**Reinforcement Learning**

Reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals.

**Introduction**

* Consider building a learning robot. The robot, or agent, has a set of sensors to observe the state of its environment, and a set of actions it can perform to alter this state.
* Its task is to learn a control strategy, or policy, for choosing actions that achieve its goals.
* The goals of the agent can be defined by a reward function that assigns a numerical value to each distinct action the agent may take from each distinct state.
* This reward function may be built into the robot, or known only to an external teacher who provides the reward value for each action performed by the robot.
* The task of the robot is to perform sequences of actions, observe their consequences, and learn a control policy.
* The control policy is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent.

Example:

* A mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn."
* The robot may have a goal of docking onto its battery charger whenever its battery level is low.
* The goal of docking to the battery charger can be captured by assigning a positive reward (Eg., +100) to state-action transitions that immediately result in a connection to the charger and a reward of zero to every other state-action transition.

Reinforcement Learning Problem

* An agent interacting with its environment. The agent exists in an environment described by some set of possible states S.
* Agent perform any of a set of possible actions A. Each time it performs an action a, in some state st, the agent receives a real-valued reward r, that indicates the immediate value of this state-action transition. This produces a sequence of states si, actions ai, and immediate rewards ri as shown in the figure.



* The agent's task is to learn a control policy, 𝝅: S → A, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.

Reinforcement learning problem characteristics

1. Delayed reward: The task of the agent is to learn a target function 𝜋 that maps from the current state s to the optimal action a = 𝜋 (s). In reinforcement learning, training information is not available in (s, 𝜋(s)). Instead, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions. The agent, therefore, faces the problem of temporal credit assignment: determining which of the actions in its sequence are to be credited with producing the eventual rewards.

2. Exploration: In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses. This raises the question of which experimentation strategy produces most effective learning. The learner faces a trade-off in choosing whether to favor exploration of unknown states and actions, or exploitation of states and actions that it has already learned will yield high reward.

3. Partially observable states: The agent's sensors can perceive the entire state of the environment at each time step, in many practical situations sensors provide only partial information. In such cases, the agent needs to consider its previous observations together with its current sensor data when choosing actions, and the best policy may be one that chooses actions specifically to improve the observability of the environment.

4. Life-long learning: Robot requires to learn several related tasks within the same environment, using the same sensors. For example, a mobile robot may need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pick up output from laser printers. This setting raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks.

**Learning Task**

* Consider Markov decision process (MDP) where the agent can perceive a set S of distinct states of its environment and has a set A of actions that it can perform.
* At each discrete time step t, the agent senses the current state st, chooses a current action at, and performs it.
* The environment responds by giving the agent a reward rt = r(st, at) and by producing the succeeding state st+l = δ(st, at). Here the functions δ(st, at) and r(st, at) depend only on the current state and action, and not on earlier states or actions.
* The task of the agent is to learn a policy, 𝝅: S → A, for selecting its next action a, based on the current observed state st; that is, (st) = at.

How shall we specify precisely which policy π we would like the agent to learn?

1. One approach is to require the policy that produces the greatest possible cumulative reward for the robot over time.

* To state this requirement more precisely, define the cumulative value Vπ (st) achieved by following an arbitrary policy π from an arbitrary initial state st as follows:



Where, the sequence of rewards rt+i is generated by beginning at state st and by repeatedly using the policy π to select actions.

* Here 0 ≤ γ ≤ 1 is a constant that determines the relative value of delayed versus immediate rewards. if we set γ = 0, only the immediate reward is considered. As we set γ closer to 1, future rewards are given greater emphasis relative to the immediate reward.
* The quantity Vπ (st) is called the ***discounted cumulative reward*** achieved by policy π from initial state ***s***. It is reasonable to discount future rewards relative to immediate rewards because, in many cases, we prefer to obtain the reward sooner rather than later.

2. Other definitions of total reward is ***finite horizon reward,***



Considers the undiscounted sum of rewards over a finite number ***h*** of steps

3. Another approach is ***average reward***

******

Considers the average reward per time step over the entire lifetime of the agent.

We require that the agent learn a policy π that maximizes Vπ (st) for all states s. such a policy is called an ***optimal policy*** and denote it by π\*



Refer the value function Vπ\*(s) an optimal policy as V\*(s). V\*(s) gives the maximum discounted

cumulative reward that the agent can obtain starting from state ***s***.

***Q* LEARNING**

***An optimal policy π \* for an arbitrary environment:***

The training information available to the learner is the sequence of immediate rewards r(si,ai) for i = 0, 1,2, . . . . Given this kind of training information it is easier to learn a numerical evaluation function defined over states and actions, then implement the optimal policy in terms of this evaluation function.

***Evaluation function the agent should attempt to learn:***

One obvious choice is V\*. The agent should prefer state sl over state s2 whenever V\*(sl) > V\*(s2), because the cumulative future reward will be greater from sl

The optimal action in state ***s*** is the action ***a*** that maximizes the sum of the immediate reward r(s, a) plus the value V\* of the immediate successor state, discounted by γ.



**--------(1)**

**The Q Function**

The value of Evaluation function ***Q(s, a)*** is the reward received immediately upon executing action a from state ***s***, plus the value (discounted by ***γ*** ) of following the optimal policy thereafter



--------(2)

Therefore, we can rewrite Equation (1) in terms of *Q(s, a)* as



This is important because it shows that if the agent learns the *Q* function instead of the *V\** function, it will be able to select optimal actions *even when it has no knowledge of the functions r and δ.*

AS the Equation makes clear, it need only consider each available action ***a*** in its current state ***s*** and choose the action that maximizes ***Q(s, a)***.*.*

**An Algorithm for Learning *Q***

* Learning the Q function corresponds to learning the **optimal policy**.
* The key problem is finding a reliable way to estimate training values for ***Q***, given only a sequence of immediate rewards ***r*** spread out over time. This can be accomplished through ***iterative approximation***



Rewriting the equations as:





Q learning algorithm assuming deterministic rewards and actions. The discount factor γ may be any constant such that 0 ≤ γ < 1

𝑄̂ to refer to the learner's estimate, or hypothesis, of the actual Q function

**An Illustrative Example**

* To illustrate the operation of the Q learning algorithm, consider a single action taken by an agent, and the corresponding refinement to 𝑄̂ shown in below figure

 The agent moves one cell to the right in its grid world and receives an immediate reward of zero for this transition.

* Apply the training rule of Equation



to refine its estimate Q for the state-action transition it just executed.

* According to the training rule, the new 𝑄̂ estimate for this transition is the sum of the received reward (zero) and the highest 𝑄̂ value associated with the resulting state (100), discounted by γ (.9).

