Reinforcement Learning

Reinforcement Learning

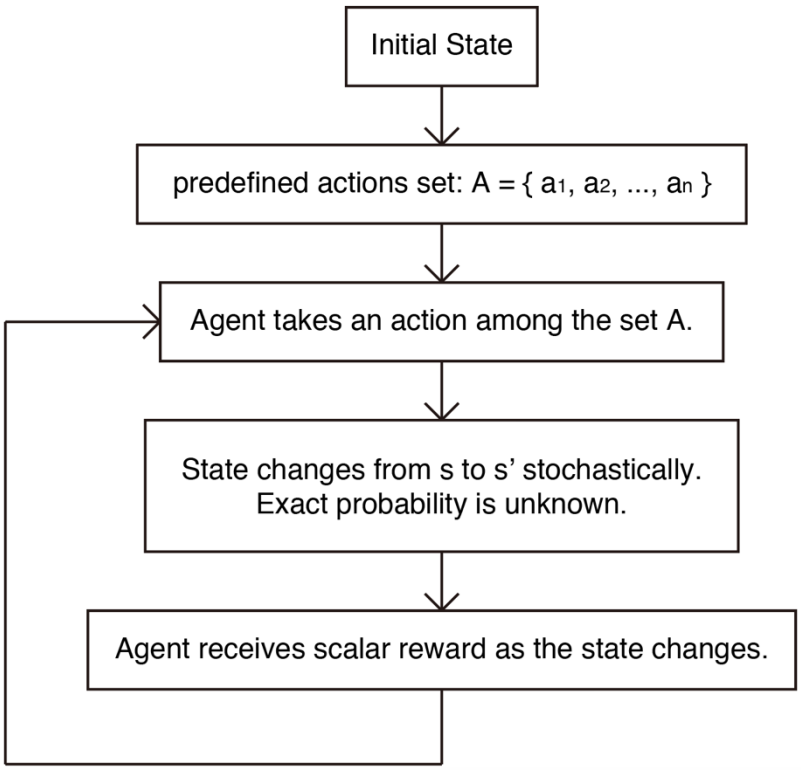
- Not supervised. No notion of ground truth and accuracy.
- Not unsupervised. Receives feedback in the form of reward.
- Used in stochastic environments and when sequential decision making is n eeded
  - Examples: grid world, bipedal robot walking.

Components of RL

- Agent
  - Lives inside the given environment and observes and interacts with it.
  - Makes decisions based on the current state.
  - Receives rewards based on its action.
  - Decision making (action) is stochastic.
- Environment
  - Produces the next state and the reward based on the agent's action.
  - The state transition and reward function are unknown to the agent.
  - Modeled as Markov decision process (MDP)
- State
  - Minimum information required to represent the environment.
  - Possible to reconstruct environment from state.
  - The state changes after agent's action.
  - State transition is stochastic (may not always land on the desired state).
- Reward

- Scalar measure of how well the agent is performing.
- Positive reward is given to the agent if the current action is good.
- Negative reward is given to the agent if the current action is bad.

Learning process



- Goal of RL: learning a state transition function π, called policy.
  - $\pi(s)$: Deterministic policy. The agent performs an action in state 's' deterministically and not probabilistically.
  - $\pi(s, a)$: Stochastic policy. The agent performs the action 'a' stochastically in state 's'. P(action = a | state = s).

Performance measure

- No accuracy => use cumulative reward instead to measure performance.
- Cumulative reward
  - $R_t = r_{t+1} + r_{t+2} + \cdots + r_{t+T}$ where 't' is timestamp.
  - Problematic as the agent may stick to a suboptimal action although the reward is small if T is infinite or indeterminate.
- Discounted cumulative reward
  - $R_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-1} r_{(t+T)} + \cdots = \sum_k \gamma^k \ r_{t+k+1}$

    where γ is a discount factor and $0 < \gamma < 1$.
  - Weigh current reward more than future rewards.
  - R_t is upper bounded: $R_t \leq \frac{1}{1-\gamma} R_{max}$
- π should choose actions such that $\max_{\theta} E[\sum_{k=0}^{\infty} \gamma^k r_{k+1}; \theta]$.
  - Such that maximizes the expected value of the discounted cumulative reward, with regard to θ, the parameter of π.
  - Expectation takes randomness into account.

Markov Decision Process

- MDP = <S, A, T, R, γ>
  - S: set of states
  - A: set of actions
  - T: transition function
    - T(s, a, s'): probability of the next state being s' when the agent takes action a in state s.
  - R: reward function
    - R(s, a): reward for performing action a in state s.
  - γ: discount factor
- The underlying MDP dynamics is unknown to the agent.
- Solving MDP itself is not RL.

Reinforcement Learning in MDP

- Model-based
  - Initially the underlying MDP is unknown to the agent.
  - The agent gradually estimates the underlying MDP in the learning phase.
  - Solve MDP using known algorithms, such as value iteration, to get the optimal policy.
- Model-free
  - The agent does not bother to learn the underlying MDP directly.
  - Update the policy only by interacting with the environment.

Value Function

- A way to measure the value of a state.
  - To let the agent to think about: is the state worth going?
- State-value function

$$V^{\pi}(s) = E_{\pi}[R_t|S_t = s]$$

From the state 's' and keep taking subsequent actions based on the policy π, what would be the expected cumulative reward?

- Action-value function
  - From the state-value function, consider action as well.

$$Q^{\pi}(s, a) = E_{\pi}[R_t|S_t = s, a_t = a]$$

From taking the action 'a' in state 's' and keep taking subsequent actions based on the policy π, what would be the expected cumulative reward?

- Optimal value function

$$V^{*}(s) = \max_{\pi} V^{\pi}(s)$$

$$Q^{*}(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

What would be the maximum of the value function with regard to the policy π?

Condition (constraint) of the optimal policy.

- Optimal value function expansion (Bellman optimality equation)

$$V^{*}(s) = \max_{a} \sum_{s'} T(s, a, s')(R(s, a) + \gamma V^{*}(s'))$$

Recursive definition => Dynamic programming can be used to solve it.

Or,

Change the equality to substitution (update) to compute the optimal policy iteratively.

Value Iteration Algorithm (Solving MDP in Model-based RL)

$$V_{k+1}(s) <= \max_a \sum_{s'} T(s,a,s')(R(s,a) + \gamma V(s'))$$

Bellman optimality equation in substitution form.

Value Iteration Algorithm:

while $\Delta < \theta$ $(\theta = error\ tolerance)$ # loop until the error Δ becomes sufficiently small

for all states in S:

$v = V(s)$ # temporarily store current V(s)

$V(s) = \max_a \sum_{s'} T(s,a,s')(R(s,a) + \gamma V(s'))$ # update V(s)

Δ = max(Δ, $|v - V(s)|$) # compare old and new value functions

Output policy $\pi(s) = argmax_a \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V(s'))$

- Value iteration itself is not reinforcement learning.
  - Expanded Bellman optimality equation involves T and R functions. The agent is learning the underlying MDP (model-based).
  - Value iteration assumes that T and R functions are known, which is not true for RL, thus value iteration itself is not RL.
  - Model estimation step should be combined to become RL.

Model-free RL

$$V^*(s) = \max_{\pi} V^{\pi}(s) = \max_{a} E[r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a]$$

Unexpanded (not expanding expectation) version of Bellman optimality equation.

Does not involve T and R function => model-free.

$V(s) <= V(s) + \alpha[r + \gamma V(s') - V(s)]$ where α is learning rate.

Rearranged Bellman optimality equation using Monte-Carlo approximation.

$r + \gamma V(s')$: Target value function (the one to be approximated)

$V(s)$: Current value function

As $V(s)$ approaches the target value, the estimation error approaches 0.

- Downside of using state-value function V(s): the algorithm requires π, which is the main goal to find. => Use action-value function Q(s,a) instead (Q-Learning and SARSA algorithms).

Q-Learning

Q-Learning Algorithm:

Initialize Q(s, a) arbitrarily

Repeat for each episode:

Initialize state s

while s is the terminal state:

Choose action a based on Q (ε-greedy)

Take action a and observe the corresponding reward r and the next state s'

$Q(s,a) <= Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ # update Q

$s <= s'$ # go to the next state

SARSA

SARSA Algorithm:

Initialize Q(s, a) arbitrarily

Repeat for each episode:

Initialize state s

Choose action a based on Q (ε-greedy)

while s is the terminal state:

Take action a and observe the corresponding reward r and the next state s'

Choose action a' based on Q

$Q(s,a) <= Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$ # update Q (ε-greedy)

$s <= s'$ # go to the next state

Model-free RL Algorithm Comparison

Q-learning	**SARSA**
Off-policy	On-policy
Use two different policies. Sampling policy != Target policy	Use two same ε-greedy Q. Sampling policy = Target policy
More explorative in terms of sampling policy and more exploitative in terms of the target policy because only the sampling policy is ε-greedy, but the target policy is not.	More explorative in terms of the target policy because both policies are ε-greedy.
Tend to take optimal (shortest) solution	Tend to take safer solution
Both converges well to optimum	

Exploration vs Exploitation

Exploitation	**Exploration**
Settle with what we already have	Explore and try to do some more learning
Maybe optimal but possible to miss better opportunity	Possible to improve more, but may waste more time
big value of ε => Q generates more certain (greedy) action	small value of ε => Q generates more random action
Balancing is crucial in RL	