LEC1

Markov Decision Processes(MDP)

is specified by:
- A set of states S • A set of actions A
- Initial state distribution p(s0) • A reward function r(s,a)
- Markov kernel p(s'|s,a) → transition model

Inverted Pendulum(倒立摆)

A set of states $S=[-1,1]\times[-1,1]\times[-8,8]$用正余弦是为了解决角度的周期性

A set of actions $A=[-2,2]$

Initial state distribution: $P(s_0)=[\cos(U(-\pi,\pi)),\sin(U(-\pi,\pi)),U(-1,1)]$

Transition kernel $p(s'|s,a)=\delta(s'-[\cos(\theta_{n+1}),\sin(\theta_{n+1}),\dot\theta_{n+1}])$

$$\dot\theta_{n+1}=\min\left(\max\left(\dot\theta_n+\left(\frac{3g}{2l}\sin(\theta)+\frac{3\tau}{ml^2}\right)\Delta t,-8\right),8\right),\quad \theta_{n+1}=\theta_n+\dot\theta_n\Delta t$$

A reward function: $r(s,a)=-(\theta^2+0.1\dot\theta^2+0.001\tau^2)$

惩罚角度偏离0(竖直向上); 惩罚角速度过大; 惩罚施加过大的力矩

A finite MDP: 只有转换矩阵变了 $P(s'|s,a)=\mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a)$

Tic-Tac-Toe

A set of states $S=\{0,1,2\}^9$代表棋盘上每个格子的内容

A set of actions $A=\{0,1,2,3,4,5,6,7,8\}$当前玩家选择下子的格子编号

Initial state distribution $P(s_0)=\delta(\mathbf{0})$ 所有元素都是 0 的 9 维向量

Transition probability $P(s'|s,a)=\mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a)$ 取决于对手的策略

A reward function $r(s,a)$, e.g. 1 if win, 0 otherwise

Value Functions in MDPs

Fix deterministic policy $\pi:S\to A$ 对于每一个状态，π 都会唯一确定一个动作 $a=\pi(s)$，而不是一个动作的概率分布。

MDP becomes a Markov chain with transition probabilities 动作 a 是确定的，不用考虑动作选择，简化为 $P(S_{t+1}=s'|S_t=s)=P(s'|s,\pi(s))$

Value function $V^\pi(s)=\mathbb{E}[\sum_{t=0}^\infty\gamma^t r(S_t,\pi(S_t))|S_0=s]$累计折扣奖励

Recursive Definition 分解：即时奖励和下一状态折后价值期望

贝尔曼期望方程: $V^\pi(s)=r(s,\pi(s))+\gamma\sum_{s'}P(s'|s,\pi(s))V^\pi(s')$

矩阵形式: $V^\pi=r^\pi+\gamma\mathbf{T}^\pi V^\pi$

Solving for Value Functions 求解 V^π 的两种策略

1. Solving Linear System $V^\pi=(I-\gamma\mathbf{T}^\pi)^{-1}r^\pi$ 适用状态有限，数量少
2. Fixed Point Iteration

Initialize V_0^π ; For t=1:T do: $V_t^\pi=r^\pi+\gamma\mathbf{T}^\pi V_{t-1}^\pi := \mathcal{B}^\pi V_{t-1}^\pi$

当迭代收敛时，即 $V_t^\pi=V_{t-1}^\pi$，找到了方程 $V^\pi=\mathcal{B}^\pi V^\pi$的解，是定点。

$\|V_t^\pi-V^\pi\|_\infty\le\gamma^t\|V_0^\pi-V^\pi\|_\infty$ 第 t 次迭代得到的V_t^π与真实值的差收敛

Picking Actions Given Value Functions

We knew V, could choose: $\arg\max_a\left(r(s,a)+\gamma\sum_{s'}P(s'|s,a)V(s')\right)$贪婪策略

Connecting Value Functions and Policies 价值函数与策略之间的联系

Every policy induces a value function:给定策略，a = π(s) 计算$V^\pi(s)$

Every value function induces a policy 给定 V，找到一个更好的策略

Bellman's Identity 贝尔曼恒等式

$$V^*(s)=\max_a\left(r(s,a)+\gamma\sum_{s'}P(s'|s,a)V^*(s')\right)$$

find value function satisfying:

Policy Iteration: 策略π得到V^π，再用V^π改进得到π',重复直到两者收敛

伪代码: Start with arbitrary policy π

While not converged do

Compute value function V^π

Compute greedy policy π_g w.r.t. V^π

Set $\pi\leftarrow\pi_g$

Convergence to optimal policy in $O(|S|^2|A|/1-\gamma)$ iterations

Value Iteration: 只有价值更新，从V_0反复迭代，直到 V_t 收敛到 V^*

$V_0(s)=\max_a r(s,a),\ V_{t+1}(s)\leftarrow\max_a\left(r(s,a)+\gamma\sum_{s'}P(s'|s,a)V_t(s')\right)$

每次迭代 t 中，用上一步的价值函数估计V_t来计算更优的估计 V_{t+1}

伪代码: Initialize $V_0(s)=\max_a r(s,a)$

Do

Define $Q_t(s,a)=r(s,a)+\gamma\sum_{s'}P(s'|s,a)V_{t-1}(s')$

Find $V_t(s)=\max_a Q_t(s,a)$

While $\|V_t-V_{t-1}\|_\infty=\max_s|V_t(s)-V_{t-1}(s)|\le\epsilon$

Compute greedy policy π_g w.r.t. V_t

Guaranteed to converge to ϵ-optimal policy!

Value vs. Policy Iteration

	Policy	Value
Converges to	exact solution	ϵ-optimal solution
complexity per iteration	$O(\|S\|^3)$	$O(\|S\|^2\|A\|)$

LEC2

The Multi-Armed Bandit Problem

衡量指标: cumulative regret $\rho_T=\max\sum_{t=1}^T\mathbb{E}[r_a]-\sum_{t=1}^T\mathbb{E}[r_{a_t}]$

在 T 个回合中，始终选择最佳臂获得的预期总奖励，与智能体策略实际获得的预期总奖励的差值，目标是设计一个策略，使累积遗憾ρ_T最小化

Uniform Exploration: $\rho_T\ge T\left(\frac{K-1}{K}\right)\min_{a\ne a^*}[r_{a^*}-r_a]$

Explore-First Algorithm:

Idea: approximate expectation empirically:对每个臂的真实 r 进行经验近似

1. Try each arm N_a times: $\mu(a)=\mathbb{E}[r_a]\approx\sum_{t=1}^{N_a}R_t/N_a=\hat\mu(a)$经验平均
2. Play arm with highest average for the remaining rounds

Hoeffding inequality 霍夫丁不等式

Assume $R_t\in[0,1]$. Then, $P\left(|\mu(a)-\hat\mu(a)|\le\sqrt{\frac{2\log(T)}{N_a}}\right)\ge1-\frac{2}{T^4}$

高概率保证，经验估计与真实值误差不超过---->

N_a太小，估计可能不准确，可能导致在剩下的回合中永远拉劣质臂。N_a太大，虽然估计准确，但会浪费大量的回合，从而增加累积遗憾ρ_T

Theorem: Assume $R_t\in[0,1]$and pick $N_a=(T/K)^{2/3}(\log(T))^{1/3}$.

$\mathbb{E}[\rho_T]\le O(KT^{2/3}(\log(T))^{1/3})$说明这样选择$N_a$，累计遗憾有上界

Epsilon-Greedy Algorithm 1-ϵ+ϵ/N

大多数时间(p=1−ϵ)用当前最好的知识，但在小部分时间(p=ϵ)随机探索。

伪代码: For each round t = 1, …, T do

Toss coin with success probability ϵ_t(以 ϵ_t 概率执行探索)

If success then

Choose arm uniformly at random 均匀探索，收集信息

Else(以 $1-\epsilon_t$ 的概率执行利用)

Choose arm with highest average 选当前$\hat\mu(a)$最高的臂

Incremental Updates: $\hat\mu_{N_a+1}(a)=\hat\mu_{N_a}(a)+(R_{N_a}-\hat\mu_{N_a}(a))/N_a$

Theorem: Assume $R_t\in[0,1]$ and $\epsilon_t=t^{-1/3}(K\log(t))^{1/3}$.

$\mathbb{E}[\rho_T]\le O(KT^{2/3}(\log(T))^{1/3})$使用最优$\epsilon_t$时$\mathbb{E}[\rho_T]$的上界与探索优先算法在最优 N_a下的上界具有相同的渐进增长率

Incremental Value Estimation 新估计=旧估计+步长×(目标−旧估计)

Exact estimates if $\sum_{N=1}^\infty\alpha_{N_a}=\infty$ and $\sum_{N=1}^\infty\alpha_{N_a}^2<\infty$ 数据量大使估计更真实

In practice often: α_{N_a}=const. → exponential recency decay

Optimism in the Face of Uncertainty 探索: 赋予不充分尝试的较高估计值

Idea: Use Hoeffding inequality to compute upper confidence bound (UCB)

1. For each round t = 1, … , T do
2. Choose arm that maximizes UCB $ucb(a)=\hat\mu(a)+\sqrt{2\log(T)/N_a}$
3. Play arm and update UCB; Theorem: $E[\rho_T]\le O(\sqrt{KT\log(T)})$

Random Policies in Bandits

- Idea: Don't sample uniformly at random=>Boltzmann distribution $\pi_t(a)=P(A_t=a)=e^{H_t(a)}/\sum_{k=1}^K e^{H_t(k)}$

For each round t = 1, ..., T do

Sample arm $A_t\sim\pi_t$ 根据 Boltzmann 分布得到的概率$\pi_t(a)$，随机采样A_t

Stochastic Gradient Ascent for Policies 随机梯度上升：更新偏好值

Update H_t using gradient ascent: 希望通过梯度上升来更新偏好值 $H_t(a)$，以最大化预期的回报 $E[r_{A_t}]$

$$H_{t+1}(a)=H_t(a)+\alpha\frac{\partial E[r_{A_t}]}{\partial H_t(a)}=H_t(a)+\alpha\mathbb{E}[(r_{A_t}-\bar R_t)(\mathbb{I}_{A_t=a}-\pi_t(a))]$$

Approximate expectation using single sample:没法算 E 的精确值，用单次采样来近似$H_{t+1}(a)=H_t(a)+\alpha(r_t-\bar R_t)(\mathbb{I}_{A_t=a}-\pi_t(a))$

Comparison of Approaches UCB 是最鲁棒和性能最好的算法

Reinforcement Learning Setting

learn transition probabilities -- model-based RL

learn value function -- model-free RL

Exploration-Exploitation in MDPs Similar problem as in bandits:

Always pick the best action(• Get good immediate reward • Can sacrifice long-term return and get stuck in sub-optimal actions 牺牲长期回报

Always pick random action(• Allows to correctly estimate all probabilities and rewards*可能准确估计• May give very poor returns 但是高遗憾)

Epsilon-Greedy Action Selection in MDPs

The R_{max} Algorithm(model-based, off-policy 可以用观测数据更新模型)

- Idea: optimism in the face of uncertainty
- If no estimate for $r(s,a)$ is known → set it to R_{max}奖励不知道就设最大
- If no estimate for $P(s'|s,a)$ is known → set $P(s^*|s,a)=1, r(s^*,a)=R_{max}$, where s^* is a 'fairy tale' state 无转移概率，假设转移到能有R_{max}的状态
- Iterate between solving estimated MDP and applying obtained policy long enough 用当前的乐观估计模型运行规划算法（如价值迭代或策略迭代）得到当前模型下的最优策略，按照该策略在真实情况下运行，用真实观测更新模型中(s, a)对的估计，当被观察多次后，就会从乐观假设中移除
- Greedy optimistic selection leads to implicit exploration!追逐最大奖励的过程中，自然而然地完成了对不确定区域的探索。

How long is 'long enough'

1. How many samples per state-action pair 确保估计准确需要多少样本 通过 Hoeffding inequality 求需要收集多少次观测样本 N，
2. How long it takes to 'traverse' the MDP 需要多长时间探索完所有状态

Lemma: Every T time steps, with high probability, either A near optimal reward is obtained or At least one unknown state-action pair is visited

3. mixing time T^* time until the Markov chain is close to steady state distribution 经过多长时间步后，状态分布会变得接近于稳定状态分布

Theory: With probability 1−δ, the R_{max} Algorithm will reach an ϵ-optimal policy in a number of time steps that is polynomial in $|S|,|A|,T,\frac{1}{\epsilon}$, and $\log(\frac{1}{\delta})$.

Challenges with Model-Based RL

Memory requirement: For every s,a,s', we need to store the probability estimate $\hat P(s'|s,a)\to O(|S|^2|A|)$ for dense MDPs.需要巨大内存

Computation time: need to repeatedly solve the estimated MDPs, e.g. using policy/value iteration.每次更新模型都要重新求解，时间会很长

LEC3

Monte-Carlo Estimates of Value Functions(model-free)

Idea: average over returns instead of estimating probabilities

(Return, G): $G_t=R_t+\gamma R_{t+1}+\gamma^2R_{t+2}+\ldots$; (Value, V)是对这个G的期望

Hoeffding inequality to bound estimation error for finite horizon tasks;

Truncation error 截断误差 in infinite horizon tasks decays exponentially;

Value of single state can be estimated independently 不需知道邻近状态 V

伪代码: For each episode n=1,…,N do:

roll-out policy π: 在环境中用策略π运行一个完整的轨迹

For each state s∈S do:

G←return after reaching s first time: (first visit MC)

append G to returns(s):

V(s)←average(returns(s)):

Monte-Carlo Estimates of State-Action Value Functions 拓展到 Q(s,a)

Q(s,a)是在 s 采取 a 之后，并永久遵循策略π所获得的预期累积折扣回报

探索不足：Requires policy to play every action 需要所有 a 都被尝试到

→ not possible with deterministic policies 在确定π时无法得到其他 Q(s,a')

伪代码: 上述的把 s 换(s,a), G 换 q, V 换 Q

Monte-Carlo Control

Idea: Use random initial state-action pair for exploration 从随机(s,a)开始

Do policy iteration with Monte-Carlo estimate instead of exact value function 在策略迭代循环中，用 MC 估计来代替价值函数的精确解

MC Policy Evaluation(π→Q)遵循π得到每(s,a)的平均回报，得到估计Q^π

Greedy Policy Improvement(Q→π):根据Q^π, 在每个状态 s 下选择使 Q(s,a)最大的动作 a，从而得到一个新的贪婪策略 π'

On-policy method: data can only be used to estimate value function for policy generating the data!

Monte-Carlo Control without Exploring Starts 不使用探索性开始

Exploring starts are unrealistic in many applications

→ use random policies: ϵ-greedy, Boltzmann

→ soft policies: every action retains minimal probability 保证任何状态 s 下的 a 都有非零概率被选中的策略: $\pi(a|s)>0$ for all s,a

Policy improvement theorem 策略改进定理 also holds for ϵ-greedy policies 过程:用软性策略产生数据，并用 MC 方法估计Q^π，用Q^π找一个对Q^π贪婪的软性策略 π'，重复直到收敛到最优的软性策略 π^*_ϵ

Importance Sampling for Off-Policy RL

Reason: On-policy methods are not data efficient,π改进后旧 data 不能 reuse

Idea: use weighted average of returns for policy μ to estimate value functions for different policy π用权重来校正策略μ的G_t，像π产生的回报

$$\rho_t=\frac{\prod_{k=t}^{T-1}\pi(A_k|S_k)P(S_{k+1}|S_k,A_k)}{\prod_{k=t}^{T-1}\mu(A_k|S_k)P(S_{k+1}|S_k,A_k)}=\prod_{k=t}^{T-1}\frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}\qquad V(s)=\frac{\sum_{n=1}^N\rho_{t(n)}G_{t(n)}(s)}{N(s)}$$

$\rho_{t(n)}$: 第 n 个回合中，状态 s 首次访问时的重要性比率

$G_{t(n)}(s)$: 状态 s 首次访问后获得的回报。$N(s)$: 访问状态 s 的总回合数。

Value Estimation using Temporal Differences (model-free, on-policy)

Idea 1: approximate expectation using one sample: $V^\pi(s)\approx r(s,\pi(s))+\gamma V^\pi(s')$

Idea 2: bootstrap using old estimate of $V^\pi(s)$: $V^\pi_{new}(s)=r(s,\pi(s))+\gamma V^\pi_{old}(s')$

TD(0) Learning $V^\pi_{new}(s)=r(s,\pi(s))+\gamma V^\pi_{old}(s')$ 用当前 V 估计真实$V^\pi(s)$ estimate can exhibit high variance 波动性大

mix old and new estimate: $V^\pi_{new}(s)=(1-\alpha_t)V^\pi_{old}(s)+\alpha_t(r(s,\pi(s))+\gamma V^\pi_{old}(s'))$

$V^\pi_{new}(s)=V^\pi_{old}(s)+\alpha_t(r(s,\pi(s))+\gamma V^\pi_{old}(s')-V^\pi_{old}(s))$

temporal difference error = new estimate − old estimate

伪代码: For each episode n=1,…,N do:

initialize S_0:

For each t=0,…,T−1 do:

apply action A_t given by $\pi(S_t)$: 在当前状态下执行动作

observe S_{t+1},R_t: 观察返回奖励和下一状态

$V(S_t)\leftarrow V(S_t)+\alpha_t(R_t+\gamma V(S_{t+1})-V(S_t))$用观测更新$S_t$价值估计

收敛到真实 V^π条件: $\sum_t\alpha_t=\infty$且$\sum_t\alpha_t^2<\infty$; 所有 s 被访问无穷多次

SARSA – On-Policy TD Control - **(s,a,r(s,a),s',a')**

$Q_{new}(s,a)=(1-\alpha_t)Q_{old}(s,a)+\alpha_t(r(s,a)+\gamma Q_{old}(s',a'))$

伪代码: For each episode n=1,…,N do:

initialize S_0:

determine policy π based on Q:

For each t=0,…,T−1 do:

apply action A_t for S_t based on policy π:

observe S_{t+1},R_t: 观察返回奖励和下一状态

choose action A_{t+1} for S_{t+1} based on policy π:

$Q(S_t,A_t)\leftarrow Q(S_t,A_t)+\alpha_t(R_t+\gamma Q(S_{t+1},A_{t+1})-Q(S_t,A_t))$

收敛到最优 Q^* 条件: $\sum_t\alpha_t=\infty$且$\sum_t\alpha_t^2<\infty$; all (s,a)被访问无穷多次

Off-Policy State-Action Value Estimation

SARSA estimate:$Q(S_{t+1},A_{t+1})$当前策略实际选择并执行的动作

Off-policy estimate:$Q^\pi(S_{t+1},\pi(S_{t+1}))$采取目标策略认为最好的那个动作

Actions do not need to be picked according to π!生成数据的动作 A_t（由行为策略 b 选择）不用与想要评估的目标策略π所建议的动作一致

Q-Learning 直接学习最优价值函数

Idea: use greedy policy in off-policy evaluation: a'是理论上最好的动作

$Q^*(S_t,A_t)\leftarrow Q^*(S_t,A_t)+\alpha_t\left(R_t+\gamma\max_{a'}Q^*(S_{t+1},a')-Q^*(S_t,A_t)\right)$

伪代码: For each episode n=1,…,N do:

initialize S0:

For each t=0,…,T−1 do:

apply action At for St based on Q^* :

observe S_{t+1},R_t:

$Q^*(S_t,A_t)\leftarrow Q^*(S_t,A_t)+\alpha_t(R_t+\gamma\max_{a'}Q^*(S_{t+1},a')-Q^*(S_t,A_t))$

多项式界限:log|S|(←number of states): 意味 Q-Learning 在状态空间较大的问题上具有较好的可扩展性能。log|A|(←number of actions);

1/ϵ(←ϵ-optimal): 与所需的精度成多项式关系。

log(1/δ)(←probability of guarantee): 与所需保证的概率成对数关系。

性质 Memory requirement: need to store $Q^*(s,a)\forall s,a\to O(|S||A|)$

Computation time: O(|A|) per iteration $a'\in\arg\max_{a'}Q^*(s,a')$耗在找最优 a'

Challenges of Tabular RL

MDP and RL polynomial in |S| and|A| in general 理论复杂度

key challenge: scaling up (Structured domains 象棋) (Continuous domains)

→ Use function approximation for value functions 用参数化函数估计

TD-Learning as Stochastic Gradient Descent

Idea: TD-Learning looks like gradient descent update

$V^\pi_{new}(s)=V^\pi_{old}(s)+\alpha_t(r(s,\pi(s))+\gamma V^\pi_{old}(s')-V^\pi_{old}(s))$

'parameterize' value function: $V^\pi(s;\theta)$转移到参数θ

$V^\pi(s;\theta)\leftarrow V^\pi(s;\theta)+\alpha_t(r(s,\pi(s))+\gamma V^\pi(s';\theta^{old})-V^\pi(s;\theta))$

→ stochastic semi-gradient descent 随机半梯度下降不求θ^{old}这部分梯度

→ identical to TD learning if $\theta^{old}=\theta$ 半梯度等同标准更新

缺点: Sample-inefficiency of model-free RL

目标函数不正确且不稳定：大方差来自单样本期望近似

Q-learning via Stochastic Gradient Descent

loss function for SGD:

$$l(\theta;s,a)=\frac{1}{2}\left(r(s,a)+\gamma E_{s'}[\max_{a'}Q^*(s',a';\theta^{old})]-Q^*(s,a;\theta)\right)^2$$

update: $Q^*(s,a;\theta)\leftarrow Q^*(s,a;\theta)+\alpha_t\left(r(s,a)+\gamma\max_{a'}Q^*(s',a';\theta^{old})-Q^*(s,a;\theta)\right)$

Model-free RL with Function Approximation

Idea: SGD does allows using other parameterizations of value functions

Q-learning:$\theta\leftarrow\theta+\alpha_t\delta_B(s,a,s')\nabla_\theta Q^*(s,a;\theta)$收敛很慢

Heuristics 启发式 for Deep Q-Learning

stabilize training by keeping θ^{old} over multiple SGD iterations 固定参数

maintain a data set of observed transitions

do SGD updates on mini batches of data → experience replay

LEC4

Drone:

State: Drone's position 位置; orientation 姿态; velocity; target command

Actions: rotor command 旋翼指令 **Initial state distribution** p(s0)

Markov transition model p(s'|s,a)

Reward function: Task reward (tracking command): $-|v_t-v_{target}|^2$,

Regularization reward (punish undesired jerky motion): $-|acc_t|^2$,

Survival reward (terminate if fail): -c 在任务失败给的一个很大负奖励

Total reward: $r(s,a)=w_{task}|v_t-v_{target}|^2-w_{reg}|acc_t|^2$

Quadrupedal robot:

State: Robot's joints state, base orientation, velocity, target command

Actions: joints command **Initial state distribution** p(s0)

Markov transition model p(s'|s,a)

Reward function: Task reward (tracking command): $-|v_t-v_{target}|^2$

Regularization reward (punish undesired joints jerky motion): $-|\delta_t|^2$,

Survival reward (terminate if fail): -c

Total reward: $r(s,a)=w_{task}|v_t-v_{target}|^2-w_{reg}|\delta_t|^2$

Good controller = Maximize objective function

LEC5

MDP in GO

State: stones state **Actions**: next stone position

RL in: Go vs. Robot Control

- Observation state: Discrete vs. Continuous
- Action: Discrete vs. Continuous
- Transition model: Table/Matrix vs. Approximated function
- policy: Over finite state vs. Prob. density function Over continuous space

Monte Carlo Tree Search (MCTS)

- Heuristic search algorithm for decision-making.启发式搜索算法
- Balances exploration and exploitation using simulations.平衡探索和利用
- No domain-specific knowledge required.无需领域知识，大量随机模拟
- Applications: Go, Chess, planning, etc.

1. Selection: Traverse tree using Upper Confidence Bound applied to Trees (UCT) to select promising node.在每个节点选择最有希望的子节点
2. Expansion: Add a new child node.有未探索的动作，添加新的子节点
3. Simulation: Run rollout to get outcome.评估新子节点的价值
4. Backpropagation: Update values up the tree.沿路径更新访问次数，总 V

Why Balances Exploration and Exploitation?

$$UCT(s,a)=\underbrace{Q(s,a)}_{\text{利用项 (Exploitation)}}+\underbrace{c\cdot\sqrt{\frac{\ln N(s)}{N(s,a)}}}_{\text{探索项 (Exploration)}}$$

$Q(s,a)$: Estimated value (average reward) of taking action a from state s, c: 探索常数通常$\sqrt2$

$N(s)$: Total number of visits to state s, $N(s,a)$:从状态 s 采取动作 a 的次数

Advantages:

- Scalable and domain-agnostic. 不用存储整个状态空间而是构建搜索树
- No value function needed.无需先验评估，用大量随机模拟的真实结果
- Strong performance with enough simulations.仿真次数多时性能强大

Limitations:

- High computational cost.计算密集型
- Struggles with sparse rewards and rollout policy design.奖励稀疏，波动

MCTS in Practice

- Used in: AlphaGo, robotics planning.
- Variants: PUCT (AlphaZero), 嵌套 Nested MCTS, MCTS + RL.
- Rollout policy and tree depth are critical for success.模拟策略和树深度

It cannot be Applied to Robot Control Problem:

Continuous space is too costly to build such tree structure.

Model-based RL

循环:generate samples→fit a model to estimate return→improve the policy

Model: estimate $p(s'|\mathbf{s},\mathbf{a})$ Supervised Learning $\min_\phi\sum_i\|f_\phi(\mathbf{s}_i,\mathbf{a}_i)-\mathbf{s}_i'\|^2$

目标: min 预测的下一状态与实际下一状态之间的误差平方,优化$\pi_\theta(\mathbf{a}|\mathbf{s})$

- Sample efficient (Leverage models)优点：需要更少的真实环境数据
- More complex to implement(Learn models, then plan based on model)
- Sensitive to model errors(Less robust)劣势：复杂，对模型错误敏感

Model-free RL

Replay Buffer<=Environment<=>Model-free Agent<=Replay Buffer

Actor 根据当前的策略π向环境发出动作，环境返回 s'和 r，这些经验被记录并送入回放缓冲区，智能体从回放缓冲区中随机抽取小批量数据，价值函数和执行器利用这些数据进行学习更新，用于指导下一轮的行为

- Simple to implement • Robust to model errors
- Lots of samples are required(Without models)

Model-based RL vs. Model-free RL

Model-based RL: Learns Mode; Uses Planning(MPC); more sample efficient

Model-free RL: Simpler setup; Less sensitive(robust)

Robot Reward Design

Run to the target position $r_{pos\_track}=\|p-p_{target}\|^2$ $r_{vel\_track}=\|v-v_{target}\|^2$

Not work in real robots:• Unsmooth motion, hard to track by hardware

- Too large contact force, easy to damage • Not user-friendly motion

Punish Undesired Motion: (Shake; Base tilts; Too fast joints motion; Large contact force)

设计奖励函数: • Usually continuous function• Friendly for approx.

Some Common Reward Function

- Quadratic 二次 function $R(s,a)=-(x-x^*)^2$
- Exponential function $R(s,a)=\exp(-\alpha(x-x^*)^2)$

Discrete Reward 特定的、不连续的事件，通常是一个固定的正值或负值

Termination Reward-正向奖励(Successfully finish the task; Usually a discrete positive reward)

Truncated Reward-负向奖励(Reset when failure happens; Fail to finish within time limits; Largely deviate from ideal states; discrete negative)

Why Truncated Reward? 内容很多，但感觉没用

•Avoid learning from outlier states 避免从异常状态中学习(• Data collected may not represent meaningful behavior• Strong rewards (positive or negative) in such outliers, it could bias the policy toward unrealistic behaviors)

•Prevent instability in value function learning(• Very large or extreme rewards can cause exploding gradients or unstable updates.• Truncating rewards smooths learning by keeping values within a bounded range.)

•Encourage robustness and Generalization 鲁棒泛化(• By ignoring extreme states, the agent learns policies that are effective in the most relevant parts of the state space.• This avoids overfitting to rare, unlikely transitions.)

Why Sim-to-Real mismatch?(• Actuators performance mismatch to ideal models• Sensors noises• Physical engines inaccuracy in simulation • Robot kinematic model inaccuracy• Unexpected deploying scenarios(e.g. perturbation force))

Improve Sim-to-Real Transfer Success 提高成功率• Reduce sim-to-real gap(Improve modelling in simulation: Better calibration, Data-driven method)• Improve robustness of control policy during training(Survive against sim-to-real mismatches)

Domain Randomization 域随机化(提高成功率的一种方法)

Technique that randomly inject noises or perturbation to the system during learning(效果：Ideally robot can meet as much as possible potential scenarios; Emerge generalized skills adapted to environment changes)

Enhance Controller with DR(Sensor noises; Sensor delay / latency; Wind perturbation; Dynamic model mismatch; Actuator performance mismatch)

DR is not the larger the better(why?Too conservative behavior if too large; Fail to learn the behavior)DR settings need robot experience & knowledge.

LEC6

What's the Trend in RL for Robots?(• More modality: from blind to perspective • More advanced network architecture: from simple MLP network to memory to attention• More training complexity: from simple environments to more diverse environments• More skills: from single task to multiple tasks performance

LEC7

Test: Reward Shaping; Constrained MDPs (strict)

Training: Constrained Predictive Control; Control Barrier functions (strict)

Safety via Reward Shaping

Idea: define reward function that assigns large negative values to unsafe behavior: $r(s,a)=r_{\text{task}}(s,a)+r_{safety}(s,a)$

Strength: Standard RL algorithms applicable; Risk-sensitive RL variants(变体).Weakness: No strong guarantees; Complicated reward design.

Risk-sensitive RL

Safety often concerns low-probability events, average expectations can weaken the impact of unsafe returns. Risk-sensitivity desired.

$V^\pi(s)=\mathbb{R}_\beta[\Sigma_{k=0}^\infty\gamma^k r(s_k,\pi(s_k))]$

Entropic Risk $\mathbb{R}_\beta[X]=\log(E[\exp(-\beta X)])/\beta$

Constrained MDPs(CMDP)

Idea: explicitly represent safety conditions as constraints in RL

A set of states $\mathbb{S}$; A set of actions $\mathbb{A}$; Initial state distribution $p(s_0)$;

Markov kernel $p(s'|s,a)$; A reward function $r_{\text{task}}(s,a)$ for the task

$r_{\text{task}}(s,a)$越大越好;A reward function $r_{safety}(s,a)$ for constraints eg binary

Policy optimization problem(on-policy):

$\pi(s)=\arg\max_a\left(r_{task}(s,a)+\gamma E_{s'}[V^\pi_{task}(s')]\right)$ 全局: $\arg\max_\pi E[V^\pi_{task}(s)]$

such that $r_{safety}(s,a)+\gamma E_{s'}[V^\pi_{safety}(s')]\ge c$ 全局: $E[V^\pi_{safety}(s)]\ge c$

constraint is defined via a value function:

$$V^\pi_{safety}(s)=r_{safety}(s,a)+\gamma E_{s'}[V^\pi_{safety}(s')]$$

Temporal difference learning is applicable:

$$V_{safety}(s)\leftarrow V_{safety}(s)+\alpha\left(r_{safety}+\gamma V_{safety}(s')-V_{safety}(s)\right)$$

方法 1.Constrained Policy Optimization (CPO) (policy iteration):

$$\pi_{k+1}=\arg\max_\pi E[A^\pi_{task}(s,a)]$$
$$s.t.\ E[V^\pi_{safety}(s_0)]-\mathrm{E}[A^\pi_{safety}(s,a)]/(1-\gamma)\ge c$$

$\bar D_{KL}(\pi\|\pi_k)\le\delta$ (新策略和旧策略的 KL 距离要小,不能变化太快)

Where,

s_0: initial state; Advantage functions: $A^\pi(s,a)=Q^\pi(s,a)-V^\pi(s)$ (动作危险为负); $1/(1-\gamma)$ is the tightening off-policy;

This guarantees:(1) Policy update is stable;(2) Will not suddenly perform dangerous actions;(3) The on-policy nature remains good.[trust region]

方法 2.Primal-dual method/Lagrangian relaxation:

$$\min_{\lambda\ge0}d(\lambda)$$
$$d(\lambda)=\max_\pi L(\pi,\lambda)=\max_\pi E[V_{task}(s)]+\lambda\left(E[V_{safety}(s)]-c\right)$$

For each round $t=1,\ldots,T$ do

simulate a trajectory using current policy

estimate primal gradient $\nabla_\pi L(\pi,\lambda)$

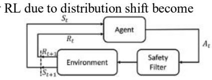
estimate dual gradient $\nabla d(\lambda)=E[V_{safety}(s)]-c$

gradient update: Gradient descent for λ, but gradient ascent for π

update value function estimates

CPO and Lagrangian methods rely on accurate constraint value functions ($E[V_{safety}(s)]-c$): If true constraint value is below the threshold but its estimate is above, Lagrange multiplier update is in wrong direction. Larger estimation errors in off-policy RL due to distribution shift become problematic.

不能用在 safe off-policy RL 上



Control Barrier Functions(CBF)

h(x): >0, inside constraint set; 0 at constraint set boundary; <0 outside constraint set. $\pi_{safe}(s) = \min_a \|a - \pi(s)\|^2$

Condition for positive value (safety filter):

$$s.t.\ E[h(s^{'})] \geq \alpha h(s), \alpha \in [0,1], s^{'} \sim p(\cdot \mid s, a)$$

Allows handling new test-time constraints: + CBF implicitly represents a safe back-up strategy; - Can cause distribution shift.对 on-policy 的挑战

Integration into training by lumping safety filter into environment: + RL agent can adapt to it; - Model knowledge required with little data

May not feasible. =>learn from expert data, RL policy roll-outs

Constrained predictive control(CPC)

Idea: optimize over sequence of actions.

$$\min_{a_0,\dots,a_K} E\left[\sum_{k=0}^{K} \gamma^k r(s_k, a_k) + \gamma^{k+1} V(s_{k+1})\right] \quad s.t.\ s_k \in S_{safe}, a_k \in A_{safe}$$

Only apply first action, optimize again; Model predictive control(MPC)

Challenge: Computational complexity of online optimization.

Online Optimization for MPC:

• Constrained Cross-Entropy Method (CEM): Randomly sample multiple action sequences (GPU parallelization!); Simulate their future trajectories (rollout); Select the top n% samples with the best performance (elite set); Update the distribution and repeat sampling.

Sequential Quadratic Programming (SQP 顺序二次规划): Linearize transitions and constraints; Approximately equivalent to a Quadratic Program (QP); Repeatedly iterate to approach the optimal solution.

Non-convex collocation / shooting methods: Taking system dynamics as the constraint of optimization variables. Suitable for deterministic systems(确定性系统)

LEC8

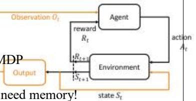
Partially Observable Markov Decision Process (POMDP)

A set of states S (状态集); A set of actions A (动作集)

A set of observations O (观察集); Initial state distribution $p(s_0)$

Markov kernel $p(s'|s,a)$; Observation probabilities $p(o|s,a)$ (观察概率)

A reward function $r_{task}(s,a)$ (奖励函数)



The challenge of solving POMDPs

Problem: A single observation does not provide full information of the state of a POMDP

仅依赖当前观察$a_k = \pi(o_k)$，可能无法区分

由于状态不同但观察相同的情况→Agents need memory!

Planning with Believe States

Idea: Iteratively estimate states from observations

→ Bayesian updates based on $p(s'|s,a)$ and $p(o|s,a)$

→ exact updates for special cases, e.g. finite state-action spaces

Believe state estimation ≈ Kalman filtering

$b_k = p(s_k | o_0, a_0, \dots, a_{k-1}, o_k)$:

b_k 是概率分布，表示在给定迄今为止所有观察和动作历史的条件下，智能体在 k 时刻处于真实状态s_k的概率，即使真实状态 s 是离散的，信念状态b_k也是一个连续变量的向量(概率分布)

→ Markovian, but continuous believe state

$b_k = P(s_k | h_k)$, $h_k = \{o_0, a_0, o_1, a_1, \dots, a_{k-1}, o_k\}$ → 基于旧b_k和a_k预测先验b_{k+1}

$b_{k+1}(s') = P(s' | h_k, a_k) = \sum_{s \in S} P(s' | s, a_k) \cdot P(s | h_k) = \sum_{s \in S} P(s' | s, a_k) \cdot b_k(s)$

→ 接收o_{k+1}后更新 得到后验信念状态

$$b_{k+1}(s') = P(s' | h_k, a_k, o_{k+1}) = \frac{P(o_{k+1} | s', a_k) \cdot b_{k+1}(s')}{P(o_{k+1} | h_k, a_k)}$$

Learning value functions:

策略的目标是最大化在信念状态b_k下的期望累积折扣奖励

$$E\left[\sum_{k=0}^{\infty} \gamma^k \sum_{s \in S} b_k(s) r(s, a_k)\right]$$

$\sum_{s \in S} b_k(s) r(s, a_k)$: 给定$b_k$时，执行$a_k$的即时期望 r

$a_k = \pi(b_k)$ $b_k = g(b_{k-1}, a_{k-1}, o_k)$ 精确求解→ requires model of transition and observation probabilities→ractable only for small problems in practice

Approximate Planning with Believe States 精确太难所以近似

MLE approximation:最大似然，用信念状态中最有可能的真实状态代替

$E\left[\sum_{k=0}^{\infty} \gamma^k r(\arg\max b_k(s), a_k)\right]$, $a_k = \pi(\arg\max b_k(s))$, $b_k = g(b_{k-1}, a_{k-1}, o_k)$

从连续信念状态空间近似回离散的真实状态空间，就可以用 MDP 求解

→ standard planning problem with continuous states:理论上仍然是连续的

→ Much more: variants of value iteration, point-based methods, policy search, Monte-Carlo methods 针对 POMDP 的各种方法

POMDPs are powerful, but solving them is very complicated!

RL for POMDPs

Model-based approaches(• Learn models for transition and output probabilities→ difficult) 学习 POMDP 的转移概率和观察，非常难

Model-free approaches:(• Finite window of past observations and actions 依赖过去 L 个时间步→ counter examples can be found 更早的历史不同)

Recurrent Neural Networks

h_t 依赖于前一时刻的隐藏状态h_{t-1}动作A_{t-1}和当前时刻的输入O_t

Neural network becomes cyclic!→RNNs have memory!

technical difficulties:(back-propagation; Exploding/vanishing gradients)

Common realizations of RNNs for RL: LSTM, GRU

Model-based: Deep Variational Reinforcement Learning, DVRL

Idea: Use RNNs in a model-based approach to determine believe

Model-free: RNNs as parameterization

Idea: Use RNNs as encoder in value function and policy parameterization

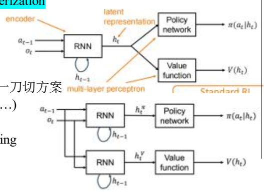
用h_t充当信念状态的近似

Design considerations: 没有一刀切方案

• RNN variant (LSTM, GRU, …)

→ no significant difference

• Sequence length during training



(short: ~5, long ~100+)

• Architecture → separate encoders seem to work better

• Model-free RL algorithm → indicators for benefits of off-policy

• Encoder inputs → reward signal can be beneficial 奖励也作为输入

• End-to-end? → additional information is accessible for training

Performance Comparison(在没有速度信息情况下，从历史推断速度)

No method consistently outperforms all others 不同算法不同任务表现不一

Model-based method is comp. Expensive 基于模型的方法计算成本高

Difference between methods insignificant"(方法间的差异微不足道)

Important Special Cases of POMDPs (Specialized algorithms exist)

Meta RL problems→learning from different tasks without knowledge about them 从观察到的奖励和环境反馈中推断出它所处的任务

Robust RL problems→unobservable state affecting transitions 抵抗扰动

Temporal credit assignment→delayed rewards

Visuomotor Policy Training 视觉运动策略训练（输入是图片）

convolutional neural network layer for low dim. Representation 用 CNN

Challenge: Training difficult/data-hungry

Design considerations:

• RNN necessary? → finite/no history can work 有时可以省略 RNN

• End-to-end? → additional information is accessible for training 额外信息

• CNN pre-training? → unclear 不一定要与训练 CNN

• Many additional tricks for training CNNs

• Additional supervision can be used in training(auxiliary local policies 子任务的策略; expert demonstrations 用人类或高表现策略的演示数据)

LEC9 Increase Data Efficiency

Inverse Reinforcement Learning(IRL)

Assumption: demonstrations(示范) are from optimal policy:知策略推奖励

$E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t) | \pi^*\right] \geq E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t) | \pi\right], \forall \pi$

在推断出的奖励函数下，专家策略获得的期望累积回报，必须大于等于任何其他策略获得的期望累积回报

Issues: Reward ambiguity: trivial solutions(琐碎解): $r(s_t) = 0$; Relies on expert optimality; Computational complexity (calculate all π).

Featurization of Reward Functions 限制奖励函数的搜索空间

Idea: Linearly parameterize the rewards $r(s) = w^T \phi(s)$.权重未知

$E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t) | \pi\right] = E\left[\sum_{t=0}^{\infty} \gamma^t w^T \phi(s) | \pi\right] = w^T \mu(\pi)$

Find w^* such that $w^T \mu(\pi^*) \geq w^T \mu(\pi)$ for all π

Max-Margin Formulation

$$\min_{w, \varepsilon} \|w\|^2 + c\varepsilon$$

$$s.t.\ w^T \mu(\pi^*) \geq w^T \mu(\pi) + m(\pi, \pi^*) - \varepsilon, \forall \pi$$

$m(\pi, \pi^*)$: the gap between the strategy π and the expert strategy π^*;

ε: slack variable.

伪代码: Initialize weights w

For $t = 1, \dots, T$

obtain $\pi(s)$ by solving forward RL(often approximate solutions in practice)

problem with $r(s) = w^T \phi(s)$

do sub-gradient update step for w(non-differentiable), ε

Max-Entropy Feature Expectation Matching

Insight: Similarity of policies can be measured independently from the weights w.(只要策略的特征期望$\mu(\pi)$和专家策略$\mu(\pi^*)$足够接近，那么在任何线性奖励权重 w 下，π的价值和专家的价值也会很接近。)

Idea: Parametrize problem in terms of path probabilities $P(\zeta)$→ Maximize entropy of distributions over paths. Trajectory $\zeta = (s_0, a_0, s_1, a_1, \dots)$

$\max_P -\sum_\zeta P(\zeta) \log P(\zeta)$ resolves ambiguity

$s.t.\ \sum_\zeta P(\zeta) \mu(\zeta) = \mu(\pi^*)$ sum of features along path

Solution via log-likelihood maximization of maximum entropy distribution

$P(\zeta) = \exp(w^T \mu(\zeta)) / z(\theta) = \exp(w^T \mu(\zeta)) / \sum_\zeta \exp(w^T \mu(\zeta))$

Imitation Learning (just want to learn a policy)

Use fixed data set $\mathcal{D} = \{(s_i, a_i)_{i=1,\dots,N}\}$ to do empirical loss minimization

$\hat{\pi} = \arg\min_\pi \sum_{i=1}^N l(\pi, s_i, a_i)$, i.e. $l(\pi, s_i, a_i) = -\log \pi(a|s)$ or $\|\pi(s) - a\|^2$

Behavior Cloning

Data is generally not uniformly distributed across the state space!

Expert policy π^*induces a state distribution $p(\pi^*)$

$\arg\min \sum_{i=1}^N l(\pi, s_i, a_i) \approx \arg\min E_{s \sim p(\pi^*)}(l(\pi, s, \pi^*(s)))$ 近似期望损失

Distribution Shift:进入专家未见过的新状态，策略表现差，误差累积

Training distribution $s \sim p_{\pi^*}$; Testing distribution $s \sim p_{\hat{\pi}}$

Distribution shift occurs between training and testing!

compounding errors drop returns by at most ϵT^2,

ϵ: training loss, T: number of time steps.

On-Policy Imitation Learning

Idea: Generate data iteratively to converge to test distribution.

=>Employ mixing techniques.伪代码如下：

Initialize $\hat{\pi}(s), \mathcal{D}$

For $i = 1, \dots, N$

$\pi_i = \beta \pi^* + (1 - \beta_i)\hat{\pi}$ (Data Aggregation (DAgger) method)

rollout policy πi to generate trajectory $\tau = \{s_0, s_1, \dots\}$

query expert to generate data $\mathcal{D}_i = \{(s_0, \pi^*(s_0)), (s_1, \pi^*(s_1)), \dots\}$

aggregate data sets $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$

retrain $\hat{\pi}$ using $\mathcal{D}$

For sufficiently large N, returns for the policy $\hat{\pi}$ obtained from DAgger drop by at most ϵT.

Downsides:(→policy needs to be retrained →needs access to expert policy)

Multi-Modality

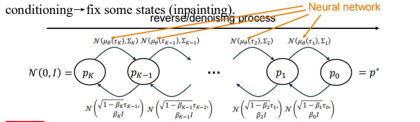
parameterize the policy $e.g., \hat{\pi} = \mathcal{N}(\mu(s), \Sigma(s))$由神经网络参数化

Diffusion Models

Idea: Interpret imitation learning as fitting a probability distribution.

μ_θ is Neural network.

• Learn/predict trajectories→increase temporal consistency; • Temporal convolution as encoder→capture temporal locality; • Include rewards in trajectories→optimization via conditioning; • Goal/initial state conditioning→fix some states (inpainting).



LEC10

State value function $V^\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right]$ 给定策略π下，从状态 s 开始，智能体所能获得的期望累积折扣奖励

Action value function $Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\right]$给定π下，从 s开始，首先采取a，从下一时刻开始遵循，所获得的期望累积折扣奖励

Connection between $V^\pi(s)$and $Q^\pi(s,a)$: $V^\pi(s) = \sum_a \pi(a \mid s) Q^\pi(s,a)$

$V^\pi(s)$等于所有可能动作 a 的$Q^\pi(s,a)$的加权平均。

Bellman Optimality 找到每个状态 s 的最优$V^*(s)$，前提是：takes the best possible action and then continues acting optimally in the future.选择使 $r(s,a) + \gamma E[V^*(s')]$最大化的动作，并假设下一状态的值$V^*(s')$也是最优

Optimal Q-value Function $Q^*(s,a) = \max_\pi E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right]$

Bellman equation:

$$Q^*(s,a) = \sum_{s'} P(s' \mid s,a) \left[r(s,a) + \gamma \max_{a'} Q^*(s', a')\right]$$

Value Iteration

$$Q_{i+1}(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[r(s,a) + \gamma \max_{a'} Q^*(s', a') | s, a\right]$$

Problem:Not scalable to cover / enumerate all the (s, a) pairs for Q(s, a)

Solution in Q Learning

Use a function approximator: Q(s, a; θ) ≈ Q*(s, a)不存储每一个值

If the function is neural network • Deep Q Learning

Curse of Dimensionality 维度灾难

• Traditional RL methods fail in large state/action spaces • Robotics tasks often have continuous states and actions • Tabular methods become infeasible • Need for function approximation

Motivation for Deep Q-Learning (DQN)

• Q-learning works well in small, discrete state spaces • Fails with high-dimensional inputs (e.g., images, robotics) • Deep neural networks approximate Q-values• Enabled breakthroughs in Atari and robotics

Deep Q Learning, DQN

Optimal Q function:

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

Loss function for SGD:

$$l(\theta; s, a) = \frac{1}{2}\Big(\underbrace{r(s,a) + \gamma \mathbb{E}_{s'}\left[\max_{a'} Q(s', a'; \theta^{old})\right]}_{\text{目标 Q 值 (Target Q-Value)}} - \underbrace{Q(s,a;\theta)}_{\text{当前 Q 值 (Current Q-Value)}}\Big)^2$$

Gradient update:

$\nabla_\theta l(\theta; s, a)$

DQN Examples (Atari)

• Objective: Complete the game with the highest score

• State: Raw pixel inputs of the game state

• Action: Game controls e.g. Left, Right

• Reward: Score increase/decrease at each time step

1. 输入(Current state s_t) 84 × 84 × 4 stack of last 4 frames 最后四帧堆叠 Preprocessing: after RGB → grayscale conversion, downsampling, and cropping

2. CNN （16 8 × 8 conv, stride 4）（32 4 × 4 conv, stride 2）

3. Fully Connected Layers, FC 展平输入 FC 来进行抽象和 Q 值计算

4. Output Layer FC-4 (Q-values): $Q(s_t, a_1), Q(s_t, a_2), Q(s_t, a_3), Q(s_t, a_4)$

CNN 处理视觉输入（解决维度灾难），用 RNN 的思想（通过堆叠多帧）来引入时序信息，FC 层输出所有可能动作的 Q 值，据此选择最高的动作

DQN Examples on Robots

•Observation: [target position in x axis (in pixels); target in y axis (in pixels); current joint 1 position (in radians); current joint 2 position (in radians)]

•Action Spaces (Discrete): [Hold current joint angle value; Increment joint 1; Decrement joint 1; Increment joint 2; Decrement joint 2; Increment joint 1 and joint 2; Decrement joint 1 and joint 2]

Training the Q-network: Experience Replay 经验回放

• Learning from batches of consecutive samples is problematic [Samples are correlated => inefficient learning 样本相关; Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops 陷入局部最优，形成不健康的负反馈循环]

• Address these problems using experience replay [Continually update a replay memory table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played 持续更新回放内存表; Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples 从回放内存中采样 minibatch 训练,打破相关性,高效利用数据]

Problem: The Q-function can be very complicated!But the policy can be much simpler.学习高维状态空间下的精确动作价值，但策略简单

Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode = 1, M do
    Initialise sequence s_1 = {x_1} and preprocessed sequenced φ_1 = φ(s_1)
    for t = 1, T do
        With probability ε select a random action a_t
        otherwise select a_t = max_a Q*(φ(s_t), a; θ)
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set s_{t+1} = s_t, a_t, x_{t+1} and preprocess φ_{t+1} = φ(s_{t+1})
        Store transition (φ_t, a_t, r_t, φ_{t+1}) in D
        Sample random minibatch of transitions (φ_j, a_j, r_j, φ_{j+1}) from D
        Set y_j = r_j                                  for terminal φ_{j+1}
                  r_j + γ max_a' Q(φ_{j+1}, a'; θ)     for non-terminal φ_{j+1}
        Perform a gradient descent step on (y_j − Q(φ_j, a_j; θ))^2 according to equation 3
    end for
end for
```

Value-based: Difficult with continuous action spaces; Indirect policy improvement via max operation; Need for explicit arg max and exploration-exploitation balancing

Policy-based methods directly learn:$\pi_\theta(a|s)$给定状态 s 时采取动作 a 概率

目标函数 $J(\theta)$ 最大化策略 π_θ 下的期望累积回报（Expected Return）

$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_t \gamma^t r_t\right]$; 直接通过调整参数$\theta$来增加预期奖励，而不是通过学习一个值函数来间接推导出策略

Policy Gradient Theorem

Gradient of expected return: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right]$

$\nabla_\theta \log \pi_\theta(a|s)$：选择的动作的对数概率的梯度；$Q^{\pi_\theta}(s,a)$该动作有多好

$\nabla_\theta \log \pi_\theta(a|s) = \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}$ Such that $\nabla_\theta \pi_\theta(a|s) = \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)$

$J(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(a|s) Q^{\pi_\theta}(s,a)$

$d^{\pi_\theta}(s)$:π_θ下的状态的平稳分布,即 agent 在长期运行中处于状态 s 的概率。

$\sum_a \pi_\theta(a|s) Q^{\pi_\theta}(s,a)$是状态 s 的值函数 $V^{\pi_\theta}(s)$

对θ求微分:$\nabla_\theta J(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a)$ 只对$\pi_\theta(a|s)$微分

为什么不对 Q 和 d 求微分:(Simplified to avoid over-complexity)

$$\nabla_\theta J(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)$$

转化为期望________________>$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right]$

Intuition Behind REINFORCE

• If an action leads to higher reward → increase probability

• If an action leads to lower reward → decrease probability

• Similar to trial-and-error learning • Policy learns from sampled trajectories

REINFORCE Algorithm

REINFORCE uses the empirical return from the episode 使用经验回报实际观察到的回报 after taking a_t in s_t: $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ 替换 Q

Monte Carlo policy gradient method

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T_i} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) G_t^i$$

对 N 条估计取平均来近似期望

Algorithm Steps: • Initialize policy parameters θ • Run episodes using $\pi_\theta(a|s)$ •Compute returns Gt (Monte Carlo cumulative reward)•Update policy: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$ • Repeat until convergence

Main Drawbacks

• High Variance in Gradient Estimates (问题:The gradient depends on full-episode returns $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$. 后果: noisy and unstable learning, especially for long or stochastic environments. 影响: Variance grows with episode length → slower convergence 高方差导致学习收敛速度慢）

• Inefficiency in Sample Usage (REINFORCE is a pure on-policy, Monte Carlo algorithm)旧数据被丢弃，环境交互成本高，样本效率低下

• Delayed Credit Assignment 更新必须等到整个轨迹完成后，不能实时学

Concept	REINFORCE	Off-policy algorithms (e.g. DQN)
Sampling	On-policy (from current π)	Off-policy (reuse old data)
Replay buffer	❌ Not compatible	✅ Essential
Why not	❌ Distribution mismatch → biased gradient	✅ Learning to just approximate the Q-function
Data efficiency	Low	High
Update target	G_t (full return)	$r + \gamma Q(s,a)$

Combination: stability + flexibility

Method	Strength	Weakness
Value-based (e.g., DQN)	Low variance, stable learning	Not suitable for continuous actions; indirect policy improvement
Policy-based (e.g., REINFORCE)	Works for continuous action spaces; directly optimizes policy	High variance, slow convergence

Actor-Critic

Policy-based part: Actor 演员是策略网络，负责学习和执行策略

Learns a parameterized policy $\pi_\theta(a|s)$, improving action selection via gradient ascent: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) A^{\pi_\theta}(s,a)\right]$

过梯度上升来改进动作选择，目标是最大化期望回报

Value-based part : Critic 评论家是一个值函数网络，负责评估演员的策略

Learns a value function to estimate future returns, reducing variance of the gradient.学一个值函数估计来估计未来回报，将这个估计值用于降低策略梯度的方差

Motivation: REINFORCE 用 Gt 估计引起的高方差可以用 learned value estimate 替代 Gt 解决→ Keeps the gradient unbiased but reduces variance

Advantage Function: How much better is this action than average?

• If $A(s,a) > 0$:increase probability of this a • If $A(s,a) < 0$:decrease

• If $A(s,a) = 0$:no change (action is average)

Make A(s, a) the most informative and stable signal for updating the policy

Why• Using Q(s,a) alone -> high variance->Includes shared baseline(to A)

• Subtracting a baseline doesn't change the expectation 无偏，期望不变

• The best baseline V(s)-> No action discrimination-> Ignores a differences

减去可以消除奖励中与动作无关的、共同的、高噪声的部分=>Variance reduction effect 方差降低(Reduce random fluctuations around the mean reward; Training becomes more stable and converge faster)

Practical Considerations (Learning rate tuning is critical; Reward normalization helps stability; Exploration vs exploitation trade-off; Use entropy regularization to encourage exploration)

Limitations(Poor exploration; On-policy inefficiency; Instability)

LEC11

Entropy-Regularized RL(Introduce entropy term to encourage exploration)

$$J(\theta) = E_{\pi_\theta}\left[\sum_t \gamma^t (r_t + \alpha \mathcal{H}(\pi(a \mid s_t)))\right]$$

Entropy: $\mathcal{H}(\pi(a \mid s)) = -E_\pi[\log \pi(a \mid s)]$

α(temperature) controls exploration – exploitation trade-off 大则探索性强

Soft Bellman Equations $V^\pi(s) = E_{a \sim \pi}[Q^\pi(s,a) - \alpha \log \pi(a \mid s)]$ Soft value adds entropy bonus → encourages policies that remain uncertain and exploratory

Comparison to Previous Methods

Q-Learning=>Q-function=>Simple, stable=>Discrete only

REINFORCE=>Policy=>continuous spaces=>High variance

Actor–Critic=>Policy,Value=>Lower variance=>Lacks exploration

SAC=>Policy + Soft Q=>Stable, efficient, exploratory=>complex(offpolicy)

Deep Deterministic Policy Gradient (DDPG)

Make the actor deterministic, but still learn via gradients from a critic.

• Pros(Works well in high-dimensional continuous spaces; Off-policy → high sample reuse; Faster convergence than stochastic actor methods

• Cons[Deterministic → poor exploration (needs noise injection); Prone to overestimation bias (one critic); Sensitive to hyperparameters → unstable]

Unstable Policy Updates=>trust region methods.

Trust Region Optimization Constrain the policy update so that the new policy is not too different from the old one $\max_\theta J(\theta)$ $s.t.$ $D_{KL}(\pi_{old} // \pi_\theta) \leq \delta$

目标仍然 $J(\theta)$，约束条件是，旧策略与新策略的 KL 散度不能超过阈值

KL 散度是分布之间的信息差 $D_{KL}(P \| Q) = \int P(x) \log \frac{P(x)}{Q(x)} dx$ 非负，不对称

$D_{KL}(P // Q)$: 衡量了使用 Q 来近似 P 时的信息损失。

$D_{KL}(Q // P)$: 衡量了使用 P 来近似 Q 时的信息损失。所以不等

TRPO Objective Function 重要性采样校正后的策略梯度目标

$L_{TRPO}(\theta) \approx E_{s,a \sim \pi_{old}}\left[\frac{\pi_\theta(a \mid s)}{\pi_{old}(a \mid s)} A^{\pi_{old}}(s,a)\right]$ Subject to $E_{s \sim \pi_{old}}[D_{KL}(\pi_{old}(\cdot \mid s) // \pi_\theta(\cdot \mid s))] \leq \delta$

调整$\pi_\theta(a \mid s)/\pi_{old}(a \mid s)$:

如果$A^{\pi_{old}}(s,a) > 0$(动作好)，增大比值，增加新策略 π_θ 选择动作 a 的概率

如果$A^{\pi_{old}}(s,a) < 0$(动作好)，减小比值，减少新策略 π_θ 选择动作 a 的概率

Importance Sampling 没有 P(x)的样本，但有 Q(x)的，权重 P(x)/Q(x)

$$E_{x \sim P}[f(x)] = \sum_x P(x) f(x) = \sum_x Q(x)\left[\frac{P(x)}{Q(x)} f(x)\right] = E_{x \sim Q}\left[\frac{P(x)}{Q(x)} f(x)\right]$$

Why Needs It: 使用重要性采样，将新策略π_θ下期望，转换为旧策略π_{old}下期望；可以在不采集新数据的情况下，计算新策略的近似性能

TRPO Derivation Intuition = policy gradient with a geometry-aware step

Linearize the surrogate objective (first-order)目标函数一阶泰勒展开

Quadratically approximate the KL constraint (second-order)KL 二阶展开

Solve via natural gradient $\theta_{k+1} = \theta_k + \sqrt{2\delta / (g^T H^{-1} g)} H^{-1} g$

Detail: $L_{TRPO}(\theta) \approx L(\theta_k) + g^T(\theta - \theta_k)$, $D_{KL}(\pi_{old} // \pi_\theta) \approx \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k)$

$g = \nabla_\theta L_{TRPO}(\theta)$ Gradient of the surrogate objective

H: KL 在新旧策略间的二阶导 it defines the trust region shape.

We use the Lagrangian to solve it:$L(\theta, \lambda) = g^T(\theta - \theta_k) - \lambda\left(\frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k) - \delta\right)$

Setting the gradient w.r.t θ to zero: $\nabla_\theta L(\theta, \lambda) = g - \lambda H(\theta - \theta_k) = 0$, $\theta_k = \theta + H^{-1} g / \lambda$

Plug back into the constraint: $1/\lambda = \sqrt{2\delta / (g^T H^{-1} g)}$

So, $\theta_{k+1} = \theta_k + \sqrt{2\delta / (g^T H^{-1} g)} H^{-1} g$

Implementation of TRPO(Collect trajectories under π_{old}; Estimate $A(s,a)$; Compute gradient g; Compute H and solve $H^{-1} g$ via conjugate gradient 计算量最大; Update parameters with line search to satisfy KL constraint)

Line Search(非线性或者步长很大就会不准确，所以使用线搜索)

full step size:$\alpha_0 = \sqrt{2\delta / (g^T H^{-1} g)}$ 沿自然梯度方向$\Delta\theta = H^{-1} g$ 减小，选择满足约束且带来最大回报改进的步长$\theta_{k+1} = \theta_k + \alpha \Delta\theta$

Advantages(Stable updates; Theoretically justified monotonic improvement; Natural gradient direction; Strong baseline method)

Limitations(Complex to implement; High computational cost;Hard to tune δ ; Hard to scale to large networks)

Proximal Policy Optimization, PPO= TRPO's lightweight, firstorder cousin (•Avoids second-order optimization--no Hessian • Achieves similar stability and performance • Easy to implement with standard gradient descent)

Clipped Objective:$L^{CLIP}(\theta) = E[\min(r_t(\theta) A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t)]$

$r_t(\theta) = \pi_\theta(a \mid s) / \pi_{old}(a \mid s)$ Don't let the prob. ratio deviate too far from 1

Full Loss Function: $L_{Total}(\theta) = L^{CLIP}(\theta) + c_1 L^{VF} - c_2 \mathcal{H}(\pi)$

Value Function Loss:L^{VF}最小化 Entropy Loss:$-\mathcal{H}(\pi)$ 没符号的最大化

Steps(Collect trajectories underπ_{old}; Estimate advantages (GAE); Compute ratio $r_t(\theta)$; Optimize clipped objective with SGD; Update $\pi_{old} \leftarrow \pi_\theta$ after several epochs) 优点 Simple to code, robust to hyperparameters

Generalized Advantage Estimation, GAE 更准确地估计A_t以减小方差

$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$ 多步 TD 残差加权求和，$\delta_t = r_t + \gamma V(S_{t+1}) - V(S_t)$

λ controls bias–variance tradeoff (0 = low bias, 1 = low variance) PPT 说γ

λ = 0; Normal A; $A_t = r_t + \gamma V(S_{t+1}) - V(S_t)$; Simple, low variance; - High bias

λ = 1; Monte Carlo A;$A_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V(S_t)$; - Unbiased; - High variance

λ = 0-1; GAE; $\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$; Bias-variance tradeoff; Controlled by λ

Explanation: (0. Relies on imperfect critic's short prediction; 1. Uses full, noisy returns; 0-1: Smooth compromise between both)

Compare:Constraint;Optimization;Computation;Performance;Implemented

TRPO:KL-divergence (hard constraint);Second-order (natural gradient); Expensive; Very stable; Theory-focused works

PPO:Clipping (soft constraint); First-order (SGD); Lightweight; Almost as stable, faster; Practically all modern RL (OpenAI Baselines)

Why PPO? (Stability:Clipping prevents large updates; Simplicity:No need for KL constraint or FIM; Efficiency:Uses multiple epochs of on-policy data; Exploration:Maintains entropy term; Universality:discrete & continuous)

Limitations of PPO (On-policy Inefficiency; Constraints guarantee; Sensitivity to hyperparameters; Value function instability; Limited exploration; Black-box system)

基础公式以及方法分类

$$V^*(s) = \max_a Q^*(s,a)$$

Bellman Optimality Equation:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s')$$

星号代表所有可能策略下最大期望回报

Bellman Equation:

$$V^\pi(s) = \sum_a \pi(a \mid s) Q^\pi(s,a)$$

对固定策略，用于评估策略价值

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^\pi(s')$$

$$Q^\pi(s,a) = \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma \sum_{a'} \pi(a' \mid s') Q^\pi(s',a')\right]$$

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

$$V^*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V^*(s')]$$

$$Q^*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} Q^*(s',a')]$$