Lecture 24 and 25: RL in a nutshell

Bolei Zhou

The Chinese University of Hong Kong

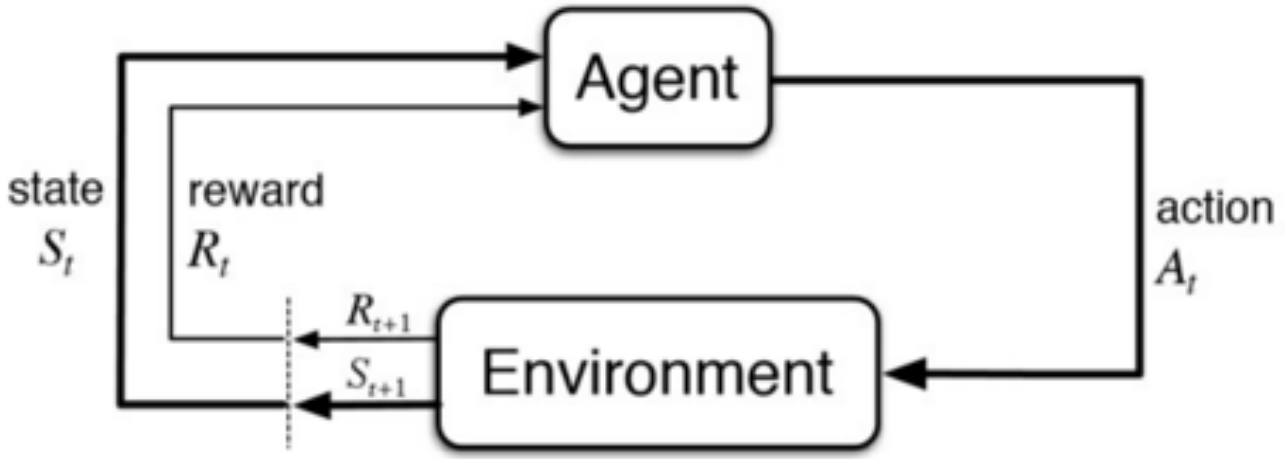
Outline

- Basics of RL
- MDP and tabular solution methods
- Value function approximation
- Policy optimization
- Other topics: model-based RL, imitation learning, RL system design
- Open problems

Basics of RL

Definition of RL:

a computational approach to learning whereby an agent tries to maximize the total amount of reward it receives while interacting with a complex and uncertain environment.



Basics of RL

Properties of RL

- Sequential data as input (not i.i.d)
- Trial and error exploration (balance between exploration and exploitation)
- Actions may have long-term consequence (rewards are delayed)

Basics of RL

An RL agent includes one or more of these components:

- Policy function: how action is selected
  - Stochastic policy: probability over actions (20% take LEFT, 80% take RIGHT)
  - Deterministic policy: take action LEFT
- Value function: how good is each state of action

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \qquad q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

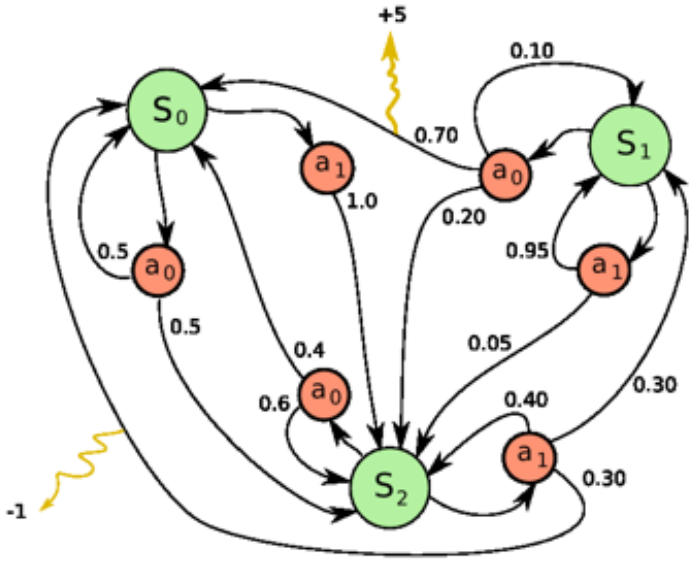
- Model: state transition of the environment

$$P^a_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] \qquad R^a_s = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$$

Markov Decision Processes (MDPs)

Definition of MDP

1. P^a is dynamics/transition model for each action
   $P(S_{t+1} = s' | S_t = s, A_t = a)$
2. $\mathcal{R}$ is a reward function $R(S_t = s, A_t = a) = \mathbb{E}[R_t | S_t = s, A_t = a]$
3. Discount factor $\gamma \in [0, 1]$



Bellman Expectation Equation

- The state-value function can be decomposed into immediate reward plus discounted value of the successor state,

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s]$$

- The action-value function can similarly be decomposed

$$Q_\pi(s, a) = E_\pi[R_{t+1} + \gamma Q_\pi(s_{t+1}, A_{t+1}) | s_t = s, A_t = a]$$

Policy Evaluation and Control in MDP

- Policy evaluation: Given an MDP, evaluate the value of a policy
- Control: Given an MDP, find the optimal policy which leads to optimal value

MDP Policy Evaluation

1. Problem: Evaluate a given policy π for a MDP
2. Output the value function under policy v_π
3. Solution: iteration on Bellman expectation backup
4. Synchronous backup algorithm:
   1. At each iteration k+1
      Bellman expectation backup on MDP

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_k(s'))$$

Or in the form of Bellman expectation backup on MRP
$< \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma >$

$$v_{k+1}(s) = R^\pi(s) + \gamma P^\pi(s'|s) v_k(s')$$

MDP Control

- Objective: Find the optimal policy, given a MDP
- Policy iteration and value iteration

1. Problem: Find the optimal policy π
2. Solution: iteration on the Bellman optimality backup
3. Value Iteration algorithm:
   1. initialize $k = 1$ and $v_0(s) = 0$ for all states s
   2. For $k = 1 : H$
      1. for each state s

$$q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_k(s')$$

$$v_{k+1}(s) = \max_a q_{k+1}(s,a)$$

      2. $k \leftarrow k + 1$
4. To retrieve the optimal policy after the value iteration:

$$\pi(s) = \arg\max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{k+1}(s')$$

https://github.com/metalbubble/RLexample/tree/master/MDP

Model-free prediction and control

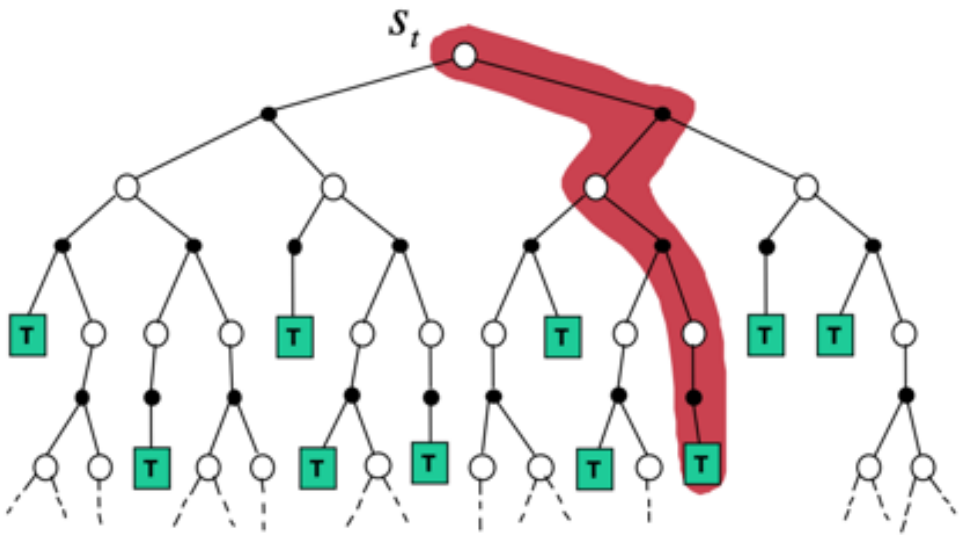
- Previously we assume R and P of the MDP are given so we can compute the state-action value of a policy

$$q_{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{\pi_i}(s')$$

- What to do if neither of them are unknow (model-free)
  - Monte Carlo (MC): Simulation-based
  - Temporal Difference (TD): Bootstrapping

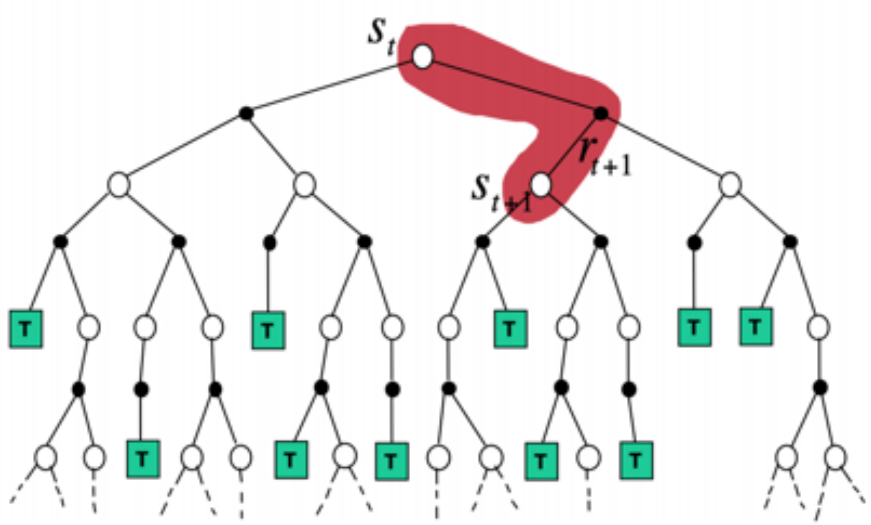
Policy Evaluation using Monte-Carlo and Temporal Difference

$$v(S_t) \leftarrow v(S_t) + \alpha(G_t - v(S_t))$$



Monte-Carlo Simulation

$$TD(0) : v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(S_{t+1}) - v(S_t))$$



Temporal Difference (1-step to n-step)

On-policy control: SARSA

- Estimating action value function Q using TD

ϵ-greedy policy for one step, then bootstrap the action value function:

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

The update is done after every transition from a nonterminal state S_t

TD target $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

Off-policy learning

1. Following behaviour policy $\mu(a|s)$ to collect data

$$S_1, A_1, R_2, ..., S_T \sim \mu$$

Update π using $S_1, A_1, R_2, ..., S_T$

2. It leads to many benefits:
   1. Learn about optimal policy while following exploratory policy
   2. Learn from observing humans or other agents
   3. Re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_{t-1}$

Off-policy control: Q-Learning

1. The target learning policy π is greedy on $Q(s, a)$

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

2. The behavior policy could be totally random, but we let it improve, thus the behavior policy μ is ϵ-greedy on $Q(s, a)$
3. Thus Q-learning target:

$$\begin{aligned} R_{t+1} + \gamma Q(S_{t+1}, A') &= R_{t+1} + \gamma Q(S_{t+1}, \arg\max Q(S_{t+1}, a')) \\ &= R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') \end{aligned}$$

4. Thus the Q-Learning update becomes

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

Comparison of Sarsa and Q-Learning

1. Sarsa: On-Policy TD control

   Choose action A_t from S_t using policy derived from Q with ϵ-greedy

   Take action A_t, observe R_{t+1} and S_{t+1}

   Choose action A_{t+1} from S_{t+1} using policy derived from Q with ϵ-greedy

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

2. Q-Learning: Off-Policy TD control

   Choose action A_t from S_t using policy derived from Q with ϵ-greedy

   Take action A_t, observe R_{t+1} and S_{t+1}

   Then 'imagine' A_{t+1} as $\arg\max Q(S_{t+1}, a')$ in the update target

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

Outline

- Basics of RL
- MDP and tabular solution methods
- Value function approximation
- Policy optimization
- Other topics: model-based RL, imitation learning, RL system design
- Open problems

Scaling Up RL: Function Approximation

- Previous tabular methods only work for RL problems with small number of states (lookup table representation)
- Many real-world problems have a large number of states
- Represent a value function with a parameterized function

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$
$$\hat{\pi}(a, s, \mathbf{w}) \approx \pi(a|s)$$

Scaling Up RL: Value Function Approximation

1. Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

2. Minimize the MSE (mean-square error) between approximate action-value and true action-value (assume oracle)

$$J(\mathbf{w}) = \mathbb{E}_\pi[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2]$$

3. Stochastic gradient descend to find a local minimum

$$\Delta \mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$$

Scaling Up RL: Value Function Approximation

Same to the prediction, there is no oracle for the true value $q_\pi(S, A)$, so we substitute a target

1. For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha\big(G_t - \hat{q}(S_t, A_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

2. For Sarsa, the target is the TD target $R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha\big(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

3. For Q-learning, the target is the TD target $R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w})$

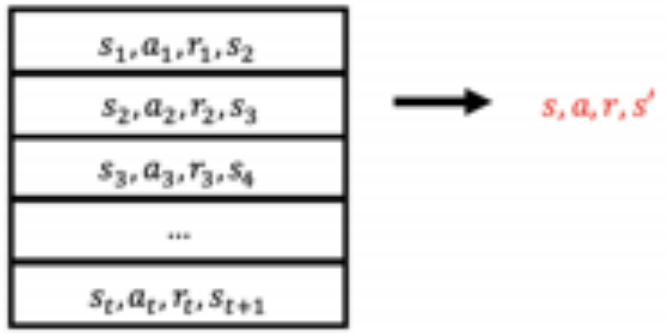
$$\Delta \mathbf{w} = \alpha\big(R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

Deep Q Learning

- Two of the issues causing problems:
  - Correlations between samples
  - Non-stationary targets
- Deep Q-learning (DQN) addresses both challenges by
  - Experience replay
  - Fixed Q targets

Deep Q Learning: Experience Replay

1. To reduce the correlations among samples, store transition (s_t, a_t, r_t, s_{t+1}) in replay memory $\mathcal{D}$



2. To perform experience replay, repeat the following
   1. sample an experience tuple from the dataset: $(s, a, r, s') \sim \mathcal{D}$
   2. compute the target value for the sampled tuple: $r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w})$
   3. use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha \Big(r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}) \Big) \nabla_{\mathbf{w}} \hat{Q}(s, a, \mathbf{w})$$

Deep Q Learning: Fixed Targets

1. To help improve stability, fix the target weights used in the target calculation for multiple updates
2. Let a different set of parameter $\mathbf{w}^-$ be the set of weights used in the target, and $\mathbf{w}$ be the weights that are being updated
3. To perform experience replay with fixed target, repeat the following
   1. sample an experience tuple from the dataset: $(s, a, r, s') \sim \mathcal{D}$
   2. compute the target value for the sampled tuple: $r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}^-)$
   3. use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha \Big(r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \Big) \nabla_{\mathbf{w}} \hat{Q}(s, a, \mathbf{w})$$

Q target

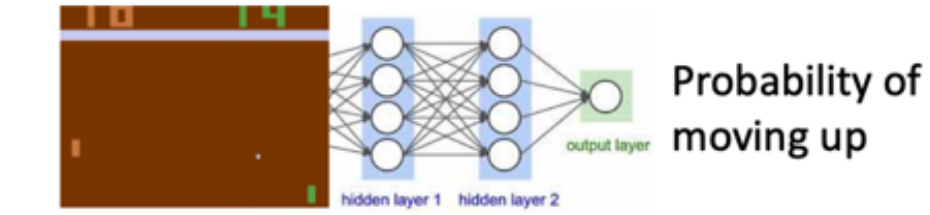
Q estimation

Outline

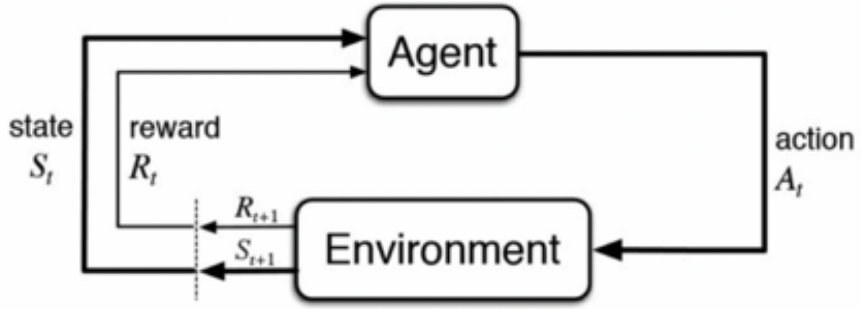
- Basics of RL
- MDP and tabular solution methods
- Value function approximation
- Policy optimization
- Other topics: model-based RL, imitation learning, RL system design
- Open problems

Policy Optimization

- Value-based RL: learn value function first, then make decision based on action value function
- Policy-based RL: optimize the policy



$$\pi_\theta(A_t|S_t)$$



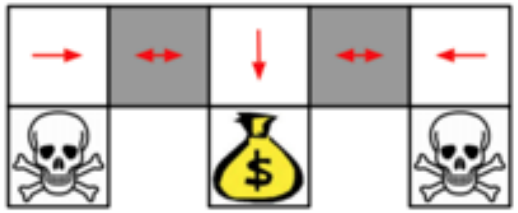
Policy Optimization

1. Value-based RL
    1. to learn value function
    2. implicit policy based on the value function
2. Policy-based RL
    1. no value function
    2. to learn policy directly
3. Actor-critic
    1. to learn both policy and value function

1. In value-based RL deterministic policy is generated directly from the value function using greedy $a_t = \arg\max_a Q(a, s_t)$
2. In policy optimization, we have stochastic policy output from $\pi_\theta(a|s)$ where θ is the policy parameter to optimize

Policy Optimization: Stochastic Policy

1. Two-player game
2. What is the best policy?
   1. A deterministic policy is easily beaten
   2. Uniform random policy is the optimal (Nash equilibrium)



1. Policy-based RL can learn the optimal stochastic policy
2. An optimal stochastic policy will randomly move E or W in two grey states

$$\pi_\theta(\text{wall to N and W, move E}) = 0.5$$
$$\pi_\theta(\text{wall to N and W, move W}) = 0.5$$

Policy Optimization: Objective

$$\theta^* = \arg\max_\theta J(\theta) \text{ where } J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)]$$

If J(θ) is differentiable, we can use gradient-based methods

- Gradient ascend
- Conjugate gradient
- quasi-newton

Some derivative-free black-box optimization methods:

- Cross-entropy method (CEM)
- Hill climbing
- Evolution algorithm

Policy Optimization: Objective

- Policy optimization objective:

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta) R(\tau)$$

- Policy gradient:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_\tau P(\tau;\theta) R(\tau) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} \\
&= \sum_\tau P(\tau;\theta) R(\tau) \nabla_\theta \log P(\tau;\theta)
\end{aligned}$$

log derivative trick

Policy Optimization: Decomposition

1. Approximate with empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \nabla_\theta \log P(\tau_i; \theta)$$

2. Decompose $\nabla_\theta \log P(\tau; \theta)$

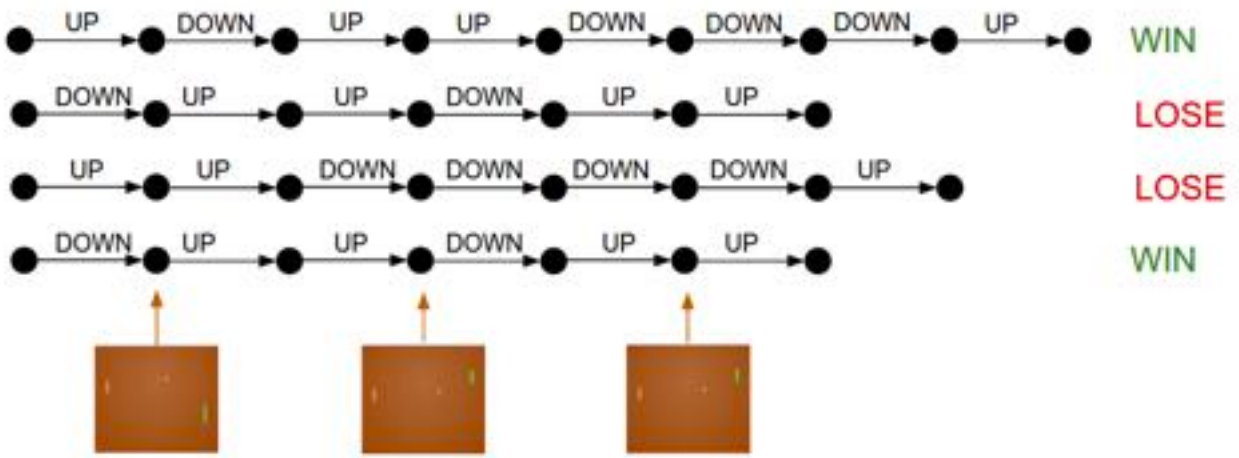
$$\begin{aligned}
\nabla_\theta \log P(\tau; \theta) &= \nabla_\theta \log \left[\mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \right] \\
&= \nabla_\theta \left[\log \mu(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t) \right] \\
&= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)
\end{aligned}$$

Policy Optimization

- So we have the following approximate gradient from several simulations

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i)$$

Unbiased but very noisy



Reduce Variance of Policy Gradient

- Causality: policy at time t' cannot affect the reward at time t<t'

$$\nabla_\theta \mathbb{E}[R] \approx \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T-1} G_t^{(i)} \cdot \nabla_\theta \log \pi_\theta(a_t^i | s_t^i)$$

- Baseline to reduce the variance

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R] = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} \big(G_t - b(s_t) \big) \cdot \nabla_\theta \log \pi_\theta(a_t | s_t) \Big]$$

$$b(s_t) = \mathbb{E}[r_t + r_{t+1} + \ldots + r_{T-1}]$$

REINFORCE: A Monte-Carlo policy gradient

- Widely used RL algorithm:

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Reducing Variance using a Critic

- Previously G is a sample from Monte Carlo simulation, which is unbiased but noisy estimate of Q(s, a)

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[R] = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

- Instead we can use a critic to estimate the action-value function

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[R] = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

Actor-Critic Policy Gradient

- Actor-critic policy gradient
  - Actor: the policy function used to generate the action
  - Critic: the value function used to evaluate the reward of the actions
- Actor-critic algorithms maintain two sets of parameters
  - Actor: Updates policy parameters θ, in direction suggested by critic
  - Critic: Updates action-value function parameters w

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[R] = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

Actor-Critic Policy Gradient

1. Using a linear value function approximation: $Q_{\mathbf{w}}(s,a) = \psi(s,a)^T \mathbf{w}$
   1. Critic: update $\mathbf{w}$ by a linear TD(0)
   2. Actor: update θ by policy gradient

Algorithm 1 Simple QAC

---

for each step **do**
  generate sample s, a, r, s', a' following π_θ
  $\delta = r + \gamma Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$ #TD error
  $\mathbf{w} \leftarrow \mathbf{w} + \beta \delta \psi(s, a)$
  $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_{\mathbf{w}}(s, a)$
end for

Advantage Actor-Critic Policy Gradient

- Advantage function:

$$A^{\pi,\gamma}(s,a) = Q^{\pi,\gamma}(s,a) - V^{\pi,\gamma}(s)$$

- Value function serves as a baseline for Q

$$\begin{aligned} V^{\pi,\gamma}(s) &= \mathbb{E}_{\pi}[r_1 + \gamma r_2 + ... | s_1 = s] \\ &= \mathbb{E}_{a\sim\pi}[Q^{\pi,\gamma}(s,a)] \end{aligned}$$

- Then the policy gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi,\gamma}(s,a)]$$

Example code: https://github.com/cuhkrlcourse/RLexample/blob/master/policygradient/ac-pong-pytorch.py

Different variants of policy gradient

1. The policy gradient has many forms

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) G_t] \text{ — REINFORCE} \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\mathbf{w}}(s,a)] \text{ — Q Actor-Critic} \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\mathbf{w}}(s,a)] \text{— Advantage Actor-Critic} \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \delta] \text{— TD Actor-Critic}\end{aligned}$$

2. Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q^\pi(s,a)$, $A^\pi(s,a)$, or $V^\pi(s,a)$

State of the art extension on PG

State-of-the-art RL methods: One line on extension of PG

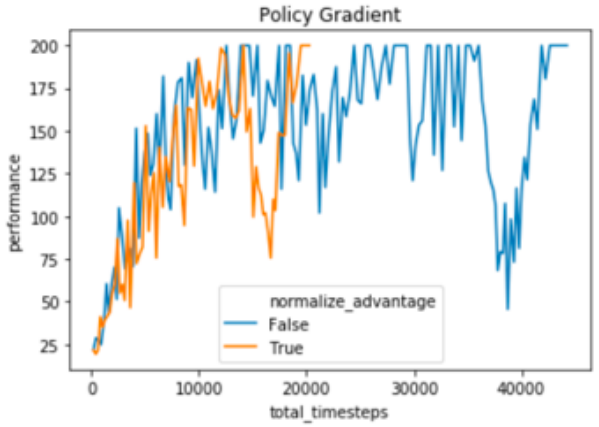
- **DDPG**: TP Lillicrap, et al (2015). Continuous control with deep reinforcement learning
- **TRPO**: Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization: deep RL with natural policy gradient and adaptive step size
- **PPO**: Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms: deep RL with importance sampled policy gradient

Challenges with policy gradient

- Poor sample efficiency due to on-policy learning

$$\nabla_\theta J(\theta) = \mathbb{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r(s,a)]$$

- Large policy update or improper step size destroy the training
  - In RL, step too far → bad policy → bad data collection
  - May not be able to recover from a bad policy, collapse in performance



Improvements over the policy objective

- Off-policy learning using importance sampling

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)] = \mathbb{E}_{\tau \sim \hat{\pi}}[\frac{\pi_\theta(\tau)}{\hat{\pi}(\tau)} r(\tau)]$$

- Increase the robustness with trust region (TRPO)

$$\text{maximize}_{\ \theta} \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t \right]$$

$$\text{subject to } KL(\pi_{\theta_{old}}(.|s_t)||\pi_\theta(.|s_t)) < \delta$$

- Clip the estimated advantage function (PPO)

$$L_t(\theta) = \min\left(r_t(\theta)\hat{A}_t, \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t\right)$$

Importance Sampling

1. We can turn PG into off-policy learning using importance sampling
2. **Importance sampling**: IS calculates the expected value of $f(x)$ where x has a data distribution p
    1. we can sample data from another distribution q and use the probability ratio between p and q to re-calibrate the result

$$\mathbb{E}_{x\sim p}[f(x)] = \int p(x)f(x)dx = \int q(x)\frac{p(x)}{q(x)}f(x)dx = \mathbb{E}_{x\sim q}[\frac{p(x)}{q(x)}f(x)]$$

    2. Code example of IS: link
3. Using important sampling in policy objective

$$J(\theta) = \mathbb{E}_{a\sim\pi_\theta}[r(s,a)] = \mathbb{E}_{a\sim\hat{\pi}}[\frac{\pi_\theta(s,a)}{\hat{\pi}(s,a)}r(s,a)]$$

Increasing the Robustness with Trust Region

1. Thus our objective with trust region becomes, to maximize

$$J_{\theta_{old}}(\theta) = \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} R_t\right]$$

$$\text{subject to } KL(\pi_{\theta_{old}}(.|s_t)||\pi_\theta(.|s_t)) \leq \delta$$

2. In the trust region, we limit our parameter search within a region controlled by δ. This is the intuition behind algorithms TRPO and PPO

Gradient ascend

Trust region

Trust Region Policy Optimization (TRPO)

Algorithm 3 Trust Region Policy Optimization

Input: initial policy parameters θ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$

Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

Form sample estimates for

- policy gradient $\hat{g}_k$ (using advantage estimates)
- and KL-divergence Hessian-vector product function $f(v) = \hat{H}_k v$

Use CG with n_{cg} iterations to obtain $x_k \approx \hat{H}_k^{-1} \hat{g}_k$

Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$

Perform backtracking line search with exponential decay to obtain final update

$$\theta_{k+1} = \theta_k + \alpha^j \Delta_k$$

end for

Proximal Policy Optimization (PPO)

1. The loss function in the Natural Gradient and TRPO

$$\text{maximize}_\theta \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t\right]$$

$$\text{subject to } \mathbb{E}_t[KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]] \leq \delta$$

2. It also can be written into an unconstrained form,

$$\text{maximize}_\theta \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t\right] - \beta\mathbb{E}_t[KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]]$$

PPO with adaptive KL penality loss

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta || \theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \leq \delta / 1.5$ **then**
    $\beta_{k+1} = \beta_k / 2$
  end if
end for

PPO with clipping loss

1. Clipping serves as a regularizer by removing incentives for the policy to change dramatically

$$L^{CLIP}(\theta) = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)$$

2. When the advantage is positive, we encourage the action thus $\pi_\theta(a|s)$ increases,

$$L(\theta; \theta_{old}) = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, (1+\epsilon)\right)\hat{A}_t$$

3. When the advantage is negative, we discourage the action thus $\pi_\theta(a|s)$ decreases,

$$L(\theta; \theta_{old}) = \max\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, (1-\epsilon)\right)\hat{A}_t$$

Q-Learning -> DDPG -> TD3 -> SAC

1. Value based RL methods starting from Q-learning are also being developed over the years
2. **DDPG**: Deterministic Policy Gradient Algorithms, Silver et al. ICML 2014
3. **TD3**: Addressing Function Approximation Error in Actor-Critic Methods, Fujimoto et al. ICML 2018
4. **SAC**: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, Haarnoja et al. ICML 2018

Deep Deterministic Policy Gradient (DDPG)

1. Motivation: how to extend DQN to the environment with continuous action space?
2. DDPG is very similar to DQN, which can be considered as a continuous action version of DQN

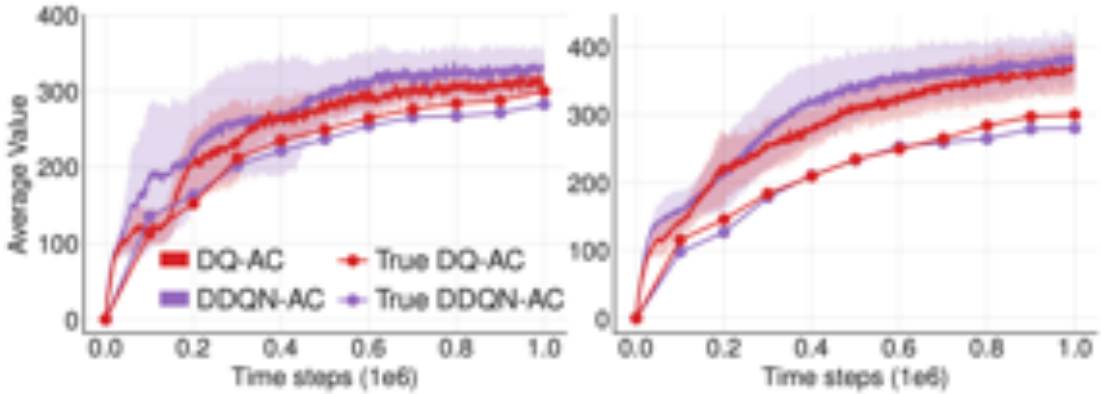
$$\textbf{DQN}: a^* = \arg\max_a Q^*(s,a)$$

$$\textbf{DDPG}: a^* = \arg\max_a Q^*(s,a) \approx Q_\phi(s, \mu_\theta(s))$$

1. a deterministic policy $\mu_\theta(s)$ directly gives the action that maximizes $Q_\phi(s,a)$
2. as action a is continious we assume Q-function $Q_\phi(s,a)$ is differentiable with respect to a

Twin Delayed DDPG (TD3)

1. One drawback of DDPG is that the learned Q-function sometimes dramatically overestimate Q-values, which then leads to the policy breaking



(a) Hopper-v1 (b) Walker2d-v1

Twin Delayed DDPG (TD3)

1. TD3 concurrently learns two Q-functions, Q_{ϕ_1} and Q_{ϕ_2}, by mean square Bellman error minimization
2. Both Q-functions use a single target, calculated using whichever of the two Q-functions gives a smaller value as the following **Q-target**:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,targ}}(s', a_{TD3}(s'))$$

3. Target policy smoothing works as follows

$$a_{TD3}(s') = clip(\mu_{\theta,targ}(s') + clip(\epsilon, -c, c), a_{low}, a_{high}), \epsilon \sim \mathcal{N}(0, \sigma)$$

    1. It serves as a regularizer: to avoid the incorrect sharp peak for some action output by the policy

Author's Pytorch implementation (very clean implementation!): https://github.com/sfujim/TD3/

Soft Actor-Critic (SAC)

1. SAC optimizes a stochastic policy in an off-policy way, which unifies stochastic policy optimization and DDPG-style approaches
2. SAC incorporates **entropy regularization**
3. Entropy is a quantity which measures how random a random variable is, $H(P) = E_{x\sim P}[-\log P(x)]$
4. Entropy-regularized RL: the policy is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy

$$\pi^* = \arg\max E_{\tau\sim\pi}[\sum_t \gamma^t \left(R(s_t, a_t, s_{t+1}) + \alpha H(\pi(.|s_t))\right)]$$

Soft Actor-Critic (SAC)

1. SAC concurrently learns a policy π_θ and two Q-functions Q_{ϕ_1} and Q_{ϕ_2}
2. Like in TD3, the shared target makes use of the clipped double-Q trick and both Q-functions are learned with MSBE minimization

$$L(\phi_i, \mathcal{D}) = E[(Q_\phi(s,a) - y(r,s',d)^2] \tag{2}$$

$$y(r,s',d) = r + \gamma(1-d)\Big(\min_{j=1,2} Q_{\phi_{\text{targ}\,j}}(s', \hat{a}') - \alpha \log \pi_\theta(\hat{a}'|s')\Big), \tag{3}$$

$$\hat{a}' \sim \pi_\theta(.|s'). \tag{4}$$

3. Policy is learned through maximizing the expected future return plus expected future entropy, thus maximize $V^\pi(s)$

$$\begin{aligned} V^\pi(s) &= E_{a\sim\pi}[Q^\pi(s,a)] + \alpha H(\pi(.|s)) \\ &= E_{a\sim\pi}[Q^\pi(s,a) - \alpha \log \pi(a|s)]. \end{aligned}$$

Soft Actor-Critic (SAC)

1. We reparameterize a as sample from a squashed Gaussian policy

$$\hat{a}_\theta(s, \epsilon) = \tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \epsilon), \epsilon \sim \mathcal{N}(0, I).$$

2. Reparameterization trick allows us to rewrite the expectation over actions (which contains a pain point: the distribution depends on the policy parameters) into an expectation over noise (which removes the pain point: the distribution now has no dependence on parameters):

$$\begin{aligned} & E_{a\sim\pi_\theta}[Q^{\pi_\theta}(s, a) - \alpha \log \pi_\theta(a|s)] \\ =& E_{\epsilon\sim\mathcal{N}}[Q^{\pi_\theta}(s, \hat{a}_\theta(s, \epsilon)) - \alpha \log \pi_\theta(\hat{a}_\theta(s, \epsilon)|s)] \end{aligned}$$

3. The policy is thus optimized as

$$\max_\theta E_{s\sim\mathcal{D}, \epsilon\sim\mathcal{N}}[\min_{j=1,2} Q_{\phi_j}(s, \hat{a}_\theta(s, \epsilon)) - \alpha \log \pi_\theta(\hat{a}_\theta(s, \epsilon)|s)] \quad (5)$$

Reparameterization Trick

1. Let's say we want to compute the gradient of the expected value of the function $\nabla_\theta E_{x\sim p_\theta(x)}[f(x)]$, the difficulty is that x depends on the the distribution with parameter θ
2. We can rewrite the samples of the distribution p_θ in terms of a noise variable ϵ which is independent of θ

$$\epsilon \sim q(\epsilon) \tag{6}$$

$$x = g_\theta(\epsilon) \tag{7}$$

$$\nabla_\theta E_{x\sim p_\theta(x)}[f(x)] = \nabla_\theta E_{\epsilon\sim q(\epsilon)}[f(g_\theta(\epsilon))] \tag{8}$$

$$= E_{\epsilon\sim q(\epsilon)}[\nabla_\theta f(g_\theta(\epsilon))] \tag{9}$$

1. where x is reparameterized as a function of ϵ and the stochasticity of p_θ is replaced by the distribution $q(\epsilon)$, which could be just a standard Gaussian $\mathcal{N}(0,1)$, thus $g_\theta(\epsilon) = \mu_\theta + \epsilon\sigma_\theta$ where $\epsilon \sim \mathcal{N}(0,1)$

http://stillbreeze.github.io/REINFORCE-vs-Reparameterization-trick/

Log Derivative Trick vs. Reparameterization Trick

Problem: $\nabla_\theta \mathbb{E}_{x \sim p_\theta(x)}[f(x)]$

Log Derivative Trick:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{x \sim p_\theta(x)}[f(x)] &= \nabla_\theta \int f(x) p_\theta(x) dx \\
&= \int f(x) \nabla_\theta p_\theta(x) dx \\
&= \int f(x) p_\theta(x) \nabla_\theta \log p_\theta(x) dx \\
&= \mathbb{E}_{x \sim p_\theta(x)}[f(x) \nabla_\theta \log p_\theta(x)] \\
&\approx \frac{1}{N} \sum_{i=1}^{N} f(x_i) \nabla_\theta \log p_\theta(x_i)
\end{aligned}$$

Reparameterization Trick:

$$\begin{aligned}
\varepsilon &\sim q(\varepsilon) \\
x &= g_\theta(\varepsilon) \\
\nabla_\theta \mathbb{E}_{x \sim p_\theta(x)}[f(x)] &= \nabla_\theta \mathbb{E}_{\varepsilon \sim q(\varepsilon)}[f(g_\theta(\varepsilon))] \\
&= \mathbb{E}_{\varepsilon \sim q(\varepsilon)}[\nabla_\theta f(g_\theta(\varepsilon))] \\
&\approx \frac{1}{N} \sum_{i=1}^{N} (\nabla_\theta f(g_\theta(\varepsilon_i)))
\end{aligned}$$

Properties	REINFORCE	Reparameterization
Differentiability requirements	Can work with a non-differentiable model	Needs a differentiable model
Gradient variance	High variance; needs variance reduction techniques	Low variance due to implicit modeling of dependencies
Type of distribution	Works for both discrete and continuous distributions	In the current form, only valid for continuous distributions
Family of distribution	Works for a large class of distributions of x	It should be possible to reparameterize x as done above

Log Derivative Trick:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{x\sim p_\theta(x)}[f(x)] &= \nabla_\theta \int f(x)p_\theta(x)dx \\
&= \int f(x)\nabla_\theta p_\theta(x)dx \\
&= \int f(x)p_\theta(x)\nabla_\theta \log p_\theta(x)dx \\
&= \mathbb{E}_{x\sim p_\theta(x)}[f(x)\nabla_\theta \log p_\theta(x)] \\
&\approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)\nabla_\theta \log p_\theta(x_i)
\end{aligned}$$

Reparameterization Trick:

$$\begin{aligned}
\varepsilon &\sim q(\varepsilon) \\
x &= g_\theta(\varepsilon) \\
\nabla_\theta \mathbb{E}_{x\sim p_\theta(x)}[f(x)] &= \nabla_\theta \mathbb{E}_{\varepsilon\sim q(\varepsilon)}[f(g_\theta(\varepsilon))] \\
&= \mathbb{E}_{\varepsilon\sim q(\varepsilon)}[\nabla_\theta f(g_\theta(\varepsilon))] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}(\nabla_\theta f(g_\theta(\varepsilon_i)))
\end{aligned}$$

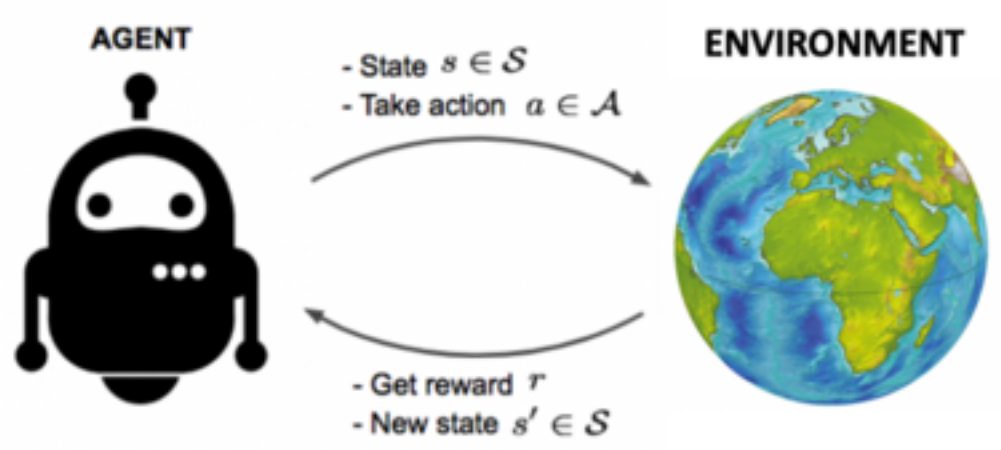
More resource on SOTA RL algorithms

- SpinningUp: Nice implementations and summary of the algorithms from OpenAI:
- https://spinningup.openai.com

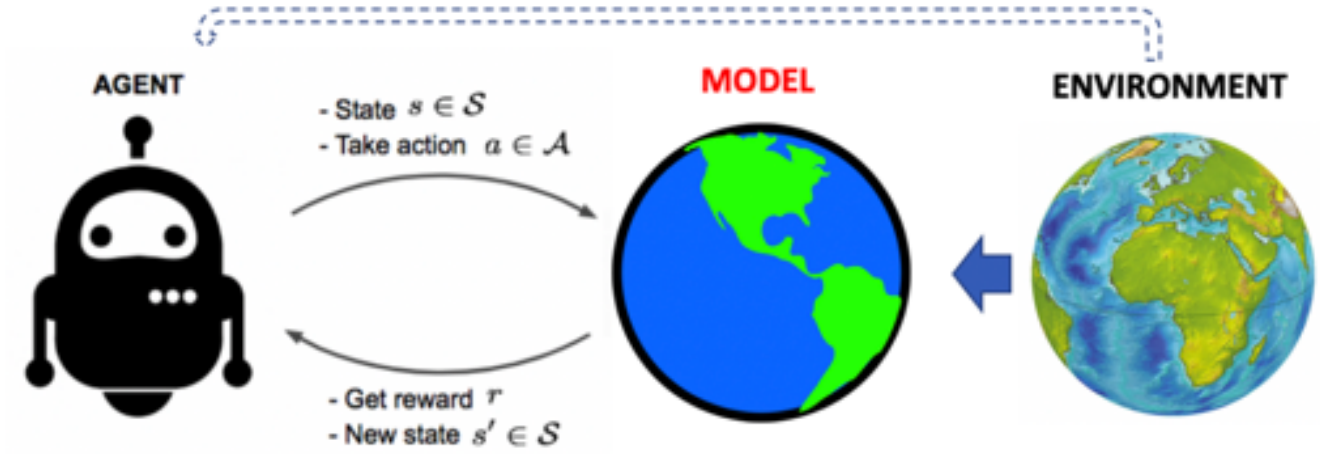
Outline

- Basics of RL
- MDP and tabular solution methods
- Value function approximation
- Policy optimization
- Other topics: model-based RL, imitation learning, RL system design
- Open problems

Model-based RL



Model-free RL



Model-based RL

Model-based RL

- Advantage: much better sample efficient

- Disadvantage: two sources of approximation error (model estimation and function approximation)

Model-based RL

- Model-based value-based RL

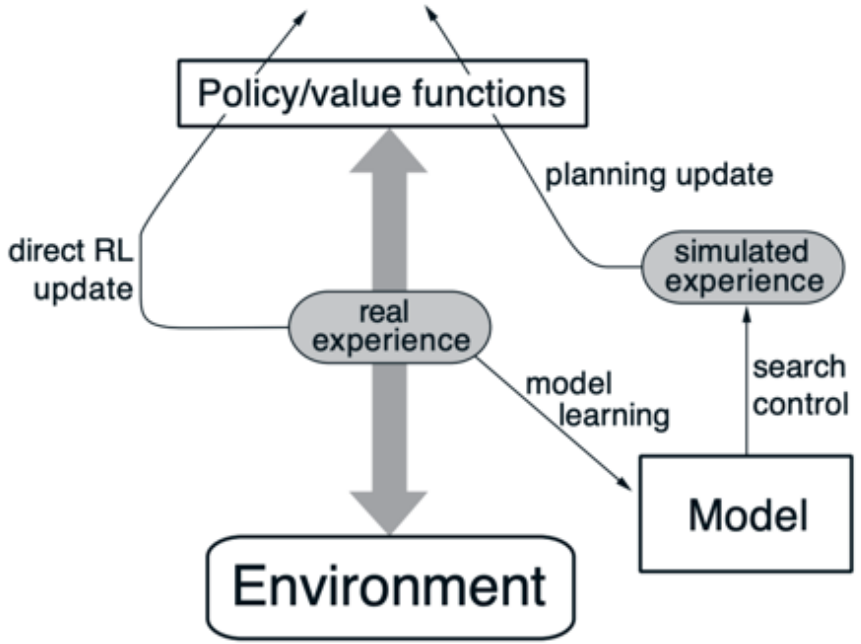
$$\text{model} \rightarrow \text{simulated environment} \xrightarrow{backups} \text{values} \rightarrow \text{policy}$$

- Model-based policy-based RL

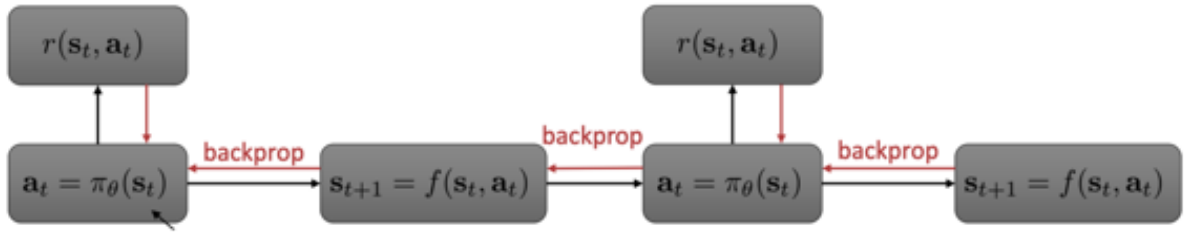
$$\text{model} \xrightarrow{improves} \text{policy}$$

Model-based RL

Value-based RL: Dyna



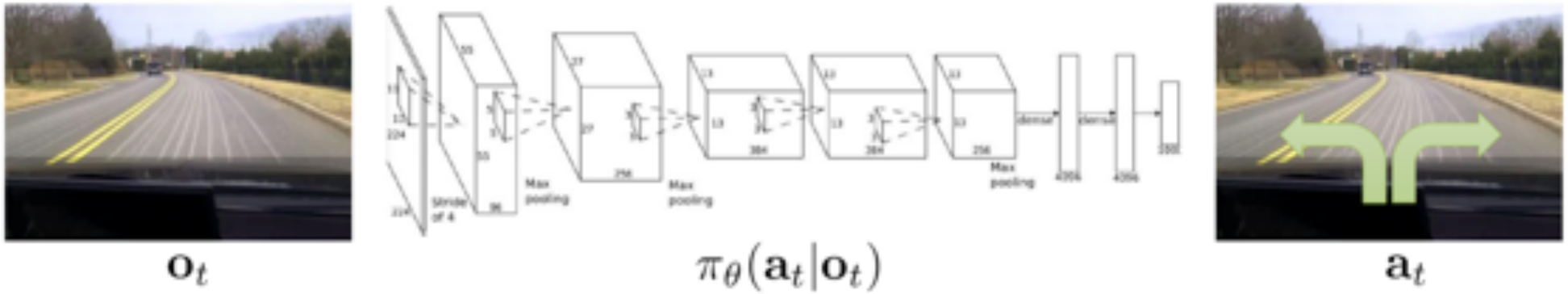
Policy-based RL: model learning from trajectory optimization



Many robotic applications

Imitation Learning

- Supervised learning of the policy network



Limitation of Supervised Imitation Learning

- Misalignment between data distribution and policy distribution
- Handle off-course situations

can we make $p_{\text{data}}(\mathbf{o}_t) = p_{\pi_\theta}(\mathbf{o}_t)$?

idea: instead of being clever about $p_{\pi_\theta}(\mathbf{o}_t)$, be clever about $p_{\text{data}}(\mathbf{o}_t)$!

DAgger: Dataset Aggregation

goal: collect training data from $p_{\pi_\theta}(\mathbf{o}_t)$ instead of $p_{\text{data}}(\mathbf{o}_t)$

how? just run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

but need labels $\mathbf{a}_t$!

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

or ask other slow but more accurate algorithms

Unifying IL and RL

Imitation Learning

- Requires demonstrations
- Issue of distributional shift
- Simple stable supervised learning
- Only as good as the demo

Reinforcement Learning

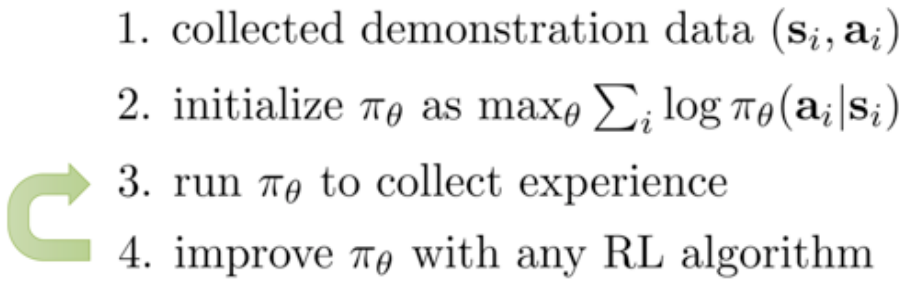
- Requires reward function
- Must address exploration
- Potentially non-convergent
- Can become superhuman good

Can we get the best of both worlds?

What if we can have both demonstrations and rewards

Unifying IL and RL

- Pretrain with IL then finetune with RL



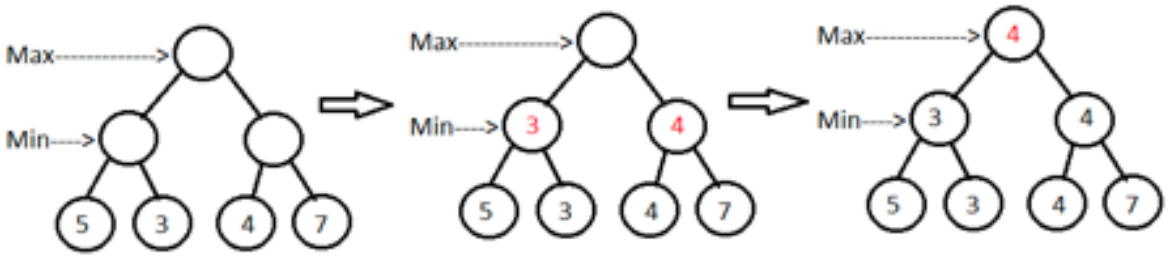
- IL as auxiliary loss function

$$\underbrace{E_{\pi_\theta}[r(\mathbf{s},\mathbf{a})]}_{\text{RL}} + \lambda \underbrace{\sum_{(\mathbf{s},\mathbf{a})\in\mathcal{D}_{\text{demo}}} \log \pi_\theta(\mathbf{a}|\mathbf{s})}_{\text{Imitation learning}}$$

- Off-policy reinforcement learning
  - Policy gradient with important sampling
  - Q-learning (put demonstration in replay buffer)

RL in Competitive Games

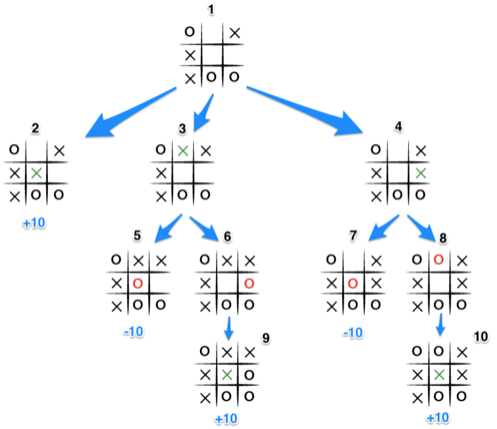
- Minimax Value: two players' decision on a game tree



- Minimax value function:

$$v_*(s) = \max_{\pi^1} \min_{\pi^2} v_\pi(s)$$

- Minimax Search on game tree



RL in Competitive Games

- Monte-Carlo Search
  - Given a model $\mathcal{M}_v$ and a simulation policy π
  - For each action $a \in \mathcal{A}$
    - Simulate K episodes from current (real) state s_t

$$\{s_t, a, R_{t+1}^k, ..., S_T^k\}_{k=1}^K \sim \mathcal{M}_v, \pi$$

    - Evaluate actions by mean return (Monte-Carlo evaluation)

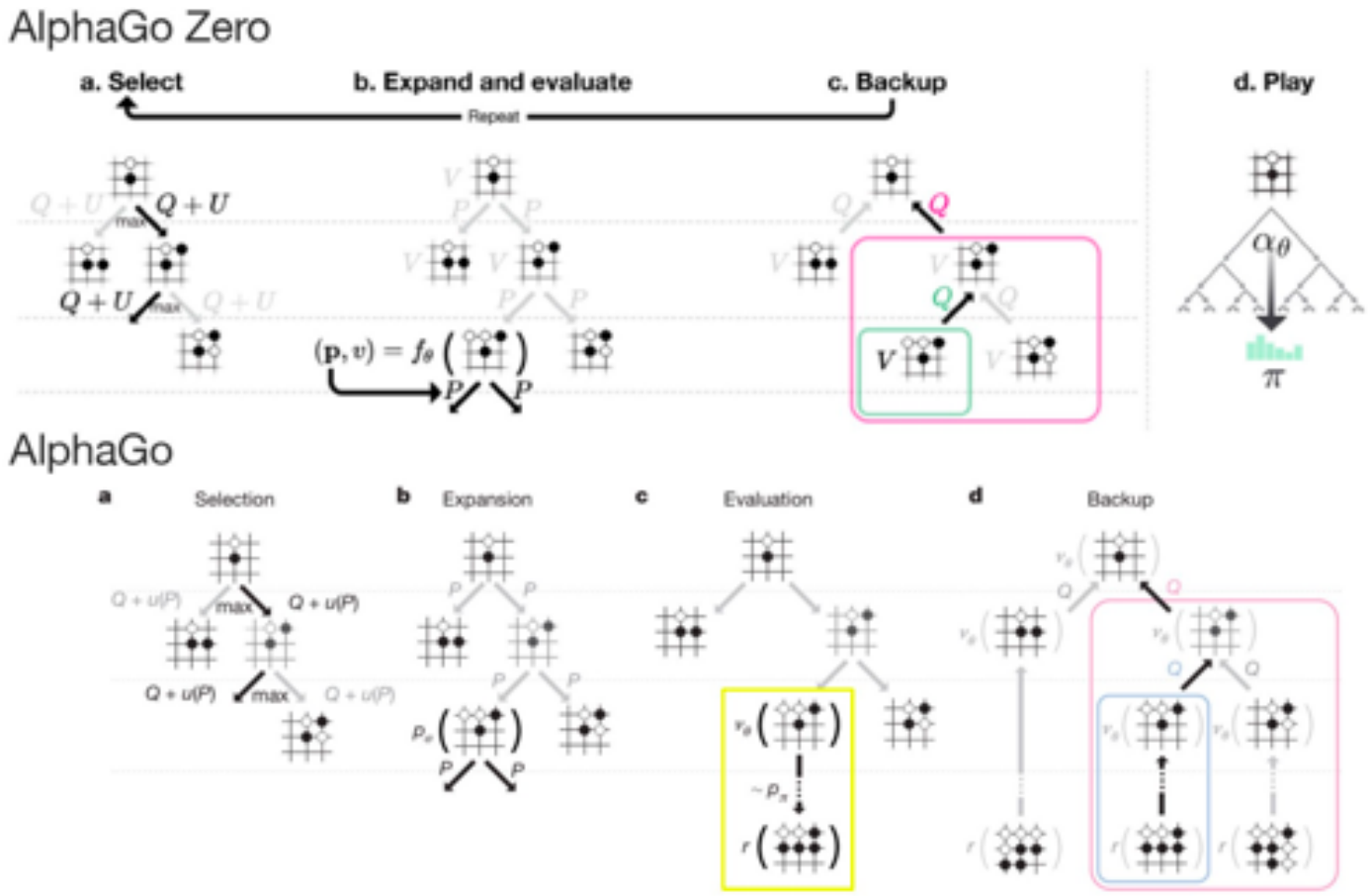
$$Q(s_t, a) = \frac{1}{K} \sum_{k=1}^{K} G_t \xrightarrow{P} q_\pi(s_t, a)$$

  - Select current (real) action with maximum value

$$a_t = \underset{a \in A}{\operatorname{argmax}}\, Q(s_t, a)$$

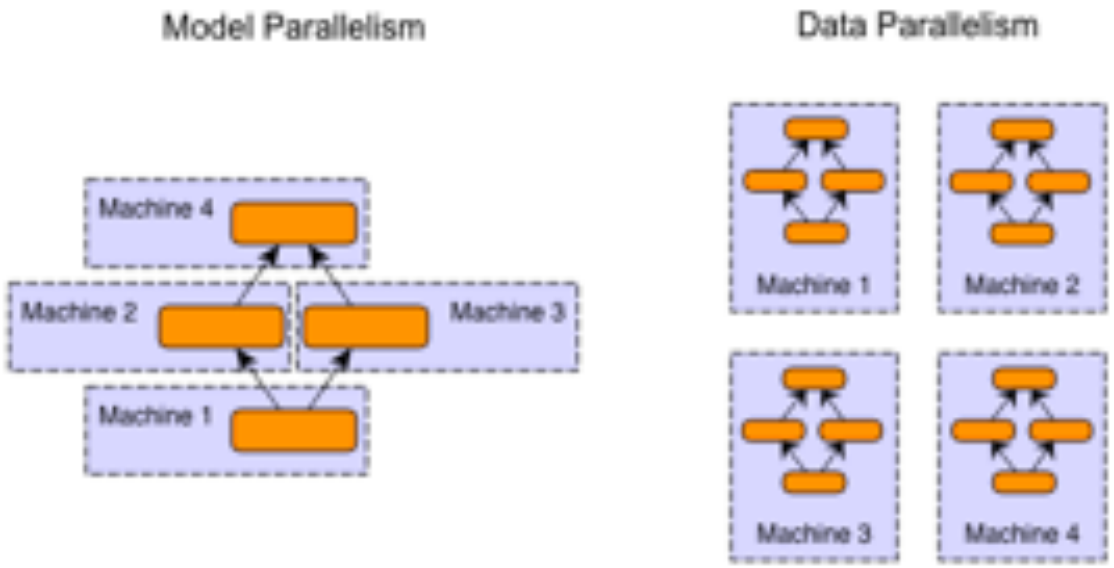
Monte-Carlo Tree Search in AlphaGo Series

- AlphaGo Zero does not use the MC rollout (in yellow)



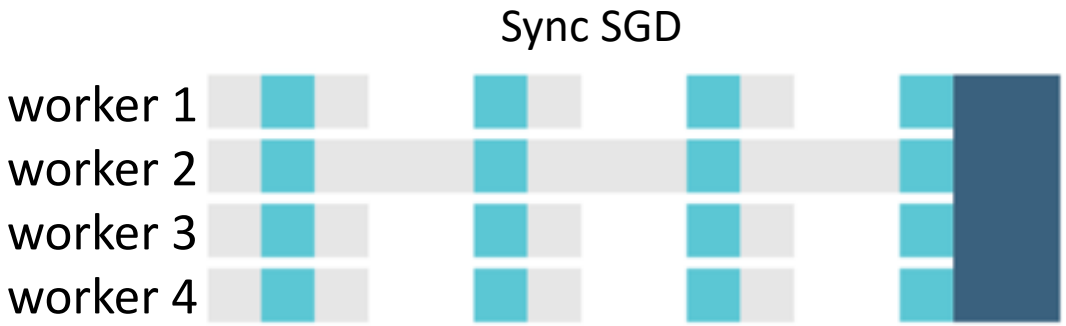
Distributed ML Systems and RL Systems

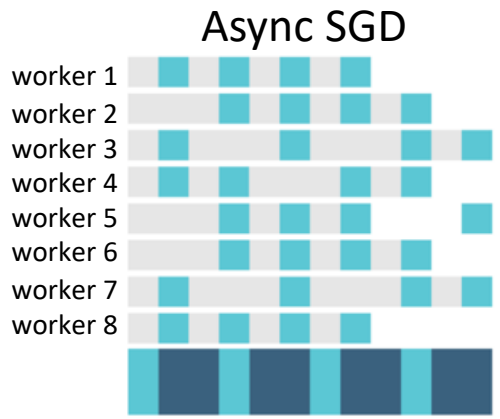
- **Model parallelism**: different machines are responsible for computation in different parts of a single network
- **Data parallelism**: different machine has a complete copy of the model, each machine gets a different portion of the data



Distributed ML Systems and RL Systems

- Synchronous update:
  - The agent are trained in parallel but the updates are collected to update the global network at one time
  - Has to wait until every agent finishes one episode
- Asynchronous update:
  - The global network is periodically updated by individual worker
  - The updates don’t happen simultaneously



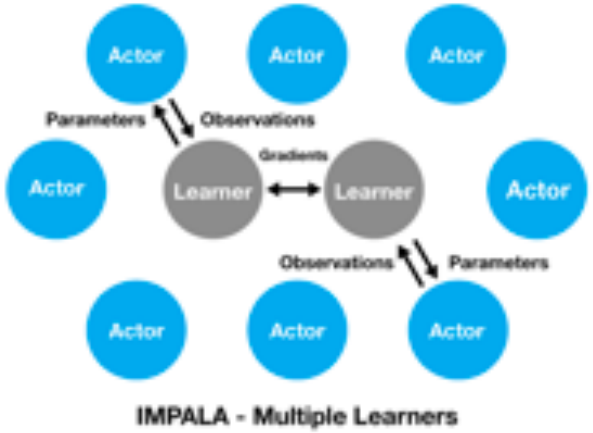


Distributed ML Systems and RL Systems

2013	2015	2016	2017	2018	2018	2018
DQN	GORILA	A3C	A2C	Ape-X	IMPALA	RLlib
Playing Atari with Deep Reinforcement Learning (Mnih 2013)	Massively Parallel Methods for Deep Reinforcement Learning (Nair 2015)	Asynchronous Methods for Deep Reinforcement Learning (Mnih 2016)	https://blog.openai.com/baselines-acktr-a2c/	Distributed Prioritized Experience Replay (Horgan 2018)	IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures	RLlib: Abstractions for Distributed Reinforcement Learning

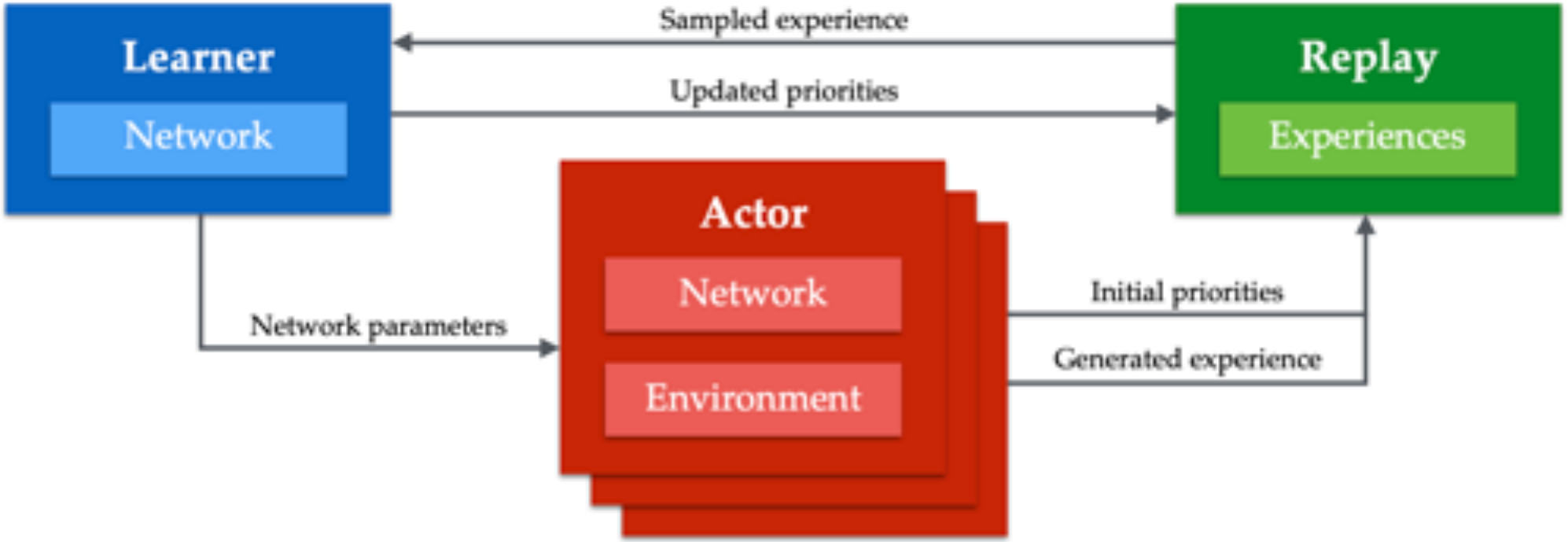
A3C

IMPALA - Single Learner



IMPALA - Multiple Learners

Distributed ML Systems and RL Systems



Horgan et al. Distributed Prioritized Experience Replay. ICLR 2018

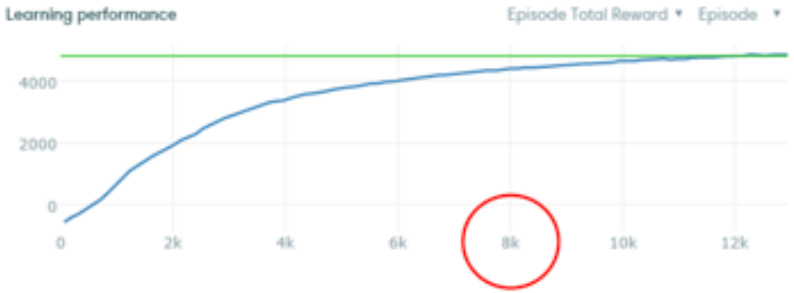
Outline

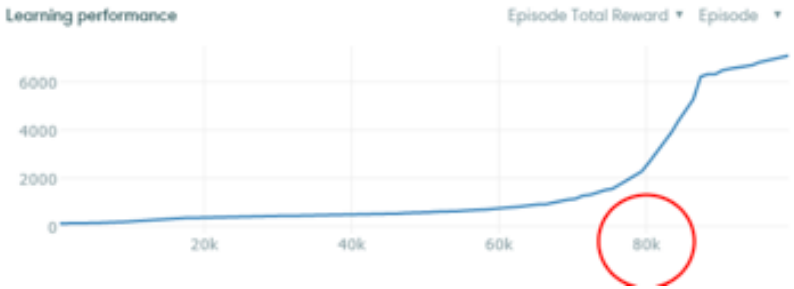
- Basics of RL
- MDP and tabular solution methods
- Value function approximation
- Policy optimization
- Other topics: model-based RL, imitation learning, RL system design
- Open problems

Open Problems in RL

Sample efficiency

Simulation: relatively easy to collect sample





Real-world problems: high cost to collect samples

Open Problems in RL

Generalist RL rather than specialist RL

- Prior knowledge?
- Multi-task learning?

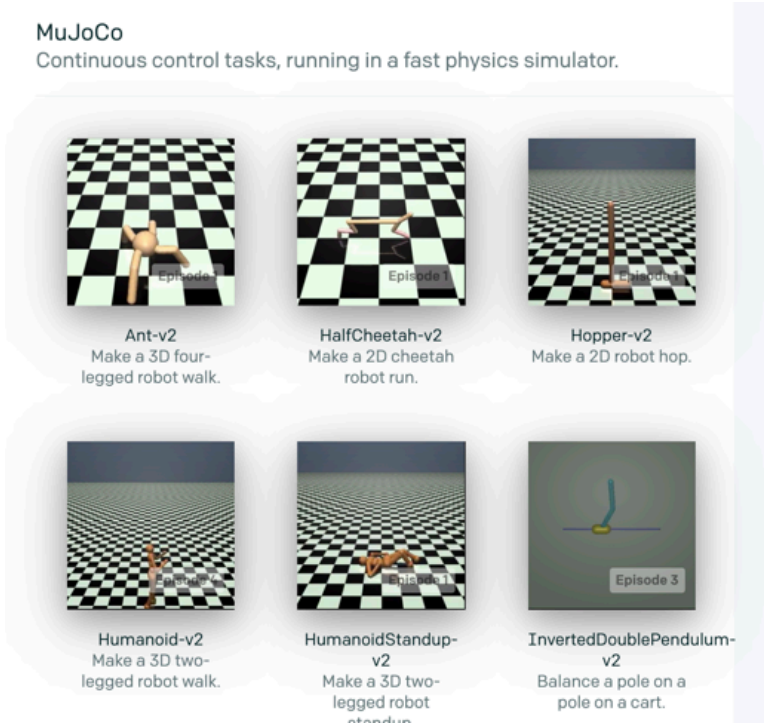
Pinto & Gupta, 2015

Levine et al. 2016

Open Problems in RL

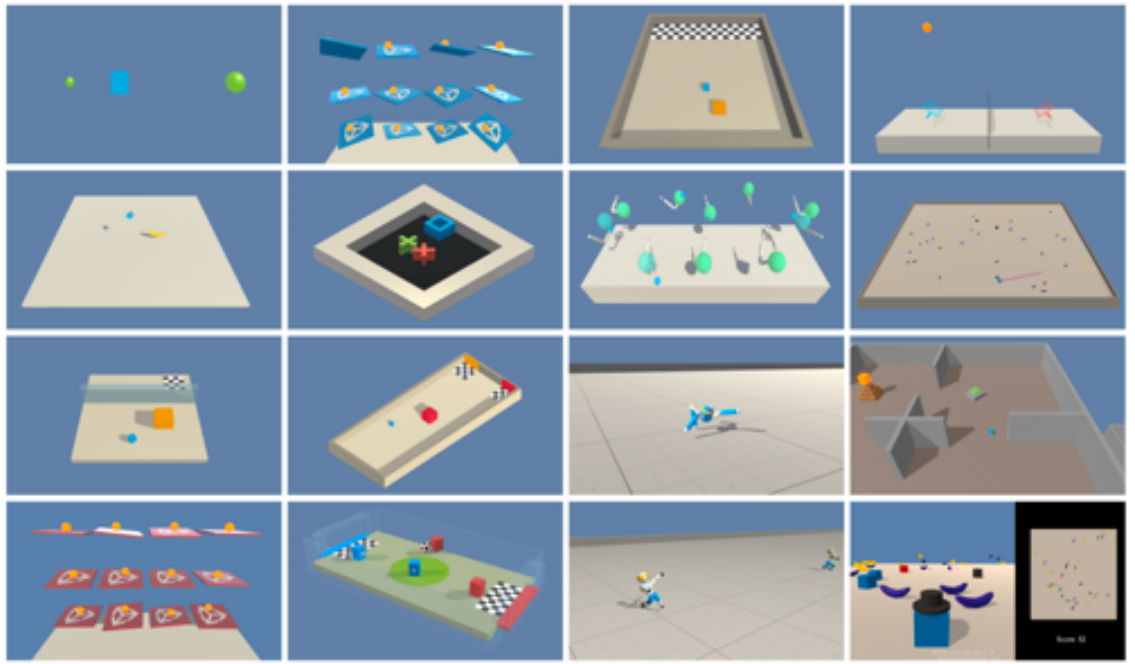
New environments and agent designs

OpenAI Gym



https://gym.openai.com/envs/#classic_control

Define your problem, and develop your own environment!

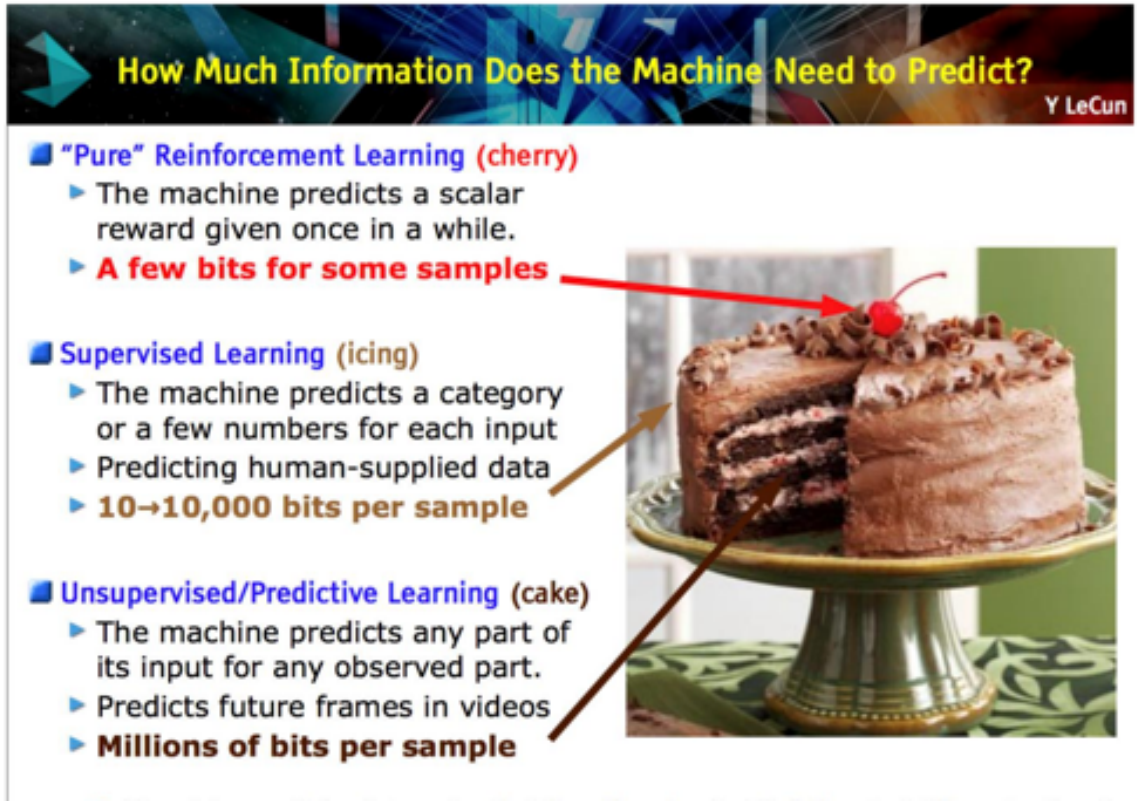


https://github.com/Unity-Technologies/ml-agents

Open Problems in RL

Bridge RL with other ML fields (predictive learning, generative models, etc)

Yann LeCun's cake



Priorities in Real-World RL

- Policy Gradient ↓
- Complex representations ↓
- Computational efficiency ↓
- Control environment ↓
- Learning ↓
- Last policy ↓

Guided Policy Learning ↑

Generalization ↑

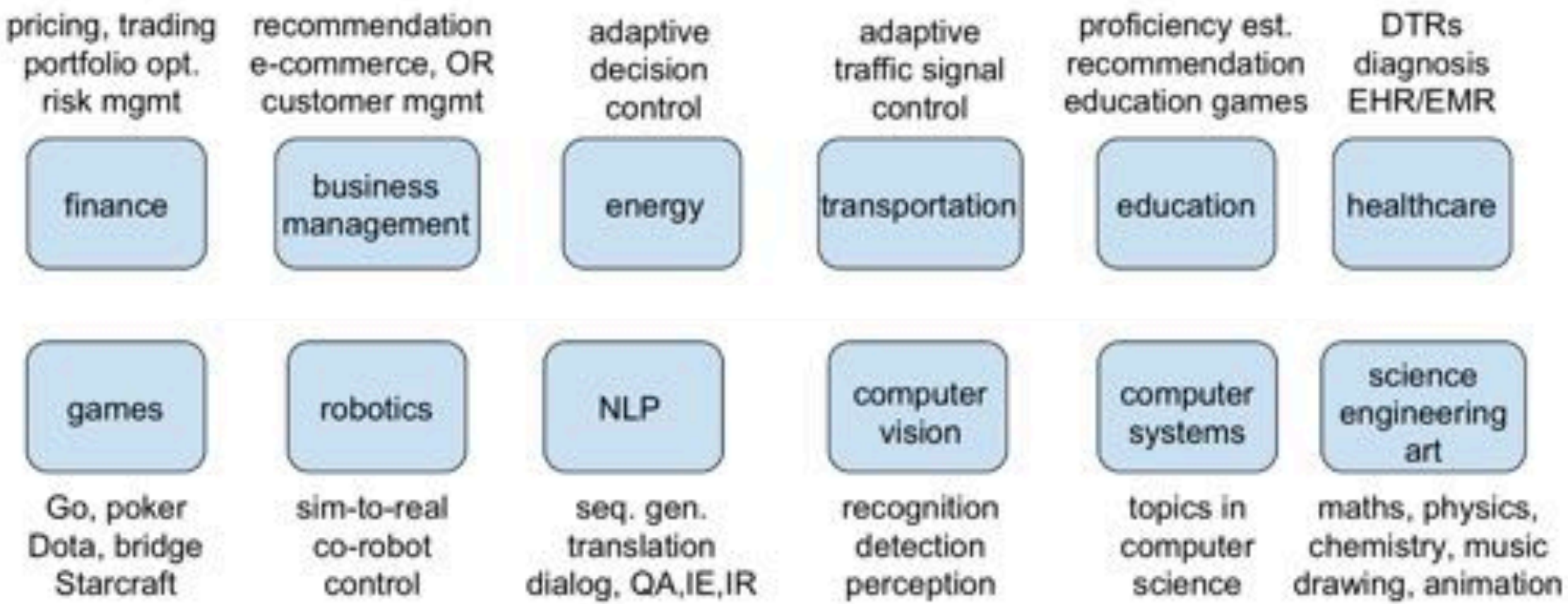
Sample efficiency ↑

Environment controls ↑

Evaluation ↑

Every policy ↑

RL Applications



https://arxiv.org/pdf/1908.06973.pdf

May ReinForce Be With You!

- Online RL introduction course in Chinese

https://github.com/zhoubolei/introRL

	Topic	Resources
Lecture 1	Overview	slide, Youtube(part1, part2), B站(上集, 下集)
Lecture 2	Markov Decision Process	slide, Youtube(part1, part2), B站(上集, 下集)
Lecture 3	Model-free Prediction and Control	slide, Youtube(part1, part2), B站(上集, 下集)
Lecture 4	Value Function Approximation	slide, Youtube(part1, part2), B站(上集, 下集)
Lecture 5	Policy Optimization: Foundation	slide, Youtube(part1, part2), B站(上集, 下集)
Lecture 6	Policy Optimization: State of the art	
Lecture 7	Model-based RL	
Lecture 8	Imitation Learning	
Lecture 9	Distributed computing and RL system design	
Lecture 10	Case Study on AlphaGo Series	

That’s it. Thank you !

- Anonymous online survey: https://forms.gle/qzQtT6F8J3hSzru99