條

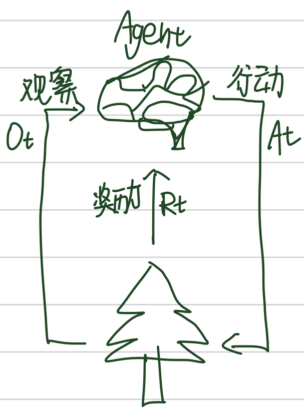
因为以前看过一些有趣的强化学习视频，对此较感兴趣，稍微试一下。能走到哪一步就不知道了。

任务：

Gym-Taxi项目

- 选择至少一个RL算法建立智能体(Agent)，在Taxi环境中训练和测试，记录测试结果并可视化
- 试图使Agent收敛（在每一个episode中，可以稳定的将乘客送达目的地），如果不能得到收敛的结果，分析原因并继续尝试，记录算法的改进过程
- 关注你使用的算法中的细节（例如动作的探索(Exploration)和利用(Exploitation)、经验回放(Experience Replay)），分析它们在该算法中被使用的原因
- 了解Reward shaping技巧，尝试在训练过程中使用它，并记录它带来的影响。

决策
交互
奖励
延迟



强化学习智能体分类：

- 基于价值
  - 没有策略（隐含）
  - 价值函数
- 基于策略：
  - 策略
  - 没有价值函数
- Actor-Critic
  - 策略
  - 价值函数

police $\pi(a|s)$

☆ $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

未来奖励的累加

S 状态

a 行动

☆ $Q_\pi(s_t, a_t) = E[U_t | S_t = s_t, A_t = a_t]$ （积分）

动态价值函数　　已知函数π，对 st 状态下的动作 at 打分.

☆ $Q^*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$　　Q-star

最优价值函数　　消去π，评价对 st 状态下的动作 at 打分

☆ $V_\pi(S_t) = E_A[Q_\pi(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a) = \int \pi(a|s_t) \cdot Q_\pi(s_t, a)\, da$

状态价值函数　　对 a 积分消去，评价当前状态的好坏

来自王树森教程

价值学习 Value-Based Reinforcement Learning

$Q^*(s,a)$

try 找到 $a^* = \underset{a}{\arg\max}\, Q^*(s,a)$

但是不知道 $Q^*(s,a)$

解决方法：DQN

使用神经网络 $Q(s,a;\vec{w})$ 来估计 $Q^*(s,a)$

DQN

TD算法

DQN过程

$s_{t+1} \sim p(\cdot \mid s_t, a_t)$ $s_{t+2} \sim p(\cdot \mid s_{t+1}, a_{t+1})$ $s_{t+3} \sim p(\cdot \mid s_{t+2}, a_{t+2})$

$s_t \to a_t \to s_{t+1} \to a_{t+1} \to s_{t+2} \to a_{t+2} \to s_{t+3}$

r_t r_{t+1} r_{t+2}

$a_t = \arg\max_a Q(s_t, a; w)$

$a_{t+2} = \arg\max_a Q(s_{t+2}, a; w)$

$a_{t+1} = \arg\max_a Q(s_t, a; w)$

$$Q(s_t, a_t; w) \approx r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; w)$$

prediction: $Q(s_t, a_t; w_t)$

TD target: $y_t = r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; w_t)$

$\qquad = r_t + \gamma \cdot \max_a Q(s_{t+1}, a, w_t)$

Loss: $L_t = \frac{1}{2}[Q(s_t, a_t; w_t) - y_t]^2$

Gradient descent: $w_{t+1} = w_t - \alpha \cdot \frac{\partial L_t}{\partial w}\Big|_{w=w_t}$.

策略学习 Policy-Based Reinforcement Learning

Policy Function $\pi(a|s)$

$$V_\pi(s_t) = E_A[Q_\pi(s_t, A)] = \Sigma_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$$

Approximate policy function $\pi(a|s_t)$ by policy network $\pi(a|s_t;\theta)$

Approximate value function $V_\pi(s_t)$ by:

$$V(s_t;\theta) = \Sigma_a \pi(a|s_t;\theta) \cdot Q_\pi(s_t, a).$$

policy-based learning:

find θ that maximizes $J(\theta) = E_S[V(S;\theta)]$

Update θ by: $\theta \leftarrow \theta + \beta \cdot \boxed{\frac{\partial V(s;\theta)}{\partial \theta}}$

Policy Gradient (two forms)

$$\frac{\partial V(s;\theta)}{\partial \theta} = \Sigma_a \frac{\partial \pi(a|s;\theta)}{\partial \theta} \cdot Q_\pi(s,a)$$

$$= \Sigma_a \pi(a|s;\theta) \cdot \frac{\partial \log \pi(a|s;\theta)}{\partial \theta} \cdot Q_\pi(s,a)$$

$$= E\left[\frac{\partial \log \pi(A|S;\theta)}{\partial \theta} \cdot Q_\pi(S,A)\right] \qquad E_{A\sim\pi(\cdot|s;\theta)}$$

Actor-Critic Methods 价值与策略的结合

$\boxed{\text{Definition:}}$ $V_\pi(s) = \sum_a \pi(a|s) \cdot Q_\pi(s,a) \approx \sum_a \pi(a|s;\theta) \cdot q(s,a;w)$

$\boxed{\text{Policy network (actor):}}$

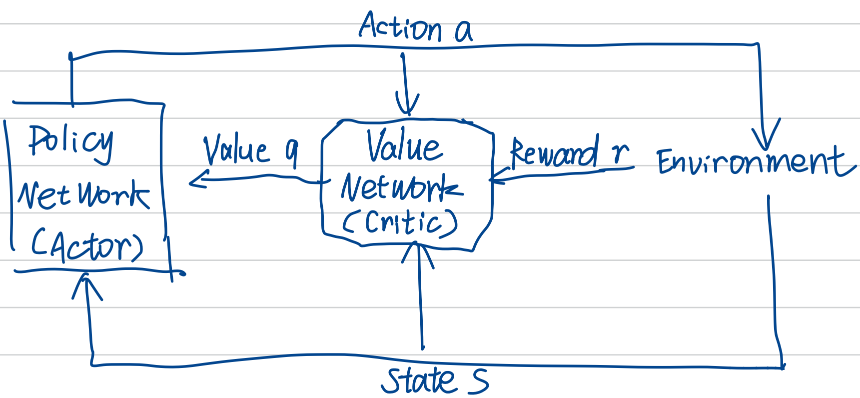
Use neural net $\pi(a|s;\theta)$ to approximate $\pi(a|s)$

θ: trainable parameters of the neural net.

$\boxed{\text{Value network (Critic)}}$:

Use neural net $q(s,a;w)$ to approximate $Q_\pi(s,a)$

w: trainable parameters of the neural net



Summary of Actor-Critic method

1. Observe state s_t and randomly sample $a_t \sim \pi(\cdot \mid s_t; \theta_t)$;
2. Perform a_t; then environment gives new state s_{t+1} and reward r_t.
3. Randomly sample $\tilde{a}_{t+1} \sim \pi(\cdot \mid s_{t+1}; \theta_t)$. (Do not perform $\tilde{a}_{t+1}$!)
4. Evaluate value network: $q_t = q(s_t, a_t; w_t)$ and $q_{t+1} = q(s_{t+1}, \tilde{a}_{t+1}; w_t)$
5. Compute TD error: $\delta_t = q_t - (r_t + \gamma \cdot q_{t+1})$
6. Differentiate value network: $d_{w,t} = \frac{\partial q(s_t, a_t; w)}{\partial w} \Big|_{w=w_t}$
7. Update value network: $w_{t+1} = w_t - \alpha \cdot \delta_t \, d_{w,t}$.
8. Differentiate policy network: $d_{\theta,t} = \frac{\partial \log \pi(a_t \mid s_t, \theta)}{\partial \theta} \Big|_{\theta=\theta_t}$
9. Update policy network: $\theta_{t+1} = \theta_t + \beta \cdot q_t \cdot d_{\theta,t}$.

Action Space 来自某教程

Discrete (离散) Single Process: DQN

Discrete Multi Processed: PPO A2C

Continuous Single Process: SAC TD3

Continuous Multi Processed: PPO A2C

时序差分算法：(伪代码)　　来自网上查阅资料

$\boxed{\text{Tabular TD(0) for estimating } v_\pi}$

Input: the policy π to be evaluated

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$

Initialize $V(s)$, for all $s \in S^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop for each episode:

    Initialize S

    Loop for each step of eposode:

        $A \leftarrow$ action given by π for S

        Take action A, observe R, S'

        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$

        $S \leftarrow S'$

    until S is terminal

在线策略
(一致)

$\boxed{\text{Sarsa (On-policy TD control) for estimating } Q \approx q}$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$

Initiate $Q(s,a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(\text{terminal}) = 0$

Loop for each episode:

    Initialte S

    Choose A from S using policy delived from Q (e.g., ε-greedy)

    Loop for each step of episode:

        Take action A, observe R, S'

        Choose A' from S' using policy derived from Q (eg. ε-greedy)

        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$

        $S \leftarrow S'$; $A \leftarrow A'$;

    until S is terminal

离线策略
策略不一样

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

Algorithm parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$

Initialize $Q(s,a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:

    Initialize S

    Loop for each step of episode:

        Choose A from S using policy derived from Q (e.g., ϵ-greedy)

        Take action A, observe R, S'

        $Q(S,A) \leftarrow Q(S,A) + \alpha [R + \gamma \max_a Q(S',a) - Q(S,A)]$

        $S \leftarrow S'$

    until S is terminal

Taxi 问题

理应可以通过Q-learning方法学习
我看到现在，对这种算法的理解大体是

建立一个Q-table

	s	s	s	s
a	Q	Q	Q	Q
a	Q	·	·	·
a	Q	·	·	·
a	Q	·	·	Q

表格中表示不同状态下采取不同行动的最优价值函数

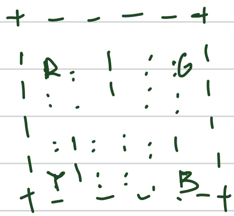
通过学习，将整个表格不断更新，最后会得到所有状态下的最优动作，将此作为策略就是得到的结果了

一开始很不理解这个过程，我原本以为是要从头那种一直迭代递推，Q的最优值一定要有先后才行什么的想法，可能是误把动态规划东西等来想了 而且也没理解这个表格是怎么能得到最优策略的

现在的理解大概是，整个表每次都同时更新，这样就避免了判断先后步骤的问题，更新的时候就收敛的顺序还是与步骤有关系。然后这个表格等于就是试错试错再试错，把所有状态（这个状态包括车辆的状态，位置、乘客状态，目的地位置），也就是所有可能出现的有一点不同的情况，都求出一个解。

Taxi: 环境:

MAP:



Action: 0: move south
1: move north
2: move east
3: move west
4: pickup passenger
5: drop off passenger

Rendering: blue: Passenger
magenta: destination
yellow: empty taxi
green: full taxi
other: (R, G, Y and B): locations for passengers and destinations

State space is represented by:
(taxi-row, taxi-col, passenger-location, destination)

Reward: -1 per step reward unless other reward is triggered
+20 delivering passenger
-10 executing "pickup" and "dropoff" actions illegally

State:
(int(s), r, d, {"prob": p})

其实感觉做这种更难的应该是如何把实际问题怎么抽象出来，把动作，状态、奖励机制什么的封装起来，使其变为一个能解决的问题。后面的部分其实大差不差，确定了方法之后后续代码是差不多的。

当然这是目前的我这时的想法，如果想吃透后面并能够创新还是挺难的吧。这个gym封装好的环境就已经把前半部分比较的问题解决好了。