Definition of RL: Agent-oriented learning, learning by **interacting with an environment** to achieve a goal. Learning by **trial and error**, with only delayed evaluative **reward**
$ No supervisor, only a reward signal $ Feedback is delayed $ Time matters, i.e. sequential decision making $ Agent actions will affect subsequent data/feedback from the environment
Elements of RL: Policy Π, reward and return, value function, model
RL Problem is a considerable abstraction of the problem of goal-directed learning from interaction with the environment.
RL methods/solutions specify how the agent changes its policy as a result of its experiences.
The agent's goal is to maximize the total amount of reward it receives over the long run.
N-armed Bandit Problem: repeatedly take a choice, reward from a stationary probability distribution. Objective is **maximize** the total reward. **Action-Value Methods** can be defined as $Q_t(a) = \frac{R_1+R_2+\ldots+R_{K_a}}{K_a}$, where $Q_t(a)$ is estimated value of action "a" at the "t-th" play, K_a is the time that action "a" has been chosen, R_1 means the first reward.
Policies: **Greedy policy**, always choose the machine(action) with current best reward. **ϵ-greedy policy** choose machine with current best expected reward with probability $1-\epsilon$. Greedy performs *worse* in long run since it often gets stuck performing *suboptimal action*. (exploitation vs. exploration dilemma) ϵ-greedy balance the dilemma.
SoftMax: choose action "a" at "t-th" play with probability, use Gibbs(or Boltzmann). $\frac{e^{Q_t(a)/\tau}}{\sum_{i=1}^{n} e^{Q_t(i)/\tau}}$, where $Q_t(a)$ is the current average rewards for machine a, τ is a positive parameter called the temperature. Large τ: nearly equal probability selection. Small τ: greedy.
Algorithm Implementation:

$$Q_{k+1} = \frac{1}{k}\sum_{i=1}^{k} R_i = \frac{1}{k}(R_k + \sum_{i=1}^{k-1} R_i = \frac{1}{k}(R_k + kQ_k - Q_k)$$
$$= Q_k + \frac{1}{k}(R_k - Q_k)$$

Q_k is the average reward for all **previous** state. $\frac{1}{k}$ is step size. R_k is target.
Use constant step size for **non-stationary problem**. Step size parameter $a_t(a)$: $\sum_{k=1}^{\infty} a_k(a) = \infty \;\; and \;\; \sum_{k=1}^{\infty} a_k^2(a) < \infty$
Markov Property: the future is independent from the past when the present is given. **Markov Process** is a memoryless process, a sequence of random states with the Markov property, tuple $< S,\ P >$
State Transition Matrix: $P = \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ P_{n1} & \cdots & P_{nn} \end{bmatrix}$
Markov Reward Process: tuple $< S, P, R, \gamma >$, S is finite set of states, P is state transition matrix, R is reward function $R_s = \mathbb{E}[R_{t+1}|S_t = s]$, γ is discount factor
Return: the total discounted reward from time-step t. $G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. γ close to 0 leads to myopic evaluation, close to 1 leads to far-sighted evaluation.
State-Value Function: expected return starting from state s. $v(s) = \mathbb{E}[G_t|S_t] = s$
Bellman Expectation Equation: $v(s) = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1})|S_t = s]$, $v(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s')$
Markov Decision Process: is a MRP with decision, with tuple $< S, A, P, R, \gamma >$, A is a finite set of actions.

Tong Wu
tw2906

Policy Π: is a distribution over actions given states $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$. In MDP, $P_{s,s'}^{\pi} = \sum_{a \in A} \pi(a|s) P_{ss'}^{a}$, $R_s^{\pi} = \sum_{a \in A} \pi(a|s) R_s^a$
Value Functions for MDP: $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$, $q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$, $q_\pi(s,a)$为策略π下的行动值函数.
Optimal Value Functions: 最优状态值函数$v_*(s) = max v_\pi(s)$, 最优行动值函数$q_*(s,a) = max\big(q_\pi(s,a)\big)$. Optimal value specify the best performance in MDP.
Optimal Policy: $\pi_*(a|s) = \begin{cases} 1 \text{ if } a = argmax_{a \in A} q_*(s,a) \\ 0 \text{ otherwise} \end{cases}$
Bellman Optimality Equation: $V_*(s) = max q_*(s,a)$, $q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$
Dynamic Programming: a collection of algorithms that simplify a complicated problem by breaking it down into simpler sub-problems in a recursive manner
Policy Evaluation/Prediction: Problem is to evaluation a given policy π. Solution is iterative application of Bellman expectation backup $v_1 \to v_2 \to \ldots \to v_\pi$. Using **synchronous backups**: at each iteration k+1, for all state s, update $v_{k+1}(s)$ from $v_k(s')$.

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$
$$v^{k+1} = R^\pi + \gamma P^\pi v^k$$

Policy Improvement: evaluate policy by $v_\pi = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s]$, improve the policy by acting greedily $\pi' = greedy(v_\pi)$. Policy iteration always converges to $\pi *$
Policy Iteration/Control: iterative policy evaluation + Greedy policy improvement. Converge to $v *, \pi *$
Generalised Policy Iteration(GPI): Any policy evaluation + any policy improvement. Converge to $v *, \pi *$
Value iteration: $v_{k+1}(s) = max R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')$, which is bellman optimality equation.
Asynchronous DP: idea: in-place DP, prioritized sweeping, real-time DP.
Monte Carlo RL: learn directly from episodes of experience; model-free, which no knowledge of MDP transition/rewards; learn from complete episodes, no bootstrapping 引导; uses the simplest possible idea: value = mean return. Can only apply MC to episodic MDPs, which all episodes must terminate.
MC Policy Evaluation/Prediction: Goal: learn v_π from episodes of experience under policy π. MC policy evaluation uses empirical mean return instead of expected return 基于样本返回值的均值而不是预期均值.
First/Every Visit MC Policy Evaluation: Increment counter $N(s)$++, increment total return $S(s)$+=G_t, value is estimated by mean return $V(s) = S(s)/N(s)$. For large numbers, $V(s) \to v_\pi(s)$ as $N(s) \to \infty$
Recursive Mean Computation 递归平均值计算： For the mean μ of a sequence x can be computed incrementally: $\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$
Recursive MC Updates: For each state S_t with return G_t: $N(S_t)$+= 1, $V(S_t)$+= $\frac{1}{N(S_t)}\big(G_t - V(S_t)\big)$
MC backup: $V(S_t)$+= $\alpha\big(G_t - V(S_t)\big)$
MC Policy Improvement: Use Model-free GPI with MC evaluation: greedy policy improvement over V(s) needs a model, while over Q(s,a) is model-free: $\pi'(s) = argmax Q(s,a)$
ϵ-Greedy: $\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon \text{ if } a^* = argmax\ Q(s,a) \\ \epsilon/m \text{ otherwise} \end{cases}$,
where all m actions are tried with non-zero probability, probability $1-\epsilon$ choose greedy action.

MC Policy Iteration: Policy Evaluation (MC policy evaluation, $Q + q\pi$) + Policy Improvement (ϵ-greedy policy improvement)
Greedy in the Limit with Infinite Exploration (GLIE) Policy Iteration: All state-action pairs are explored infinitely many times, $lim_{k\to\infty} N_k(s,a) = \infty$. The policy converges on a greedy policy, $lim_{k\to\infty}\pi_k(a|s) = 1\big(a = argmaxQ_k(s,a')\big)$
Evaluation: $N(S_t,A_t)\mathrel{+}= 1,\ Q(S_t,A_t)\mathrel{+}= \frac{1}{N(S_t,A_t)}\big(G_t - Q(S_t,A_t)\big)$
Improvement: $\epsilon = \frac{1}{k}, \pi = \epsilon\text{-greedy}(Q)$
Two equations converge to $q_*(s,a)$
On policy: learn π from experience sampled from π
Off policy: learn π from experience sampled form μ
Importance Sampling: a general technique for estimating expected values under one distribution given samples from another
Temporal Difference (TD) Learning: TD methods: learn directly from episodes of experience; model-free; learns from **incomplete** episodes, by bootstrapping; updates a guess towards a guess.
Simplest TD learning algorithm: Update value $V(S_t)$ toward estimated return
$R_{t+1} + \gamma V(S_{t+1})$: $V(S_t)\mathrel{+}= \alpha\big(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big)$, $R_{t+1} + \gamma V(S_{t+1})$ is TD target. $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is TD error.
TD Backup (Prediction):

$$V(S_t)\mathrel{+}= \alpha\left(R_{t+1} + \gamma V\big(S_{t+1} - V(S_t)\big)\right)$$

Bootstrapping: update involves an estimate. DP and TD use bootstrapping, not for MC.
Sampling: update samples an expectation. MC and TD samples, not for DP
TD can learn **before** knowing the final outcome, TD can learn **without** final outcome.
MC has high variance, zero bias (good convergence properties, not sensitive to initial value, simple to use).
TD has low variance, some bias (more efficient than MC, converges to $v_\pi(s)$, more sensitive to initial value).
TD Control: Prediction + ϵ-greedy: Apply TD to Q(S,A), use ϵ-greedy policy improvement, update every time-step.
SARSA Prediction: $Q(S,A)\mathrel{+}= \alpha\big(R + \gamma Q(S',A') - Q(S,A)\big)$
Q-Learning: $Q(S,A)\mathrel{+}= \alpha\big(R + \gamma maxQ(S',a') - Q(S,A)\big)$
T/F
Bandit Problem / Greedy / ϵ-greedy / SoftMax:
Incremental implementation is efficient and memory-saving. ϵ-greedy and SoftMax balance the e-e dilemma. (2019-F)
Finite MDP / MDP:
The policy iteration process must converge to an optimal policy and value in a finite number of iterations. (2022-S)
MDP is **not** a mathematical formalization of an agent. (2019-F)
The optimal value functions for states and state-action pairs are unique for a given MDP, there can be many optimal policies.
State-value Function:
For an episode $S_0, A_0, R_1, S_1, A_1, R_2, S_2 \ldots, \sum_{k=0}^{\infty}\gamma^k R_{k+1}$ is an unbiased estimator of $v_\pi(S_0)$. (2019-F)
Value iterations / Policy Iterations:
Let π_k and π_{k+1} be the policies that are *greedy* with respect to v_k and v_{k+1}, $\pi_{k+1} > \pi_k$ is **not** always true. (2022-S) (2019-F)
Policy and Value iterations are **not** clear of which is better.
Drawback of policy iteration is that each of its iteration involves policy evaluation, which may be an iterative process which requires multiple sweeps through the state space and performance. (2019-F)
GPI: GPI is a term used to refer to the general idea of letting policy-evaluation and policy improvement processes interact, independent of the granularity and other details of the two processes. (2022-S)
SARSA / Q-Learning:
In order to converge SARSA to optimal policy, all state-actions pairs need to be visited an infinite number of times **and** the policy converges in the limit to the greedy policy. (2019-F)
Using ϵ-greedy with $\epsilon = 0.1$, Q-Learning is **not** always achieving higher total rewards per episodes than SARSA. (2019-F)
SA
Discount Factor:
Purpose of discount factor in infinite horizon problem: Give a higher value to plans that reach a reward sooner; bounds the utility of a state. (2019-F)
Small discount factor have effect: discount future rewards more heavily, which means it prefer immediate rewards. (2019-F)
Non-stationary:
Why usually use constant step-size in incremental update as the real world most tasks are non-stationary which has random time-step: For a non-stationary environment, we would better weight recent reward more heavily than long-term rewards.
ϵ-greedy:
当一系列动作中有 reward 是不同于其他的，则此次行动是 exploration，第一次不算。 (2022-S)
MDP Markov Property:
MDP not suit for goal-directed task, since the Markov property, example: poke, GO. (2022-S)
Policy Evaluation / Value Iteration (IMPORTANT):
For iterative policy evaluation: The value function $v(s) = R(s) + \gamma\sum\big(P_{ss'}V(s)\big)$, the value $v_{k+1} = R_\pi + \gamma P_\pi v_k$.(2019-F)
For iterative value evaluation: **By using Bellman optimality backup**, the value function $v(s) = \max\big(R(s) + \gamma V(s)\big)$, the value $v_{k+1} = \max\big(R_\pi + \gamma v_{k(des)}\big)$. $v_{k(des)}$即目的地的上一次 value.(2019-F)
Bellman optimality equation: $v_1^* = \gamma max(v_1*, v_2^*), v_2^* = \gamma max(v_1*, v_2^*)$, max 中的 v_1 和 v_2 取迭代极限。(2019-F)
MC / TD:
One-step TD error: $\delta_t = R_{t+1+\gamma v(S_t+1)-v(S_t)}$. If true state-value function is used, $\mathbb{E}[\delta_t|S_t = s] = \mathbb{E}[R_{t+1} + \gamma v \ldots . |S_t = s] \to= v_\pi(s) - v_\pi(s) = 0$ (2022-S)(2019-F)
Why TD is bootstrapping method: TD method bases its update in part on existing estimates. (2019-F)
Why use action-value function in the GPI of MC and TD: because the model-free setting do not have state transition and reward function underlying MDP. (2019-F)
Rewards: If a constant 'c' is added to the rewards with maze running task, the effect to optimal policy: G_t will become $G_t + c * \left(\frac{1-\gamma^T}{1-\gamma}\right)$. (2022-S)
SARSA / Q-Learning:
Q-Learning is off-policy, since it always use greedy policy to update. (2019-F)
If greedy is used in SARSA rather than nearly greedy: The agent will get stuck assuming that some actions are worse than the current taking, and will not retry other actions, so it cannot learn. (2019-F)