

What is reinforcement learning?

Notes

The principles of reinforcement learning are simple

- An agent must solve a task and has a set of actions available
- The agent chooses the action or set of actions it believes can solve the task
- The agent receives feedback information from the environment about its performance
- The agent uses this information to modify its behaviour

Where reinforcement learning is applied?

Notes

- Reinforcement learning is used when an agent can interact with the environment
- The environment has to be able to quantify the success or failure of the agent actions
- The state space of the problem can be discrete or continuous
- Different learning goals can be used (assess the quality of a set of actions, find the optimal set of actions, ...) with different complexities (states or effects of the actions unknown, non deterministic effects of the actions, ...)

When reinforcement learning is used?

Notes

- It is used for problems too complex to be solved hand coded
- To generate a set of examples to learn the task by classical inductive methods is too costly
- It is more easy to allow the agent to learn the task by trial and error
- Applications: Robotics, computer games, sequencing of manufacturing processes, temporal sequencing, ...

Elements of reinforcement learning

Notes

- A set of states that describe the environment (discrete or continuous)
- A set of actions that the agent can perform to modify the environment
- A reinforcement function that informs to the agent of the result obtained from performing the action on the environment
- **Goal:** To find a policy that links states with actions and maximize the gains obtained from the reinforcements

Learning in finite environments

Notes

- The simplest reinforcement learning scenario is:
 - A finite set of states
 - States can be observed
 - The results of the actions are deterministic
- In this case the problem can be modeled as a markov decision process (MDP)

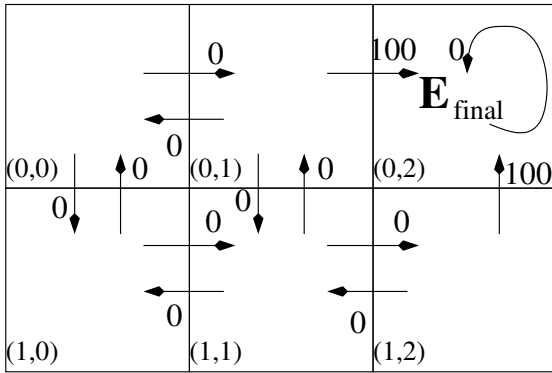
Markov decision processes

Notes

- An agent can perceive a set S of possible states
- There are a set A if actions that can be performed
- Each interval of time (discrete) the agent observes the current state (s_t) and picks an action to perform (a_t)
- The environment gives the agent a reward corresponding with the state and the action ($r_t = r(s_t, a_t)$) and changes to the next state ($s_{t+1} = \delta(s_t, a_t)$)
- r and δ are functions of the environment and are not necessarily known by the agent

Markov decision processes (r and δ)

Notes



Markov decision processes

Notes

- In a MDP the functions r and δ depend only on the current state and not on the past states
- We will suppose that these functions are deterministic
- The goal of the agent is to derive a function $\pi : S \rightarrow A$ that allows to pick the action a_t using the state s_t

Reward distribution

Notes

- The simpler approach is to require that the actions are picked so the largest accumulative reward is obtained over the time:

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

- Where r_{t+i} are the rewards generated by following the actions determined by π and $0 \leq \gamma \leq 1$ is a constant that determines the weight of the immediate rewards respect the future rewards
- There are other ways to distribute the weights of the rewards such as limiting the number of steps rewarded or obtaining the mean of the rewards

Choosing the optimal policy

Notes

- From this function we can establish the learning goal as the function π^* that determines the best action for all states (optimal policy):

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

- We will define $V^*(s)$ as the function that computes the largest accumulative reward that an agent could obtain from the state s if the policy determined by π^* is followed
- Now we only need an algorithm able to compute the function π^*

Q-Learning

Notes

- It is difficult to learn directly the function π^* because the only that we can observe are the rewards sequence that the agent receives over time $r(s_i, a_i)$ for $i = 0, 1, 2, \dots$
- When we know function r and function δ we could learn the function V^* that allows to compare states
- This way we could determine the best action as:

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

- Unfortunately in most of the problems r and δ are unknown

The function Q

Notes

- We define function $Q(s, a)$ as the function that computes the maximum accumulative reward from state s performing the action a as the first action
- This function is equivalent to:

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

- That allows to rewrite π^* as:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q(s, a)$$

- By learning function Q it is not necessary to know functions r and δ
- That means that a global policy can be derived from local observations

An algorithm for learning Q (1)

Notes

- The idea is to use the immediate rewards sequence r to derive the function Q
- The function V^* is related to Q in the following way:

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

- That allows to rewrite the definition of Q as:

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

- This recurrence allows to derive an iterative algorithm to compute the function Q

An algorithm for learning Q (2)

Notes

- Let be \hat{Q} an estimation of Q
- \hat{Q} is represented as a table indexed by all possible combination of states (s) and actions (a) that stores the estimated value of $\hat{Q}(s, a)$
- The agent will iteratively observe the current state s and pick an action a
- It will observe the reward obtained from the environment $r = r(s, a)$ and the new state $s' = \delta(s, a)$
- With this information the value of $\hat{Q}(s, a)$ in the table will be updated using the rule

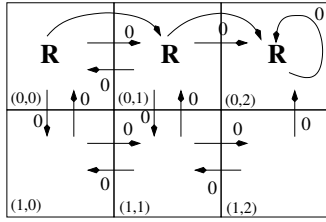
$$\hat{Q}(s, a) = r + \gamma \max_{a'} \hat{Q}(s', a')$$

- The agent does not need to know neither r , nor δ , it only has to observe the answer of the environment to its action

The training process

Notes

- To learn the agent has to be trained in the environment
- The idea is to let the agent play in the environment for a period of time (usually plenty) until the function \hat{Q} converges
- After certain number of steps \hat{Q} converges to Q if:
 - The problem can be modeled as as MDP
 - The reinforce is bounded by a value c
 - Each pair (state, action) is visited an infinite number of times

Example (1st Training)

- 1 $s = (0, 0)$, action= a_{right} , $s' = (0, 1)$, $r = 0$, $\gamma = 0.9$

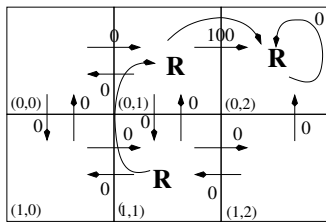
$$\hat{Q}((0, 0), a_{right}) = r((0, 0), a_d) + 0.9 \cdot \max\{\hat{Q}((0, 1), a_{right}),$$

$$\hat{Q}((0, 1), a_{left}), \hat{Q}((0, 1), a_{ab})\} = 0 + 0.9 \cdot \max\{0, 0, 0\} = 0$$

- 2 $s = (0, 1)$, action= a_{right} , $s' = (0, 2)$, $r = 100$, $\gamma = 0.9$

$$\hat{Q}((0, 1), a_{right}) = 100 + 0.9 \cdot \max\{0\} = 100$$

Notes

Example (2nd Training)

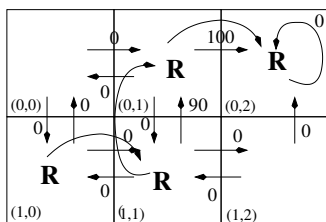
- 1 $s = (1, 1)$, action= a_{up} , $s' = (0, 1)$, $r = 0$, $\gamma = 0.9$

$$\hat{Q}((1, 1), a_{up}) = 0 + 0.9 \cdot \max\{0, 0, 100\} = 90$$

- 2 $s = (0, 1)$, action= a_{right} , $s' = (0, 2)$, $r = 100$, $\gamma = 0.9$

$$\hat{Q}((0, 1), a_{right}) = 100 + 0.9 \cdot \max\{0\} = 100$$

Notes

Example (3rd Training)

- 1 $s = (1, 0)$, action= a_{right} , $s' = (1, 1)$, $r = 0$, $\gamma = 0.9$

$$\hat{Q}((1, 0), a_{right}) = 0 + 0.9 \cdot \max\{0, 0, 90\} = 81$$

- 2 $s = (1, 1)$, action= a_{up} , $s' = (0, 1)$, $r = 0$, $\gamma = 0.9$

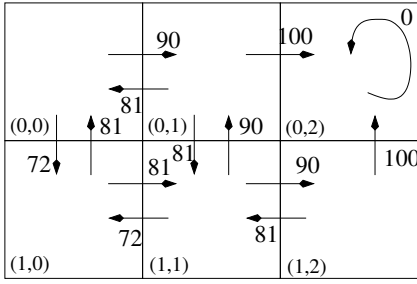
$$\hat{Q}((1, 1), a_{up}) = 0 + 0.9 \cdot \max\{0, 0, 100\} = 90$$

- 3 ...

Notes

Example - After convergence

Notes



Notes

Notes
