.

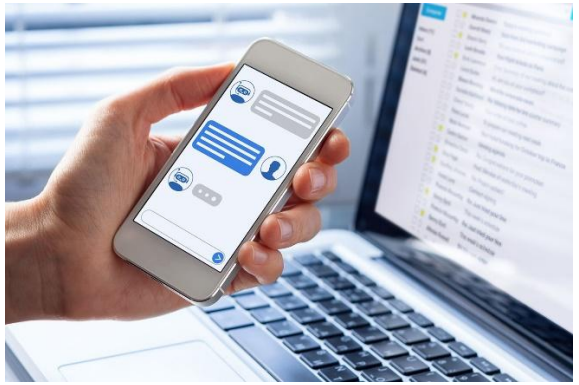
2018
- Wikipedia

# Reinforcement learning

- It is a family of problems
  - Sequential decision making

Game playing

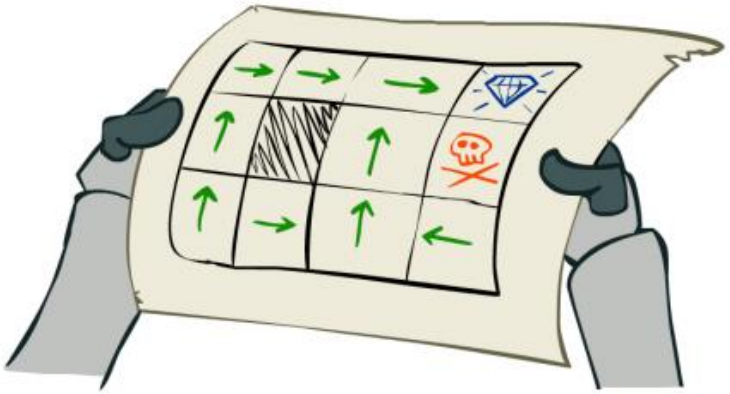

Self-driving car



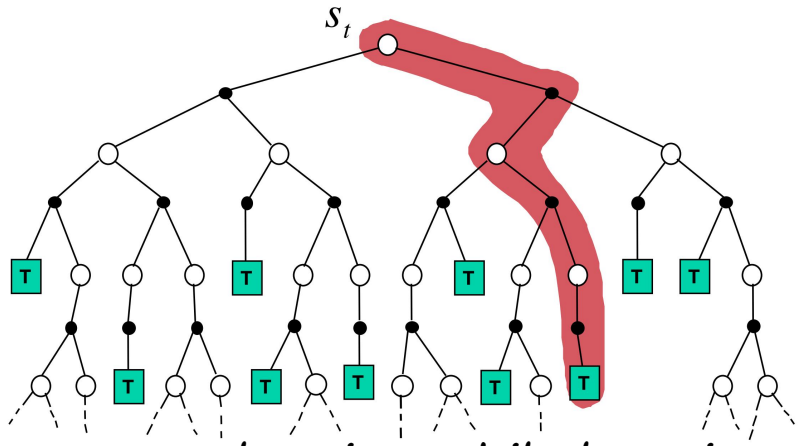
Conversational system

# Reinforcement learning

- It is a family of solutions
  - Taking a series of actions to maximize cumulative return



Planning




~~Planning~~ while learning

Reinforcement

*Image credit: David Silver, "Model-Free Prediction"*

# Reinforcement learning

- It is a collection of fields that study such problems and solutions
  - Computer science, psychology, neuroscience, optimization, operational research, and many others

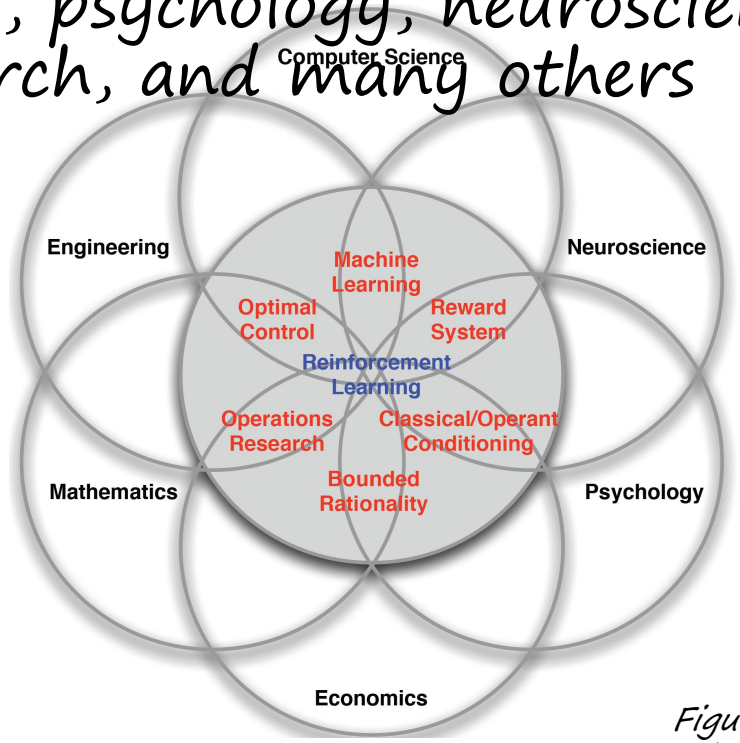



Figure credit: David Silver, "Introduction to RL"

# Full v.s., partial information

- Full information – in most supervised ML
  - $(x_t, y_t)$ is always given
- Partial information – bandit feedback
  - Only $L(f_\theta(x_t), y_t)$ is provided

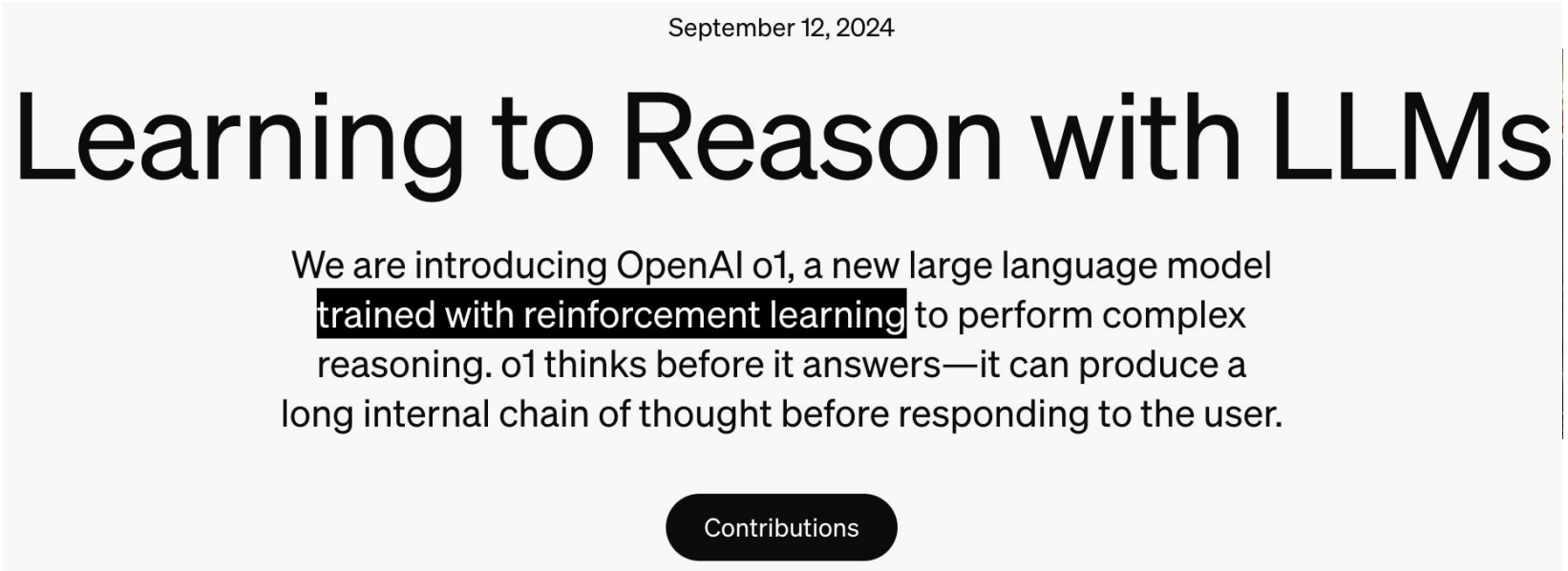
$(x_t, \hat{y}_t = \text{Beagle})$ ✗

# A quick comparison

- Reinforcement learning
  - Reward
  - Partial information
  - Delayed consequence
  - Agent's choice affects the subsequent data it receives, i.e., non-i.i.d.
- Supervised machine learning
  - Ground-truth labels
  - Full information
  - Immediate consequence
  - Pre-determined data distribution, i.e., i.i.d.

# Why reinforcement learning

- Sequential decision making is everywhere



# Why reinforcement learning

- Sequential decision making is challenging
  - Huge search space

| Board size n×n | $3^{n^2}$ | Percent legal | $L$ (legal positions) (A094777)[11] |
|---|---|---|---|
| 1×1 | 3 | 33.33% | 1 |
| 2×2 | 81 | 70.37% | 57 |
| 3×3 | 19,683 | 64.40% | 12,675 |
| 4×4 | 43,046,721 | 56.49% | 24,318,165 |
| 5×5 | 847,288,609,443 | 48.90% | 414,295,148,741 |
| 9×9 | $4.43426488243\times10^{38}$ | 23.44% | $1.03919148791\times10^{38}$ |
| 13×13 | $4.30023359390\times10^{80}$ | 8.66% | $3.72497923077\times10^{79}$ |
| 19×19 | $1.74089650659\times10^{172}$ | 1.20% | $2.08168199382\times10^{170}$ |

Source: https://en.wikipedia.org/wiki/Go_and_mathematics


Complexity: $10^{50}$

# How to do reinforcement learning

- A taxonomy of solutions

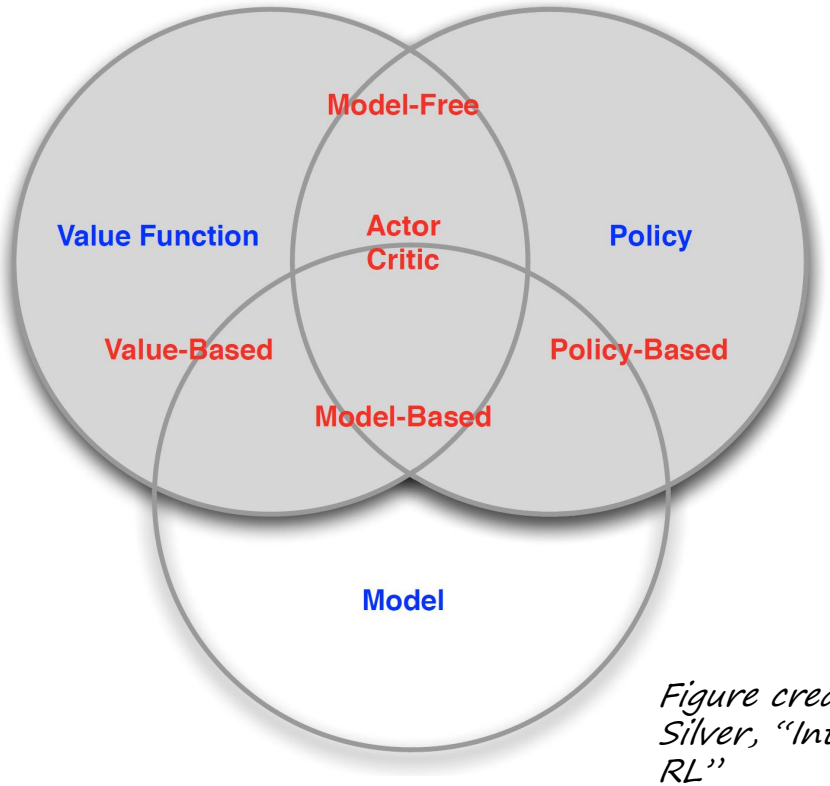
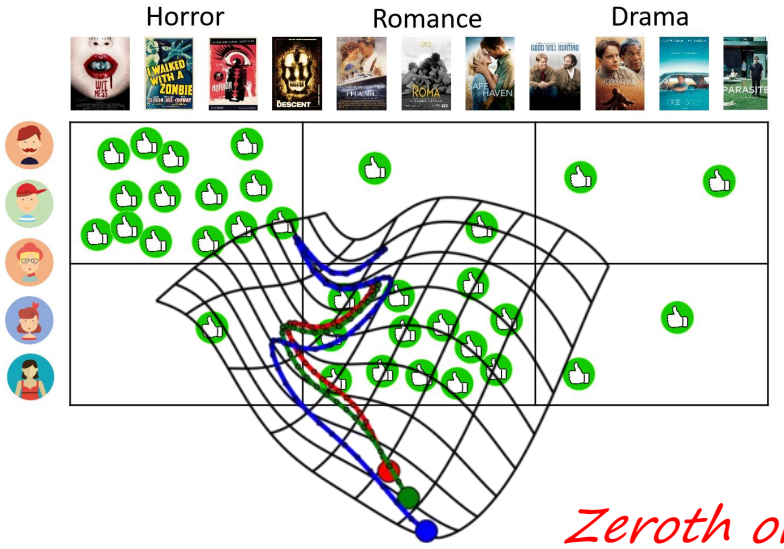



Figure credit: David Silver, "Introduction to RL"

# Action taking in reinforcement learning

- Making a choice out of presented options
  - Discrete actions
    - Move left or right in Atari Breakout game
    - Recommend an item to a target user
  - Continuous actions
    - Drone
    - Model


n with




~~Out~~ of agent's control!

Zeroth order optimization

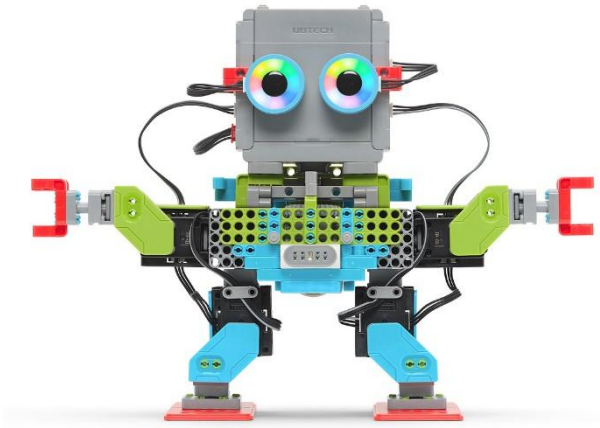
# Reward in reinforcement learning

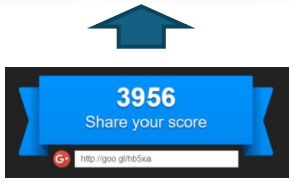
- A scalar feedback signal about the taken action
  - Suggest good/bad immediate consequence of the action
    - Score in Atari game
    - User clicks/purchase in a recommender system
    - Change of black-box function value
  - Delayed feedback
    - GO game
    - Generate a sentence in chat-bot
  - Goal of RL – maximize cumulative rewards
    - Reward hypothesis: "*All goals can be described by the maximization of expected cumulative reward.*"

# How to take an action

- With respect to the current observation

Observation $o_t$






Reward $r_t$

Action $a_t$

# How to take an action

- With respect to history
  - How did we reach the current observation

$$o_t$$

History  Current observation

  - Why do we care about history?
    - In case this has happened before
    - Generalize from history
  - State – a function of history
    - $s_t = f(o_1, a_1, r_1, o_2, a_2, r_2, \ldots\ldots, o_t)$

How to construct states?

# How to take an action

- To maximize cumulative reward in future

$o_1, a_1, r_1, o_2, a_2, r_2, \ldots\ldots, o_t, a_t$

History

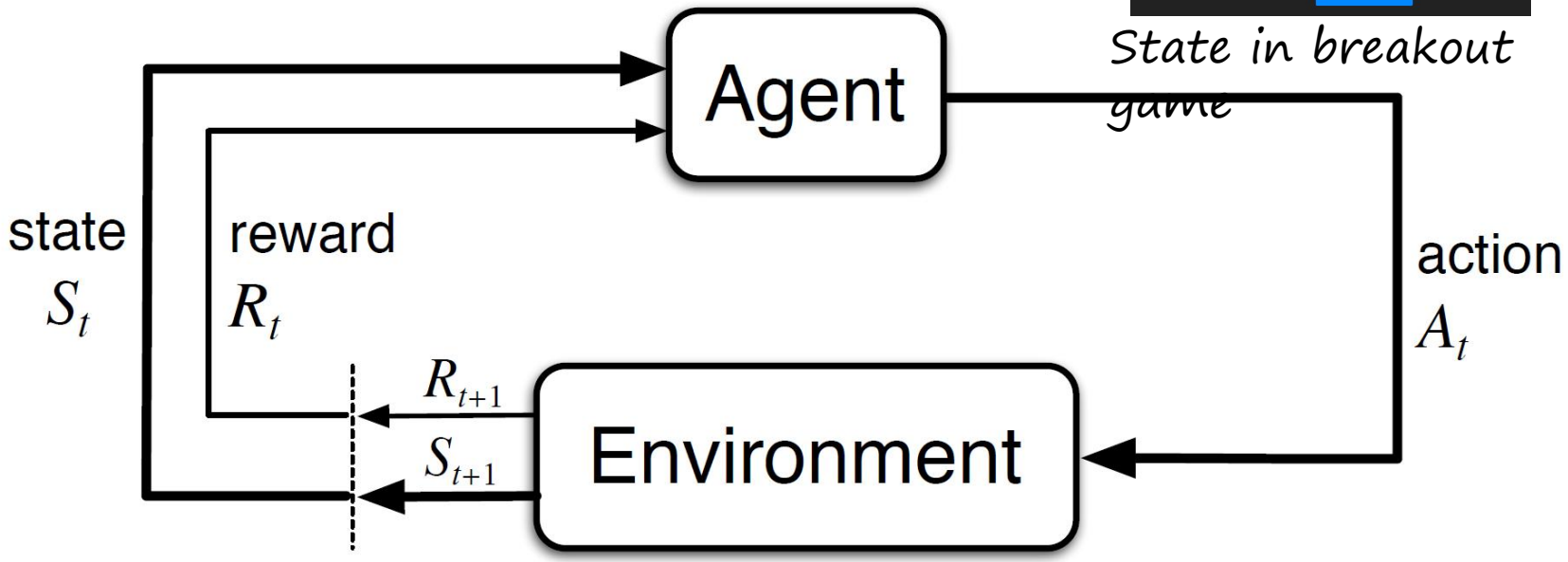
Future

- Value function
  - State-action value $v_\pi(s_t, a_t) = \boldsymbol{E}_\pi\left[\sum_{i=t}^{T} \gamma^{i-t} r_{i-t+1}\right]$ ← Oftentimes approximation is needed
  - State value $v_\pi(s_t) = \boldsymbol{E}_{a_t \sim \pi(s_t)}[v_\pi(s_t, a_t)]$ ← Why do we need this?
- Goal: choose an action that leads to a highest value state

With respect to a particular policy!

# Markov decision process

State in breakout game




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

# Bellman equation for value function

- Value function Return $G_t$
  - $v_\pi(s_t) = \boldsymbol{E}_\pi\left[\sum_{i=t}^{\infty} \gamma^{i-t} r_{i-t+1}\right]$
- Compute it in a recursive way

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$

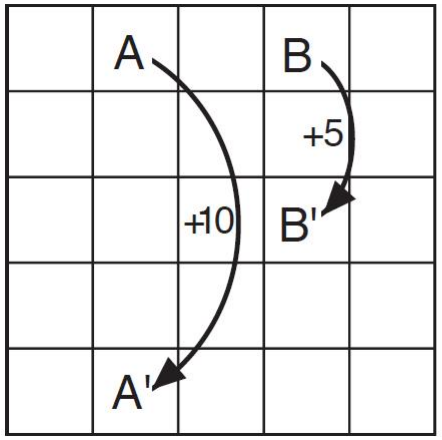
Unique solution of this equation

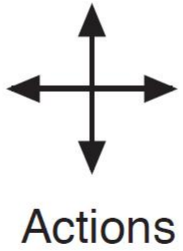
w.r.t a particular policy $\pi$

# Bellman equation for value function

- Gridworld example ($\gamma$=0.9)
  - Q: how do we get the values at the first place?
  - A: solving a system of equations with $|S|$ variables and $|S|$ equations

$$0.25 \times (2.3 + 0.7 - 0.4 + 0.4) \times 0.9 = 0.675$$




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

$$v = 0.25 \times [-1.2 \times 0.9 - 1.4 \times 0.9 + (-1 + 0.9v) \times 2]$$

# Bellman equation for value function

- A closer look with a matrix form

$$v_\pi(s) = r\big(s, \pi(s)\big) + \gamma p(s'|s, \pi(s)) v_\pi(s')$$

$$V_\pi = R_\pi + \gamma P_\pi^{ss'} V_\pi$$

$$(I - \gamma P_\pi^{ss'}) V_\pi = R_\pi$$

$$V_\pi = \boxed{\left(I - \gamma P_\pi^{ss'}\right)^{-1}} R_\pi$$

State occupancy matrix

Complexity: $O(|S|^3)$

Too large?

$$d_\pi^s(s') = E\left[\sum_{t=1}^{\infty} \gamma^{t-1} \Delta(s_t = s') | s_1 = s, \pi\right]$$

Each row depicts the discounted probability of visiting state s' starting from state s

# Policy evaluation

- Assume a known environment described by a finite MDP
  - $p(s', r|s, a)$
  - Finite state space and action space
- Compute state-value function $v_\pi$
  - Iterative policy evaluation

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

Current iteration

Last iteration

# Policy improvement

- Improvement achieved by a "greedy" search

At each state, take the best action at the moment

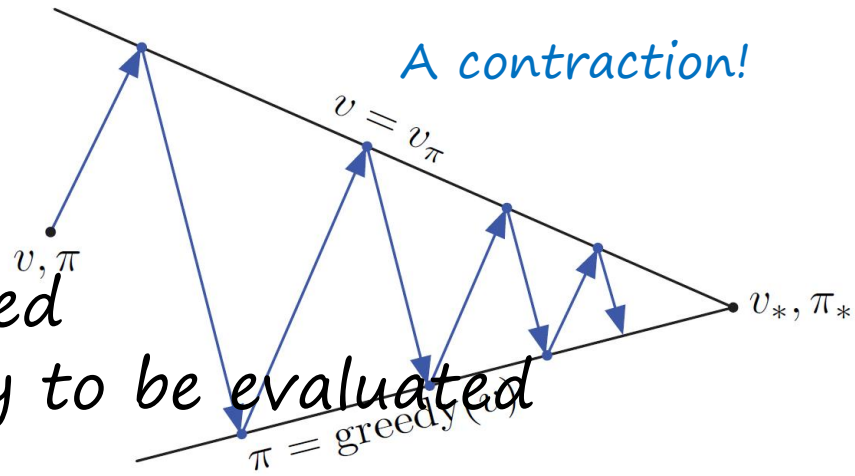
$$
\begin{aligned}
\pi'(s) &\doteq \arg\max_a q_\pi(s,a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right],
\end{aligned}
$$

One step look ahead!

What if no change happ
found!

# Policy iteration

- Iteratively improve the policy
  - Policy evaluation → where can be improved
  - Policy improvement → get the next policy to be evaluated




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

*Exponential convergence!*

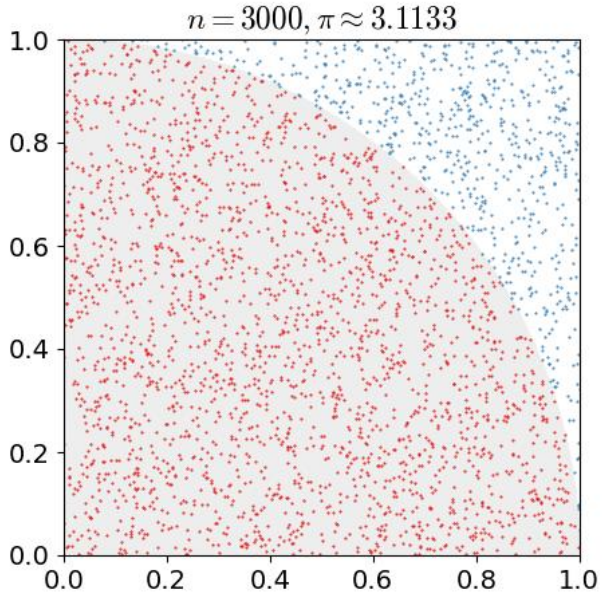
Like an EM algorithm?

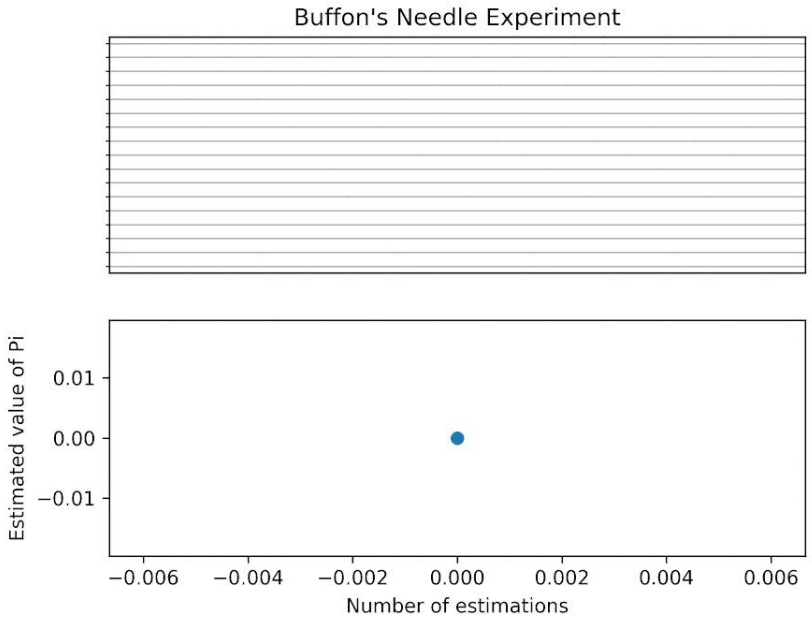
# Complexity of dynamic programming

- Typically it is quadratic to the number of states
  - At least one has to sweep through all the successor states for policy evaluation or improvement
  - Key performance bottleneck in practice
- Approximations!
  - Sample backups – using sampled rewards and successor states ➡ Monte Carlo methods!
    -
  - Functional approximation of the value function ➡ Q-learning
    -

# Monte Carlo methods

- A family of methods that rely on repeated random sampling to obtain numerical results



# Monte Carlo prediction

- Policy evaluation (a.k.a prediction)
  - Assume the environment is accessible
    - Interact with the environment to get experiences — Sampling
    - Use the experiences to estimate the ~~values~~ — Evaluation

$$v_\pi(s_t) = \mathrm{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-1} R_T | s_t]$$

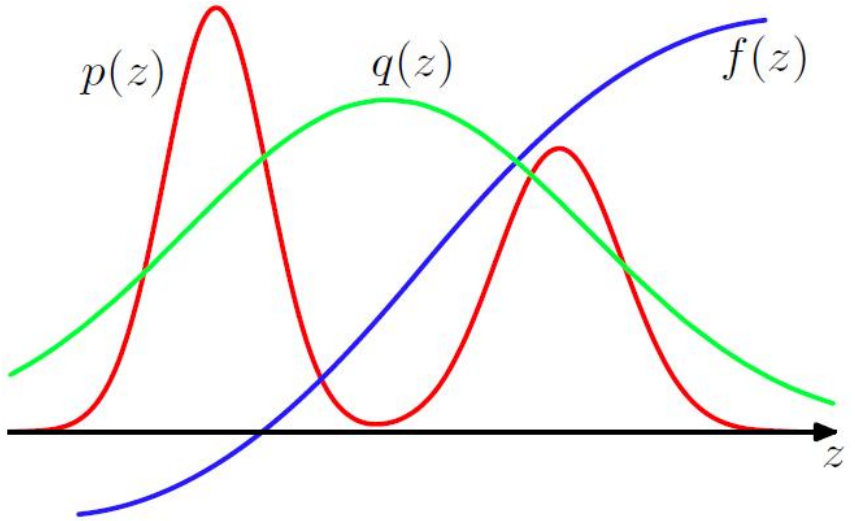
With respect to the given policy $\pi(a|s)$!

$$\{s_{n,t}, a_{n,t}, R_{n,t}, s_{n,t+1}, a_{n,t+1}, R_{n,t+1}, \ldots\ldots, s_{n,T}, a_{n,T}, R_{n,T}\}_{n=1}^{N}$$

$$v_\pi(s_t) = \frac{1}{N}\sum_{n=1}^{N}\left[R_{n,t} + \gamma R_{n,t+1} + \gamma^2 R_{n,t+2} + \ldots + \gamma^{T-1} R_{n,T} | s_t\right]$$

# Importance sampling

- Let's sample from a proposal distribution and reweigh its contribution to the expectation




$$\begin{aligned}\mathbb{E}[f] &= \int f(\mathbf{z})p(\mathbf{z})\,\mathrm{d}\mathbf{z} \\ &= \int f(\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}q(\mathbf{z})\,\mathrm{d}\mathbf{z} \\ &\simeq \frac{1}{L}\sum_{l=1}^{L}\frac{p(\mathbf{z}^{(l)})}{q(\mathbf{z}^{(l)})}f(\mathbf{z}^{(l)})\end{aligned}$$

Assumption: both distributions have the same support!

Importance weight

# Off-Policy MC evaluation

- Use importance sampling to reweigh experiences
  - As the environment follows an MDP

$$\begin{aligned}
&\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
&= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
&= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),
\end{aligned}$$

  - The importance weight is the product of the series of actions under two policies

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \boxed{\frac{\pi(A_k|S_k)}{b(A_k|S_k)}}$$

Source of large variances!

# Temporal difference methods

- A method in between DP and MC
  - Policy evaluation to begin with
    - MC: $V(s_t) \leftarrow V(s_t) + \alpha[G_t - V(s_t)]$
    - DP: $V(s_t) = R_\pi + \gamma P_\pi^{ss'} V(s_t)$

Incremental mean update:

$$\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{1}{k}\left(x_k + \sum_{i=1}^{k-1} x_i\right)$$
$$= \frac{1}{k}(x_k + (k-1)u_{k-1})$$
$$= \mu_{k-1} + \frac{1}{k}(x_k - u_{k-1})$$

A shrinking coefficient

Everything here is known and immediately used

Need an entire episode

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-1} R_T$$

What is th $V(s_{t+1})$!

$$G_t \doteq R_{t+1} + \gamma V(s_{t+1})$$

Essentially, this is boot-strappin

From the environment

$$\text{TD(0)}: \quad V(s_t) \leftarrow V(s_t) + \alpha[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

From current value estimate

# TD vs., MC vs., DP

- They all do approximations, but in different components

MC approximates expectation by sampling

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]
\end{aligned}
$$

DP approximates $v_\pi(s_{t+1})$ based on current value estimate

TD approximates $v_\pi(s_{t+1})$ based on current value estimate and $R_{t+1}$ by a single sample

# On-Policy TD control: Sarsa

- Learning the action value function using TD method

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

  - Exploration? Everything we have learnt about it applies here!

**Sarsa (on-policy TD control) for estimating $Q \approx q_*$**

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Loop for each step of episode:
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until $S$ is terminal

# Off-Policy TD control: Q-learning

**Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$**

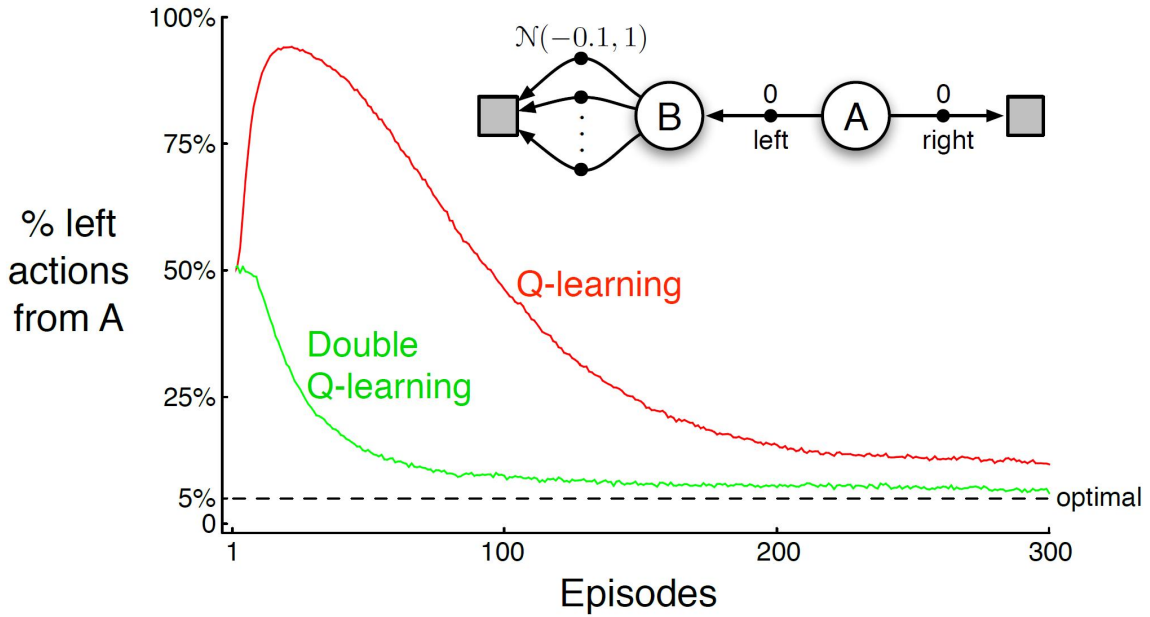
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until $S$ is terminal

# Maximization bias

- We use maximum to choose an action and also treat it as an estimate of the maximum of the true value
  - Q-learning as an example

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$



# Double Q-learning

**Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$**

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\textit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    Choose $A$ from $S$ using the policy $\varepsilon$-greedy in $Q_1 + Q_2$
    Take action $A$, observe $R$, $S'$
    With 0.5 probabilility:
      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
    else:
      $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$
    $S \leftarrow S'$
  until $S$ is terminal

Unbiased estimate for each other!

# A taxonomy of RL solutions

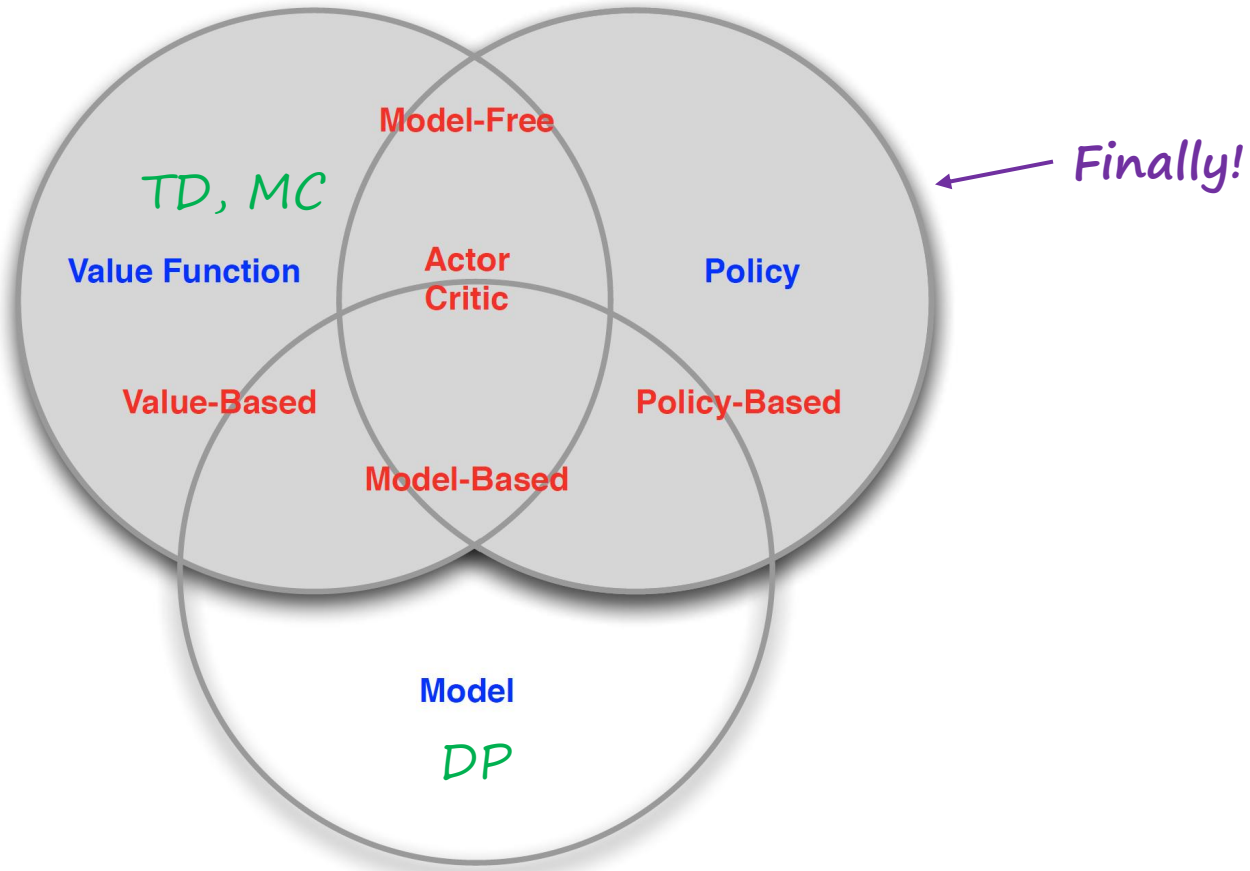



*Figure credit: David Silver, "Introduction to RL"*

# Policy learning as an optimization problem

- Objective function $J(\theta)$
  - In an episodic environment
    - Value of a particular initial state $J(\theta) = v_{\pi_\theta}(s_0)$
  - In a continuous environment
    - Average rate of reward per time step

Then how should we optimize it?

Gradient ascent!

$$J(\boldsymbol{\theta}) \doteq r(\pi) \doteq \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t \mid S_0, A_{0:t-1} \sim \pi]$$

$$= \lim_{t\to\infty} \mathbb{E}[R_t \mid S_0, A_{0:t-1} \sim \pi]$$

$$= \sum_s \mu(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) r$$

Stationary state distribution under policy $\pi(a|s)$, needs an ergodic environment

# Policy gradient theorem

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla v_\pi(s_0) \\
&= \sum_s \left( \sum_{k=0}^{\infty} \Pr(s_0 \to s, k, \pi) \right) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_s \eta(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a)
\end{aligned}
$$

Expected number of visits under policy $\pi(a|s)$, recall state occupancy matrix

Stationary state distribution under policy $\pi(a|s)$

move the constant into the step size of gradient ascent

The return after taking this action

The reason we take this action

# REINFORCE: Monte Carlo policy gradient

- Sample gradients for policy optimization

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi \left[ \sum_a \pi(a|S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})} \right]$$

$$= \mathbb{E}_\pi \left[ q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right]$$

$$= \mathbb{E}_\pi \left[ G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right]$$

Action taken by policy $\pi(a|s)$

Return under policy $\pi(a|s)$

Sample gradient for policy update

A sample of return, needs to wait for the episode

Completely from the environment, no need to know the MDP!

*Gradient: the direction and rate of fastest increase*

1. Encourage the action that leads to a high return
2. Discount the action that is preferred by the current policy

# Construction of policy

- Softmax policy – discrete action space
  - Parameterized state-action pair $\phi(s,a) \in R^d$ — a feature vector
  - A stochastic policy $\pi(a|s) \propto \exp\left(\theta^\top \phi(s,a)\right)$ — a distribution over actions
  - Gradient: $\nabla_\theta \log \pi(a|s) = \phi(s,a) - E_{a\sim\pi}[\phi(s,a)]$
- Gaussian policy – continuous action space
  - Parameterized states $\phi(s) \in R^d$ — a feature vector about the state
  - A stochastic policy $\pi(a|s) \propto \mathrm{N}\left(\theta^\top \phi(s),\ \sigma^2\right)$ — a scalar action can be extended to vectorized actions
  - Gradient: $\nabla_\theta \log \pi(a|s) = \dfrac{\left(a - \theta^\top \phi(s)\right)}{\sigma^2}\phi(s)$

# REINFORCE with baseline

- REINFORCE is a Monte Carlo method in nature
  - Suffer from a high variance (just as in SGD)
  - Use a baseline function for variance reduction

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \Big( q_\pi(s,a) - \boxed{b(s)} \Big) \nabla \pi(a|s,\boldsymbol{\theta})$$

Any function independent of action $a$

  - Typical choice: $b(s) = v(s)$, which leads to the advantage function

$$A_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$$

# Actor-Critic methods

- One-step actor-critic

Let's bootstrap this

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big(G_{t:t+1} - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$$

Critic evaluates actions

- Actor takes actions

Only need one step sampling from environment

Based on the current value model estimate, everything we have learnt so far applies (e.g., MC/TD)!

**A source of bias due to bootstrapping!**

# Reinforcement learning in practice is challenging

- Learning by trial and error is expensive
  - We need an environment to repeatedly interact with


Environment defined by

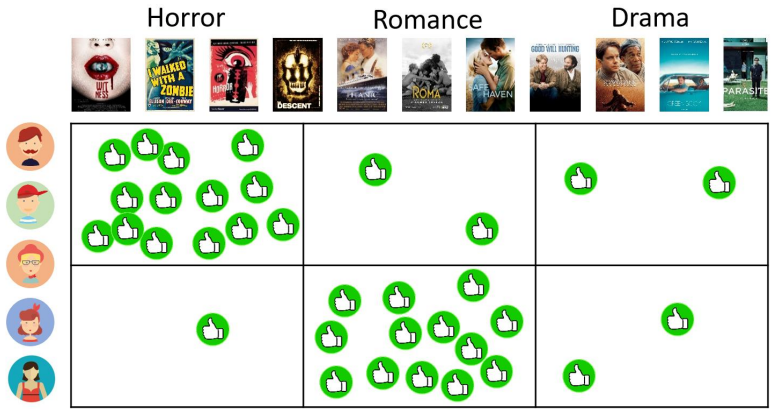
game rules

physical rules

*Simulation is easy*

# Reinforcement learning in practice is challenging

- Learning by trial and error is expensive
  - We need an environment to repeatedly interact with



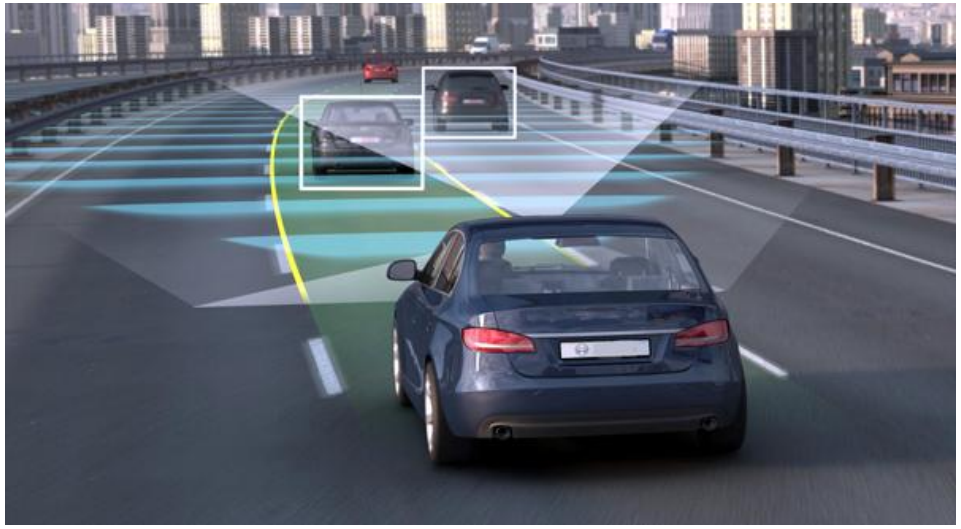

Environment defined by

Users' behaviors

*Simulation is hard!*

# Reinforcement learning in practice is challenging

- There are also safety, privacy, and ethic concerns in exploration
  - We are dealing with unknown unknowns
  - We are learning from explorations



# Thank you!

Q&A