Two potential approaches.

① Find the value of state

② Find the value of action

Two RL schemes:

1. Policy iteration [monte carlo, Temporal differencing]

evaluate policy to improve it

emphasis on Sampling and sampling as much as we can

What is improving TD(λ) (what is now and next)

2. Value iteration [Q-learning]

maximize accumulate reward for (s,a)

Deal with rewards ?

MDP - markov decision process

MDP — keep everything (order n) $Q_K = \dfrac{r_1 + r_2 + \dots + r_k}{K}$

— Do it incrementally

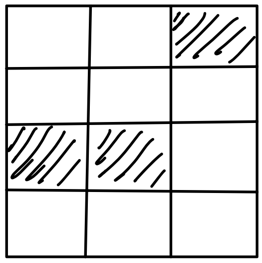
$$(\text{Avg})\, Q_{K+1} = (\text{Avg})\, Q_K + \frac{1}{K+1}\left[r_{K+1} - Q_K\right]$$

Common update rule:

New estimate = Old estimate + step size * [Target - Old estimate]

Caution: state-space problem

elevator system



$4 \times 4 \times 4 = 64 \leftarrow$ total states

2 2 4 $\longrightarrow$ eg: state 62

Design RL agents :

1. Discretize states
2. Define actions 
3. Determine the reward / punishment
4. Stablish the action policy (taking action)

How to take actions :

1. Random
2. Greedy (action with maximum reward)
3. Epsilon ε- greedy (max action with $p = 1 - \varepsilon$ and random with ε, $\varepsilon > 0$)

   At the begining ε is large

4. Softmax action selection

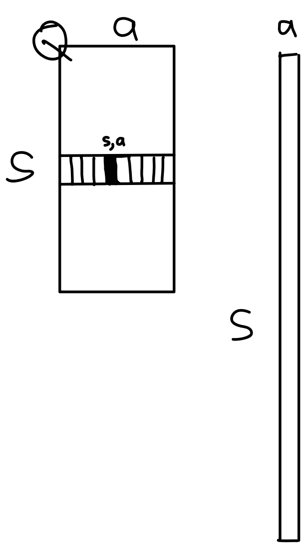
Softmax : We use Gibbs or Boltzmann Distribution

$$P(a) = \frac{\exp\left(\frac{Q(s,a)}{\tau}\right)}{\sum_{b=1}^{n} \left(\exp Q(s,b) / \tau\right)}$$

accumulated reward from the table

} for normalization

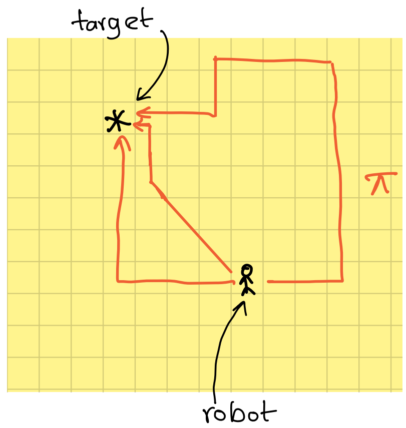
\ temperature (control)



(In reality)

The RL agent learns a policy π.

eg: Grid world



need ai when problem is stochastic, not deterministic

exploiting policy, explore

At step t, $\pi_t(s,a)$ is the probability that $a_t = a$ when $s_t = s$.

RL methods enable a change of policy based on experience.

Maximise reward :

1) Episodic task : Return at t, $R_t = r_{t+1} + r_{t+2} + \dots r_T$

Terminal

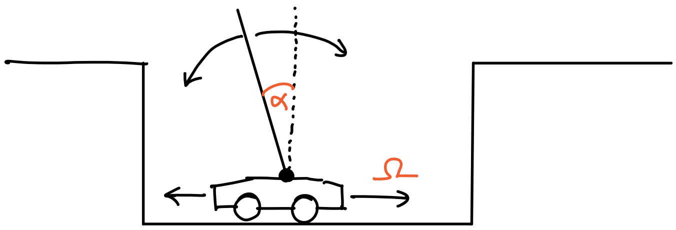
2) Continuous task : $R_t = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \dots$

$$= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$ γ - discount factor

0 —— γ —— 1

short sighted — far sighted

Inverted pendulum



Avoid failure

1) Poll falls
2) Car hit the wall

Understand as an episodic task.

$r = +1$ (for each step prior to fall)

R = number of steps before fall

Understand as a continuous task

$r = -1$ (for falling), 0 otherwise

$R = -\gamma^k$ for k steps before the fall

Temporal Differencing methods - TD

TD Prediction = Policy evaluation

⟶ Compute the state value function V^π for a given policy π

Policy evaluation

① Simple every visit Monte-Carlo

Return

$$V(s_t) \leftarrow V(s_t) + \alpha\left(R_t - V(s_t)\right)$$

value of the state

② The simplest TD $\longrightarrow$ TD(0) estimate of the return

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

Learning an action value function

estimate

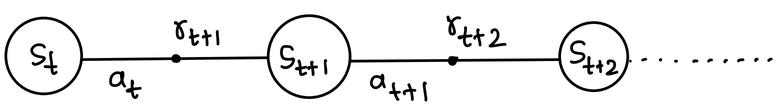
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

accumulated reward

reward

target

(reward together)



Q-Learning

design of the agent is difficult, learning algorithm is simple.

① Initialize $Q(s,a)$ (matrix) randomly.

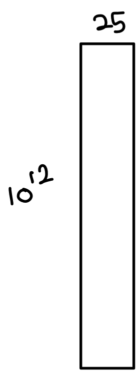
② for each episode,

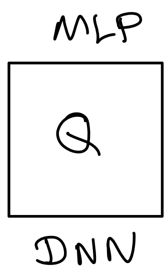
- intialize state s
- For each step
  - choose a from s using action policy
  - take action a

- observe r, $s' = s_{t+1}$
  - update $Q(s,a) \leftarrow Q(s,a) + \alpha \ldots$
  - update $s \leftarrow s'$
- until s terminal

Convergence

Every state has to be visited multiple times.





train NN to acumulate the reward

Deep RL $\longrightarrow$ Use DNN to hold/model Q-matrix