Part1 Key Concepts in RL

In a nutshell, RL is the study of agents and how they learn by trial and error. It formalizes the idea that rewarding or punishing an agent for its behavior makes it more likely to repeat or forego that behavior in the future.

(完成一个关卡后，收集难易程度的选择，来进行下一次选择，自适应，不断调整)

What Can RL Do?

1. It's also famously been used to create breakthrough AIs for sophisticated strategy games, most notably Go and Dota, taught computers to play Atari games from raw pixels, and trained simulated robots to follow human instructions.

Key Concepts and Terminology

1. main characters: agent and environment
2. At every step of interaction, agent sees an observation of the state of the world, and decides on an action to take.

   Environment changes -> agent may change on its own
3. agent会从环境获取一个reward signal，这个信号用来描述当前世界状态的好坏程度。agent的目标就是尽可能获取更多的reward，称之为return。强化学习方法就是agent为了达到它的目标而去学习一些行为。

States and Observations

1. **state s** : a complete description of the state of the world

   observation o : a partial description of a state (omit some information)
   - always represent them by a real-valued vector, matrix or higher-order tensor *(For instance, a visual observation could be represented by the RGB matrix of its pixel values; the state of a robot might be represented by its joint angles and velocities.)*
2. **fully observed**: the agent is able to observe the complete state of the environment

   partially observed: can only see a partial observation

Action Spaces

1. **action space**: the set of all valid actions in a given environment

   discrete action spaces: Some environments, *like Atari and Go*, where only a ***finite*** number of moves are available to the agent.

   continuous action spaces: actions are ***real-valued vectors***, *like the agent controls a robot in physical world.*
   - *This distinction has some quite-profound consequences for methods in deep RL. Some families of algorithms can only be directly applied in one case, and would have to be substantially reworked for the other.*

Policies

1. **policy**: a rule used by an agent to decide what actions to take
    - **deterministic**: in which case it is usually denoted by μ

      $$a_t = \mu(s_t)$$

    - **stochastic**: in which case it is usually denoted by π

      $$a_t \sim \pi(\cdot \mid s_t)$$

    - it is common to substitute "policy" for "agent" *(eg "The policy is trying to maximize reward")* , because the policy is essentially the agent's brain.
2. **parameterized policies**: (in deep RL) policies whose outputs are computable functions that depend on a set of parameters *(eg the weights and biases pf a neural network)* which we can adjust to change the behavior via some optimization algorithm.
    - the parameters often are denoted as θ or ϕ, and then write this as a subscript on the policy symbol to highlight the connection:
        - $a_t = \mu_\theta(s_t)$
        - $a_t \sim \pi_\theta(\cdot \mid s_t)$

Deterministic Policies

1. **Example:** (jump back to here later)

Stochastic Policies

0. **Two key computations** are centrally important for using and training stochastic policies:
    - ***sampling*** actions from the policy,
    - and computing ***log likelihoods*** of particular actions, $\log \pi_\theta(a|s)$
1. **categorical（明确的） policies**: in discrete action space
    - like a classifier over discrete actions, u build the neural network for a categorical policy the same way u would for a classifier： 输入即是一个观察，followed by some number of layers， 然后就有一个最终的线性层，这个层会给你每个action的logits，softmax 会把logits转化成probabilities
    - **sampling** *Given the probabilities for each action, frameworks like PyTorch and Tensorflow have built-in tools for sampling.*
    - **log-likelihood** 把probabilities的最后一层记为 $P_\theta(s)$，很多actions都会以这个vector作为entries，所以我们可以把这些actions作为这个vector的indices（指标）。action a 的 log likelihood 就可以通过将vector进行索引来获得：

      $$\log \pi_\theta(a|s) = \log[\, P_\theta(s) \,]_a$$

2. **diagonal Gaussian policies**: in continuous action space
    - (refer to multivariate normal distribution, jump back to here later)
3. **?** logit回归？ log likelihood对数似然?

Trajectories

1. **trajectory** τ : a sequence of states and actions in the world

   $\tau = (s_0, a_0, s_1, a_1, \ldots)$

    - The very first state of the world, s_0, is randomly sampled from the **start-state distribution**, sometimes denoted by $\rho 0$:

      $$s_0 \sim \rho_0(\cdot)$$

2. **state transitions**: what happens to the world between the state at time t, (s_t), and the state at t+1, (s_{t+1})
    - are governed by the natural laws of the environment and depend on ***only*** the most recent action, a_t
    - they can be either ***deterministic***

      $s_{t+1} = f(s_t, a_t)$
    - or ***stochastic***

      $s_{t+1} \sim P(\cdot \mid s_t, a_t)$
    - (actions come from an agent according to its policy)
3. trajectories are also frequently called **episodes** or **rollouts**

Reward and Return

1. **R** : the reward function *(critically important)*
2. reward depends on the current state of the world, the action just taken, and the next state of the world

   $r_t = R(s_t, a_t, s_{t+1})$

   although frequently this is simplified to just a dependence to the current state, $r_t = R(s_t)$, or state-action pair $r_t = R(s_t, a_t)$
3. **the goal of agent**: maximize some notion of cumulative reward over a trajectory
    - mean a few things, so notating all these cases with $R(\tau)$
4. **two kinds of return**
    - **finite-horizon undiscounted return**: just the sum of rewards obtained in a fixed window of steps

      $R(\tau) = \sum_{t=0}^{T} r_t$
    - **infinite-horizon discounted return**: the sum of all rewards *ever* obtained by the agent, but ***discounted*** by how far off in the future they're obtained**?**. This formulation of reward includes a discount factor $\gamma \in (0, 1)$:

      $R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$
        - why would we ever want a discount factor?
            - ***intuitively***: cash now is better than cash later
            - ***mathematically***: an infinite-horizon sum of rewards may not converge to a finite value, and is hard to deal with in equations. But with a discount factor and under reasonable conditions, the infinite sum converges.
    - *While the line between these two formulations of return are quite stark in RL formalism, deep RL practice tends to blur the line a fair bit—for instance, we frequently set up algorithms to optimize the undiscounted return, but use discount factors in estimating* ***value functions****.*

The RL Problem

1. Whatever the choice of return measure (whether infinite-horizon discounted, or finite-horizon undiscounted), and whatever the choice of policy, **the goal in RL** is to select a policy which maximizes **expected return** when the agent acts according to it.
2. **probability distributions over trajectories -> expected return**

- suppose both the environment transitions and the policy are stochastic ($s_{t+1} \sim P(\cdot \mid s_t, a_t)$ and $a_t \sim \pi(\cdot \mid s_t)$). In this case, the ***probability of a T-step trajectory*** is:

  $p(\tau \mid \pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$

  $0-T-1$的下一状态的可能性和这一状态下采取的$action$的可能性的累积

- ***the expected return*** (for whichever measure), denoted by **J(π)**, is then:

  $J(\pi) = \int_\tau P(\tau|\pi)R(\tau) = E_{\tau\sim\pi}[R(\tau)]$（$E$为数学期望）

3. **the central optimization problem in RL** can then be expressed by

   $\pi^* = arg\ max_\pi J(\pi)$ (π^* being the **optimal policy**)

Value Functions

1. **value** of a state or state-action pair: the expected return if u start in that state or state-action pair, and then act according to a particular policy forever after.
2. **value functions** are used in almost every RL algorithm, there are ***four*** main functions:
   - the **On-Policy Value Function** $V^\pi(s)$ (基于特定$policy$，即π) , which gives the expected return if u start in state ***s*** and always act according to policy π:

     $V^\pi(s) = E_{\tau\sim\pi}[R(\tau) \mid s_0 = s]$

   - the **On-Policy Action-Value Function** $Q^\pi(s,a)$, which gives the expected return if you start in state ***s***, take an arbitrary action ***a*** (which may not have come from the policy), and then forever after act according to policy π:

     $Q^\pi(s,a) = E_{\tau\sim\pi}[R(\tau) \mid s_0 = s, a_0 = a]$

   - the **Optimal Value Function** $V^*(s)$, which gives the expected return if you start in state ***s*** and always act according to the *optimal* policy in the environment:

     $V^*(s) = max_\pi\ E_{\tau\sim\pi}[R(\tau) \mid s_0 = s]$

   - the **Optimal Action-Value Function** $Q^*(s,a)$, which gives the expected return if u start in state ***s***, take an arbitrary action ***a***, and then forever after act according to the *optimal* policy in the environment:

     $Q^*(s,a) = max_\pi\ E_{\tau\sim\pi}[R(\tau) \mid s_0 = s, a_0 = a]$

3. *When we talk about value functions, if we do not make reference to time-dependence, we only mean expected **infinite-horizon discounted return**. Value functions for finite-horizon undiscounted return would need to accept time as an argument. Can you think about why? Hint: what happens when time's up?*
4. there are two key connections (can be directly proved via definition):
   - $V^\pi(s) = E_{a\sim\pi}[Q^\pi(s,a)]$
   - $V^*(s) = max_a\ Q^*(s,a)$

The Optimal Q-Function and the Optimal Action

1. There is an important connection between the optimal action-value function $Q^*(s,a)$ and the action selected by the optimal policy.
2. If we have Q^*, we can directly obtain the optimal action, $a^*(s)$, via

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

   ($argmax$为）后面那个函数取最大值的时候参数a的值

3. *Note: there may be multiple actions which maximize $Q^*(s, a)$, in which case, all of them are optimal, and the optimal policy may randomly select any of them. But there is always an optimal policy which deterministically selects an action.*

Bellman Equations

1. **bellman equations:** all four of the value functions obey these special self-consistency equations. The basic idea behind Bellman equations is:

   The value of your starting point is the reward you expect to get from being there, plus the value of wherever you land next.

2. the Bellman equations for the on-policy value functions are:

   - $V^{\pi}(s)$ = $E_{a\sim\pi\ s'\sim P}$ [$r(s,a) + \gamma V^{\pi}(s')$]
   - $Q^{\pi}(s,a)$ = $E_{s'\sim P}$ [$r(s,a) + \gamma\ E_{a'\sim\pi}$ [$Q^{\pi}(s', a')$]]

   Note:

   s' ~P : s' ~ P (· | s, a) , indicating that the next state s' is sampled from the environment's transition rules

   a ~ π : a ~ π (· | s); a' ~ π : a' ~ π (· | s')