T/F

Finite MDP / MDP

Reinforcement learning uses the **formal framework** of Markov decision processes (MDP) to define the interaction between a learning agent and its environment in terms of **states**.

MDP instances with **small** discount factors tend to emphasize (prefer) **near-term** rewards. (F19F)(F21M)

If the only difference between two MDPs is the value of discount factor γ, then they **may have the same or different** optimal policy. (F19F)(F21M)

For an infinite horizon MDP with a finite number of states and actions and with a discount factor $\gamma \in (0,1)$, value iteration **can guaranteed to converge**. (F19F)(F21M-S)

MDP is **not a mathematical formalisation** of an agent. (F19M)

Assume MRP model is known, then the bellman equation for MRP **can be solved** in a closed matrix form. However, direct solution is **only possible** for small MRPs. (F19M-S)

ϵ-greedy, SoftMax, UCB

Incremental implementation is **efficient and memory-saving**.

Compared to ϵ-greedy and SoftMax, UCB does not explore by sampling but rather inflates 夸大 the estimated expected **payout** according to its **uncertainty**. (F19F)

Policy Iterations / Value Iterations

Policy and value iterations are **not clear** of which is better.

GPI

Generalized policy iteration (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.

Monte Carlo

Typically, each iteration of a basic version of Monte Carlo Tree Search (MCTS) consists of four steps: **selection, expansion, simulation and backup**. (F19F)

In the case each return is an independent, identically distributed estimate of the value function v(s) with finite variance, the convergence of first-visit Monte Carlo method to the true value v(s) as the number of visits of each s goes to infinity is guaranteed by following the law of large numbers. (F19M-S)

In general, Monte Carlo method is **better on the existing data** (e.g. rewards collected from the episodes) while TD methods produce lower error on future data. (F19M-S)

λ return, TD(λ)

The on-line λ return algorithm is **not equivalent** to the on-line TD(λ) in terms of the total update to a value function at the end of an episode. (F19F)

SARSA / Q-Learning

Expected SARSA **may not has higher variance** in its updates than SARSA. (F19M-S)

Deep Q-Network (QDN)

In deep Q-Network, experience replay **randomises** over the data, thereby **removing correlations** in the observation sequence and **smoothing** over changes in the data distribution.

SA

Discount Factor

Q: For a gird world, using reward formulation to receive reward +1 on reaching the goal state and 0 for others. If two variants of this reward formulation: (P1) use discounted returns with $\gamma \in (0,1)$. (P2) no discounting is used. As a conclusion, a good policy can be learnt from (P1) but not from (P2), why? (F19F)

A: In (P2), since no discounting, the return for each episode regardless of the number of steps is +1. This **prevents the agent from learning a policy which tries to minimise the number of steps to reach the goal state.** In (P1), **the discount factor ensures that the longer the agent takes to reach the goal**, the lesser reward it gets. This motivates the agent to find the shortest path to the goal state.

Q: *Purpose of discount factor in infinite horizon problem:* (F19M)

A: Give a higher value to plans that **reach a reward sooner**; bounds the utility of a state.

Q: *Small discount factor have effect:* (F19M)

A: Discount future rewards more heavily, which means it **prefer immediate rewards**.

Q-Value Function

Q: A robot we cannot access in the real world, but can access the simulation software. We know that the software is built using the transition model $P_{sim}(s,a,s')$ that is different than the transition model $P_{real}(s,a,s')$ used in the robot. Select the new update rule for the Q-Value functions for the real world robot: (F19F)

(a) $Q(s,a) \leftarrow Q(s,a) + \alpha\big(P_{sim}(s,a,s')[r + \gamma max_{\alpha'} Q(s',a')] - Q(s,a)\big)$

(b) $Q(s,a) \leftarrow Q(s,a) + \alpha\left(\frac{P_{real}(s,a,s\prime)}{P_{sim}(s,a,s\prime)} \dots\right)$

(c) $Q(s,a) \leftarrow Q(s,a) + \alpha\left(\frac{P_{sim}(s,a,s\prime)}{P_{real}(s,a,s\prime)} \dots\right)$

A: (b) is correct. This is the importance sampling idea in off-policy learning. The Q-value function should be weighted by $\frac{P_{real}(s,a,s\prime)}{P_{sim}(s,a,s\prime)}$, so that they are correct in expectation instead of sampling from the correct distribution directly.

Watkins's Q(λ) algorithm

Q: What is the disadvantage of Watkins's Q(λ) algorithm? (F19F)

A: The disadvantage is the **early in learning**, the eligibility trace will be "cut" (zero out) frequently in little advantage to traces.

Policy Gradient

Q: In solving a multi-arm bandit problem using the policy gradient method, are we assured of converging to the optimal solution? (F19F)

A: No. Depending upon the properties of the function whose gradient is being ascended, the policy gradient approach **may converge to a local optimum.**

Q: Compare with the stochastic policy gradient, the deterministic policy gradient (DPG) can be estimated much more efficiency and perform better in a high-dimensional task, i.e., much lower computation cost. Why? (F19F)

A: DPG is the expected gradient of action-value function which **integrates over the state space**, while the stochastic policy gradient **integrates over the state and action space**.

A3C

Q: Benefits of using multiple agents in an asynchronous manner in A3C. (F19F)

A:

(1) It helps to stabilise the training. Since each agent has its own copy of the environment, agents are allowed to explore different parts of the environment as well as to use different policies at the same time. Therefore, when agents update the global parameters with their local parameters in an asynchronous manner, the global parameters update will be less correlated than using a single agent.

(2) The nature of multi-threads in A3C indicates that A3C needs much less memory to store experience, which means that no need to store the samples for experience replay as that used in DQN.

(3) The practical advantages of A3C is that it allows training on a multi-core CPU rather than GPU. When applied to a variety of Atari games, for instance, agents achieve a better result with asynchronous methods, while using far less resource than these needed on GPU.

ϵ-greedy:

当一系列动作中有 reward 是小的，则此次行动是 exploration，第一次不算。 (F22M-S)

Policy Evaluation / Value Iteration (IMPORTANT):

For iterative policy evaluation: The value function $v(s) = R(s) + \gamma\sum\big(P_{ss\prime}V(s)\big)$*, the value* $v_{k+1} = R_\pi + \gamma P_\pi v_k$.(F19F)

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.*(F19F)

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。(F19F)

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, $\mathbb{E}[\delta_t|S_t = s] = \mathbb{E}[R_{t+1} + \gamma v. \dots |S_t = s] \rightarrow = v_\pi(s) - v_\pi(s) = 0$ (F22M-S)(F19F)

Why TD is bootstrapping method: TD method bases its update in part on existing estimates. (F19F)

Rewards:

If a constant 'c' is added to the rewards with maze running task, the effect to optimal policy: G_t will become $G_t + c * \left(\frac{1-\gamma^T}{1-\gamma}\right)$. (F22M-S)

SARSA / Q-Learning

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. (F19F)