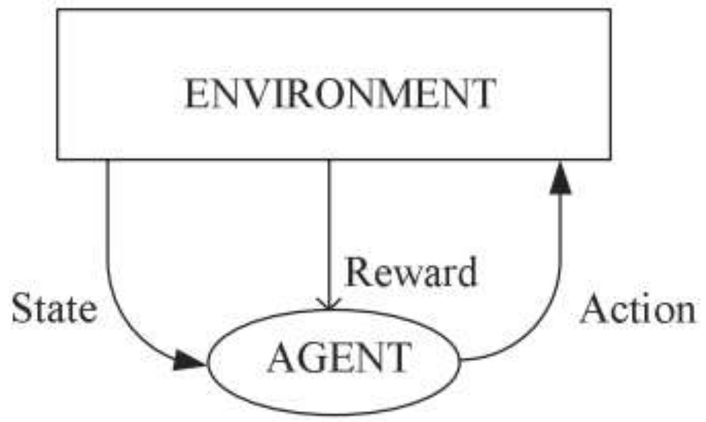
Reinforcement Learning

Prepared by
P.Subhashree

In reinforcement learning, the learner is a decision-making agent that takes actions in an environment and receives a reward (or penalty) for their actions in trying to solve a problem. After a set of trial-and-error runs, it should learn the best policy, which is the sequence of actions that maximizes the total reward.



Prepared by

P.Subhashree

Single State Case: K-Armed Bandit

- The K-armed bandit is a hypothetical slot machine with K levers. The action is to choose and pull one of the levers, and we win a certain amount of money which is the reward associated with the lever (action).
- The task is to decide which lever to pull to maximize the reward. This is a classification problem where we choose one of K. If this were supervised learning, then the teacher would tell us the correct class, namely, the lever leading to maximum earning. In this case of reinforcement learning, we can only try different levers and keep track of the best.
- This is a simplified reinforcement learning problem because there is only one state or one slot machine, and we need only decide on the action. Another reason why this is simplified is that we immediately get a reward after a single action; the reward is not delayed, so we immediately see the value of our action.

- We can choose different actions and store Q(a) for all a. Whenever we want to exploit, we can choose the action with the maximum value, that is,

  choose a_* if $Q(a_*) = \max Q(a)$

- An online update can be defined as

$$Q_{t+1}(a) \leftarrow Q_t(a) + \eta[r_{t+1}(a) - Q_t(a)]$$

where rt+1(a) is the reward received after taking action a at time (t +1)st time.

Elements of Reinforcement Learning

- The learning decision maker is called the agent. The agent interacts with the environment that includes everything outside the agent. The agent has sensors to decide on its state in the environment and takes an action that modifies its state. When the agent takes an action, the environment provides a reward.
- Time is discrete as t = 0, 1, 2,..., and st ∈ S denotes the state of the agent at time t where S is the set of all possible states. at ∈ A(st) denotes the action that the agent takes at time t where A(st) is the set of possible actions in state st . When the agent in state st takes the action at , the clock ticks, reward rt+1 ∈ $\mathfrak{R}$ is received, and the agent moves to the next state, st+1. The problem is modeled using a Markov decision process (MDP).

- The policy, π, defines the agent's behavior and is a mapping from the policy states of the environment to actions: π : S→A. The policy defines the action to be taken in any state st: at = π(st). The value of a policy π, V π (st), is the expected cumulative reward that will be received while the agent follows the policy, starting from state st.
- In the finite-horizon or episodic model, the agent tries to maximize the expected reward for the next T steps:

$$V^{\pi}(s_t) = E[r_{t+1} + r_{t+2} + \cdots + r_{t+T}] = E\left[\sum_{i=1}^{T} r_{t+i}\right]$$

- Certain tasks are continuing, and there is no prior fixed limit to the episode. In the infinite-horizon model, there is no sequence limit, but infinite-horizon future rewards are discounted:

$$V^{\pi}(s_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots] = E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right]$$

where $0 \leq \gamma < 1$ is the discount rate to keep the return finite.

- For each policy π, there is a V π (st), and we want to find the optimal policy π∗ such that

$$V^*(s_t) = \max_\pi V^\pi(s_t), \forall s_t$$

- for example, in control, instead of working with the values of states, V (st), we prefer to work with the values of state action pairs, Q(st , at). V (st) denotes how good it is for the agent to be in state st , whereas Q(st , at) denotes how good it is to perform action at when in state st . We define Q∗(st , at) as the value, that is, the expected cumulative reward, of action at taken in state st and then obeying the optimal policy afterward

- The value of a state is equal to the value of the best possible action:

$$\begin{aligned}
V^*(s_t) &= \max_{a_t} Q^*(s_t, a_t) \\
&= \max_{a_t} E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right] \\
&= \max_{a_t} E\left[r_{t+1} + \gamma \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i+1}\right] \\
&= \max_{a_t} E\left[r_{t+1} + \gamma V^*(s_{t+1})\right] \\
V^*(s_t) &= \max_{a_t} \left(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) V^*(s_{t+1}) \right)
\end{aligned}$$

Model-Based Learning

- We start with model-based learning where we completely know the environment model parameters, p(rt+1|st , at) and P(st+1|st , at). In such a case, we do not need any exploration and can directly solve for the optimal value function and policy using dynamic programming.

- Once we have the optimal value function, the optimal policy is to choose the action that maximizes the value in the next state:

$$\pi^*(s_t) = \arg\max_{a_t}\left(E[r_{t+1}|s_t, a_t] + \gamma \sum_{s_{t+1}\in S} P(s_{t+1}|s_t, a_t)V^*(s_t + 1)\right)$$

1.Value iteration

- To find the optimal policy, we can use the optimal value function, and there is an iterative algorithm called value iteration that has been shown to converge to the correct V ∗ values.
- We say that the values converged if the maximum value difference between two iterations is less than a certain threshold δ:

$$\max_{s \in S} |V^{(l+1)}(s) - V^{(l)}(s)| < \delta$$

```
Initialize a policy $\pi'$ arbitrarily
Repeat
    $\pi \leftarrow \pi'$
    Compute the values using $\pi$ by
        solving the linear equations
            $V^{\pi}(s) = E[r|s,\pi(s)] + \gamma \sum_{s' \in S} P(s'|s,\pi(s)) V^{\pi}(s')$
    Improve the policy at each state
        $\pi'(s) \leftarrow \arg\max_a (E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s'))$
Until $\pi = \pi'$
```

2.Policy Iteration

- In policy iteration, we store and update the policy rather than doing this indirectly over the values. The idea is to start with a policy and improve it repeatedly until there is no change.

- The value function can be calculated by solving linearequations. We then check whether we can improve the policy by taking these into account. This step is guaranteed to improve the policy, and when no improvement is possible, the policy is guaranteed to be optimal.

- Each iteration of this algorithm takes O(|A||S|2 + |S|3) time that is more than that of value iteration, but policy iteration needs fewer iterations than value iteration.