UNIT-V

Reinforcement Learning

Introduction:

consider the robot or an agent, it has a set of sensors to observe the state of its environment, and a set of actions it can perform to alter this state.

Ex:- Mobile robot may have sensors such as camera and sonars, and actions such as "move forward" and "turn".

Its task is to learn a control strategy, or policy for choosing actions that achieve its goals.

for example, the robot may have a goal of docking onto its battery charger whenever its battery level is low.

RL (Reinforcement Learning) is concerned with how such agents can learn successful control policies by experimenting in their environment.

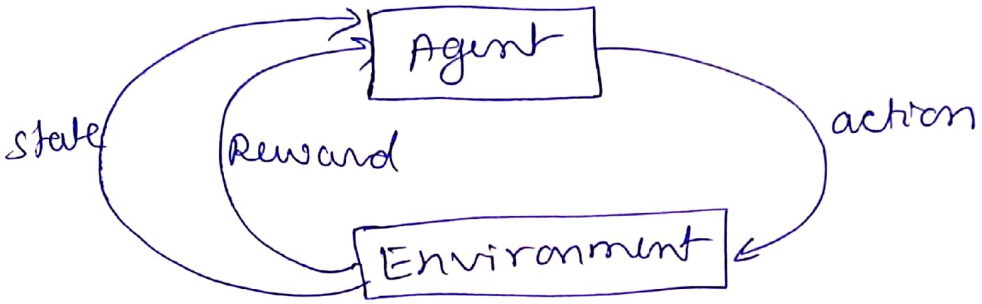
Goal of the agent can be defined by a reward function that assigns a numerical value to the desired agent for its action on specific state

A desired control policy is the one that from any initial state, chooses actions that maximize the reward accumulated over time by the agent.

The task of an agent is to perform sequence of actions, observe their consequences & learn a control policy.

In general, the focus is on building any type of agent that must learn to choose actions that alter the state of its environment and where a cumulative reward function is used to define the quality of any given action sequence.

A general setting of RL is given as following



Five major components of RL

① Agent : Is the learning program.

② Environment : Everything with which the agent interacts

③ State : A specific situation in which agent finds itself.

④ Action : All possible moves the agent can make

⑤ Reward : Feedback from environment based on agent's action.

As there is no historical data available, RL initiates its learning based on trial and error approach and tries to improve its action sequence over the period of learning based on rewards.

RL problem differs from other function approximation tasks in several respects as follows:

① Delayed reward

The agent 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

The agent influences the distribution of training examples by the action sequence it chooses. Agent faces the problem of trade off between choice of exploration and exploitation method

③ Partially observable states

In many practical situations, sensors may provide only partial information. ~~it~~ Some times the state of environment may be partially observed because of unforeseen cases.

④ Life-long learning

Agents need to learn continuously for longer period until it come across all possible States of an environment. In some cases, it can be a learning for unlimited period

Learning Task

In a Markov Decision Process (MDP) the agent can perceive a set 'S' of distinct states of its envt and has a set of actions 'A' that it can perform. At each discrete time step t, the agent senses the current state s_t, chooses a current action a_t, and performs it.

The Envt responds by giving the agent a reward $r_t = r(s_t, a_t)$ and by producing successful state $s_{t+1} = \delta(s_t, a_t)$

Here δ and r are the part of the envt and are not necessary to known to the agent.

In MDP the fun $\delta(s_t, a_t)$ and $r(s_t, a_t)$ depends only on the current state ~~and action~~, and not on earlier states or actions.

The task of agent is to learn a policy, ~~π~~ ii $\pi : S \rightarrow A$

The policy should accumulate greatest possible reward over the learning period. ~~and~~ The policy can be represented as follows

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots\dots$$
$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where the sequence of rewards r_{t+i} is generated by starting at state s_t and by repeatedly using the policy π to select actions. at

i.e $a_t = \pi(s_t)$, $a_{t+1} = \pi(s_{t+1})$, etc.

Here $0 \leq \gamma < 1$ is a constant that determines the relative value of delayed v/s immediate rewards.

It is required that the agent learn a policy π that maximizes $V^\pi(s)$ for all states s, and such policy is called as optimal policy and is denoted by π^*. & represented as follows

$$\pi^* \equiv \underset{\pi}{\text{argmax}}\ V^\pi(s), (\forall s).$$

Learning :-

How can an agent learn an optimal policy π^* for an arbitrary environment?

It is difficult to learn the fun $\pi^* : S \to A$ directly, as the available training data does not provide training examples of the form $\langle s, a \rangle$.

Instead it will have only the sequence of rewards $r(s_i, a_i)$ $\forall$ $i = 0, 1, 2 \ldots$

The optimal action in state 's' is the action 'a' that maximizes the reward

$$\pi^*(s) = \underset{a}{\text{argmax}} \left[r(s,a) + \gamma V^*(\delta(s,a)) \right]$$

eqn ① — reward function — discounted factor

The Q Function :

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

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a)) \quad -(2)$$

ie $\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$ [Based on eq (1)]

An algorithm for Learning Q

Learning the Q fun 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 reward r spread over time. This can be achieved/accomplished through iterative approximation.

ie $V^*(s) = \max_{a'} Q(s,a')$

which allows rewriting the eq (2) as follows.

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a), a').$$

Here s is the current state of environment

a is the current action to be taken by an agent

$\delta(s,a)$ is the transition function that returns a next/changed state of environment

ie $\delta(s,a) = s'$ where s' is the new state.

r is the reward

γ is the discounted factor (0 to +1) or learning rate.

Q learning algorithm

① For each s, a initialize the table any $\hat{Q}(s,a)$ to zero

② Observe the current state s

③ Do forever:

(a) Select an action 'a' and execute it

(b) Receive immediate reward r.

(c) Update the table entry for $\hat{Q}(s,a)$ as follows:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a') \quad \text{—③}$$

(d) $s \leftarrow s'$

Algorithm convergence

The key idea underlying the proof of convergence is that table entry $\hat{Q}(s,a)$ with the largest error must have its error reduced by a factor of γ whenever it is updated. The reason is that its new value depends only in part on error-prone $\hat{Q}$ estimates, with the remainder depending on the error-free observed immediate reward r.

Theorem: convergence of Q learning for deterministic Markov decision processes.

Consider a Q learning agent in a deterministic MDP with bounded rewards $(\forall s,a)\ |r(s,a)| \le c$. The Q learning agent uses the training rule of eq ③, initializes its table $\hat{Q}(s,a)$ to arbitrary

finite values, and uses a discounted factor γ such that $0 \leq \gamma < 1$. Let $\hat{Q}_n(s,a)$ denote the agent's hypothesis $\hat{Q}(s,a)$ following the n^{th} update. If each state-action pair is visited infinitely often, then $\hat{Q}_n(s,a)$ converges to $Q(s,a)$ as $n \to \infty$, $\forall s, a$.

Proof: The proof shows that the maximum error over all entries in the $\hat{Q}$ table is reduced by at least a factor of γ during each such interval. $\hat{Q}_n$ is the agent's table of estimated Q values after n updates.

Let Δ_n be the max error in $\hat{Q}_n$;

$$\text{ie } \Delta_n \equiv \max_{s,a} |\hat{Q}_n(s,a) - Q(s,a)|$$

Let $\delta(s,a) = s'$

Then for any table entry $\hat{Q}_n(s,a)$ updated on iteration $n+1$, the magnitude of the error in the revised estimate $\hat{Q}_{n+1}(s,a)$ is

$$|\hat{Q}_{n+1}(s,a) - Q(s,a)| = |(r + \gamma \max_{a'} \hat{Q}_n(s',a')) - (r + \gamma \max_{a'} Q(s',a'))|$$

$$= \gamma |\max_{a'} \hat{Q}_n(s',a') - \max_{a'} Q(s',a')|$$

γ is dropped by considering it as const

$$\leq \gamma \max_{a'} \left| \hat{Q}_n(s', a') - Q(s', a') \right|$$

$$\leq \gamma \max_{s'', a'} \left| \hat{Q}_n(s'', a') - Q(s'', a') \right|$$

$$\boxed{\left| \hat{Q}_{n+1}(s, a) - Q(s, a) \right| \leq \gamma \Delta_n}$$

Thus, the updated $\hat{Q}_{n+1}(s, a)$ for any s, a is at most γ times the max err in $\hat{Q}_n$ table, Δ_n. The largest error in the initial table, Δ_0, is bounded because values of $\hat{Q}_0(s, a)$ and $Q(s, a)$ are bounded for all s, a.

Temporal Difference learning

Q learning algorithm learn by iteratively reducing the discrepancy between Q value estimates for adjacent states.

In this sense, Q learning is a special case of a general class of temporal difference algs that learn by reducing discrepancies between estimates made by the agent at diff times.

Let $Q^{(1)}(s_t, a_t)$ denote the training value calculated by this one-step lookahead.

$$Q^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$

where $\delta(s_t, a_t) = s_{t+1}$ (new state)

(s_t: current state; a_t: current action.)

Alternative way to compute $Q(s_t, a_t)$ is

$$Q^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a)$$

or in general for n steps

$$Q^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a)$$

$TD(\lambda)$ is a method introduced by Sutton (1988) for blending these alternative training estimates.

The idea is to use a constant $0 \leq \lambda \leq 1$ to combine the estimates obtained from various lookahead distance in the following way

$$Q^{\lambda}(s_t, a_t) \equiv (1-\lambda)\left[Q^{(1)}(s_t, a_t) + \lambda Q^{2}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \cdots \right]$$

An equivalent recursive definition for Q^{λ} is

$$Q^{\lambda}(s_t, a_t) = r_t + \gamma \left[(1+\lambda) \max_a \hat{Q}(s_t, a_t) + \lambda (Q^{\lambda}(s_{t+1}, a_{t+1}) \right]$$