PPO tutorial

强化学习是什么?

- **智能体（Agent）**：做决策的主体（车辆）。
- **环境（Environment）**：智能体所处的世界，初步为自定义的high-way开源环境
- **状态（State, s）**：环境在某一时刻的描述，在我们当前的case中，即自车状态+环境拓扑。
- **动作（Action, a）**：智能体在状态 s 下可以执行的决策，在我们的当前case中，即根据生成的X,Y映射的action。
- **奖励（Reward, r）**：智能体执行动作后，环境返回的反馈信号（正向鼓励或负向惩罚）。
- **目标**：让智能体学习一个策略 π(a|s)，在长期累计奖励（return）上表现最优。

RL to DRL

阶段	核心瓶颈	关键推进	代表算法
经典 RL（表格法）	状态、动作离散且规模小	使用查表计算价值或策略	Q-Learning, SARSA
函数逼近 RL	连续或高维状态出现	用线性/径向基展开逼近价值	LSPI, Fitted Q
深度 RL (DRL)	状态/动作极高维（图像、激光点云）	使用深度神经网络做近似(策略或价值)，实现端到端学习	DQN, A3C, PPO, SAC

Example

- Q-Table（表格 Q-Learning）

采用二维表 Q(s,a) 显式存储每个状态-动作的价值，循环更新

[核心特征：易实现、样本效率高；状态-动作必须可枚举 ⇒ 仅适合超小规模问题。]

- Fitted Q / LSPI（线性或树模型 Q 逼近）

将“查表”替换为函数逼近（多项式基、回归树、线性回归），一次性最小二乘拟合 Q

- DQN（Deep Q-Network）

用神经网络端到端估计Q（替代Q-Table），引入 [经验回放 + 目标网络] 稳定训练

- A3C / IMPALA（分布式 Actor–Critic）

多线程或多机 Actor 并行采样、Learner 异步更新；

吞吐量高、收敛快，可在 3D 导航 / StarCraft II 等复杂环境中取得 SOTA。

分类

Reinforcement Learning

├── Model-based

│ ├── Plan/Search (MCTS, Dyna, **AlphaZero**)

│ └── Learn Dynamics + Planning

└── Model-Free

├── Value-Based

│ ├── Tabular（Q-Learning, SARSA)

│ └── Deep（DQN, Rainbow, QR-DQN, Apex-DDQN, R2D2)

└── Policy-Based

├── REINFORCE

└── Actor–Critic

├── Trust-Region（TRPO)

├── Proximal（PPO) ★

├── Entropy-Regularized（SAC)

└── Distributed（IMPALA, A3C)

*工业界常用方法用黄色背景表示。

DRL 方法选择

在DRL中，解决问题的主流思路有两类：

值函数方法（Value-based）

核心思想：

- Value-based方法通过学习**价值函数（Value Function）**，来估计“在当前状态下选择每个动作，未来能够得到的长期累计奖励”。
- 最典型的方法是 **DQN（Deep Q-Network）**，它为每个动作计算一个价值，再选择 $Q(s,a)$ 价值最高的动作。

适用场景：

- 动作空间较小且**离散**的情况（如经典的游戏场景，围棋、象棋，或自动驾驶中离散的决策如左转、右转、停车）。
- 问题规模不太大，比如动作控制器（motion planner）

优点：样本效率高，适合离散小动作。

局限：连续动作需离散化 维度爆炸；难直接约束策略平滑。

策略梯度方法（Policy-based）

核心思想：

- Policy-based方法直接训练一个**策略网络（Policy Network）** $\pi(a|s)$，策略网络直接输出在给定状态下的动作或动作的概率分布，算法优化 $J(\theta)$。
- 典型的方法如**PPO（Proximal Policy Optimization）** 和 **SAC（Soft Actor-Critic）**。

适用场景：

- 复杂、高维状态空间下（例如用视觉、雷达、激光雷达作为输入的自动驾驶任务），策略梯度方法通常表现更稳定，更容易收敛。

优点：连续动作原生支持，收敛稳定，可加入熵等正则。

缺点：方差大，需要 Critic 降方差。

PPO（Proximal Policy Optimization）

设计目标

- **小步更新**：避免策略一次更新过大而“崩溃”。
- **计算效率**：PPO近似等价于一阶优化, 而TRPO（Trust Region Policy Optimization）需要二阶优化（计算量大）。
    - TRPO: **Trust Region Policy Optimization**
        - 约束新旧策略 KL 散度。
        - 采用二阶优化（共轭梯度 + Fisher 信息矩阵乘向量），数学优雅但实现复杂、计算重。
        - 目前极少使用
- **优势区间**：适合Alignment Fine-Tuning(目标对齐)
    - **预训练**：快速学到 “ 能开车 ” —— 产生合理轨迹。
    - **目标对齐**：进一步学会“把车开得好”——在定制奖励下优化长期收益。

算法核心

策略裁剪

限制新旧策略比值 $r_t(\theta) = \pi_\theta / \pi_{\theta_k} \in [1-\varepsilon, 1+\varepsilon]$

Actor–Critic 架构

Backbone

将高维输入（图像、点云、历史轨迹）编码为低维特征；在当前case中，就是我们的encoder

Actor

生成动作策略 $\pi_\theta(a|s)$，在当前case中，就是我们的decoder，输出3步控制转Aciton

Critic

估计状态价值 $V_\phi(s)$ 或动作价值 $Q_\phi(s,a)$，用于指导 Actor 更新

代码块

```
┌──────────────────── Shared Backbone (CNN / Transformer / MLP) ────────────────────┐
│                                                                                   │
│   features = fθ(s)                                                                │
│                                                                                   │
├──────────────────── Policy Head (Actor) ──────────────┬──────────── Value Head (Critic) ────────────┤
│                                                      │                             │
│   logits / μ,σ = Wπ · features                       │   Vϑ(s) = Wv · features     │
│   πθ(a|s)  = squashed_dist(logits)                   │   (or Qϑ(s,a) for Twin-Q / SAC)  │
│                                                      │                             │
└──────────────────────────────────────────────────────┴─────────────────────────────┘
```

伪代码

代码块

```
actor = TrajGen()
critic = TrajGen().copy()
optimizer_actor = Adam(actor.parameters(), lr=3e-5)
optimizer_critic = Adam(critic.parameters(), lr=1e-4)

for iter in range(iters):
    # collect data (on-policy)
```

```
    traj_batch = []
    # use first 3 step pred to gen action
    for _ in range(max frames=30/3):
        states, actions, rewards = run_episode(actor, env)
        traj_batch.append((states, actions, rewards))

    # cal hat A_t
    adv_batch, ret_batch = compute_GAE(traj_batch, critic)

    # update Actor
    for _ in range(K_actor):
        for mini in mini_batch(traj_batch):
            ratio = actor.log_prob(mini.actions) - mini.old_log_probs
            clip_adv = torch.clamp(ratio.exp(), 1-ε, 1+ε) * mini.adv
            loss_actor = -torch.min(ratio.exp()*mini.adv, clip_adv).mean()
            optimizer_actor.zero_grad(); loss_actor.backward();
optimizer_actor.step()

    # update Critic
    for _ in range(K_critic):
        v_pred = critic(mini.states)
        loss_critic = F.mse_loss(v_pred, mini.ret)
        optimizer_critic.zero_grad(); loss_critic.backward();
optimizer_critic.step()
```

Reward 设计(上下游reward/目标对齐)

潜在对齐目标:

a. 转弯半径
b. 平滑度
c. 达成率
d. 绕行幅度