Reinforcement Learning
(Asynchronous Multiagent Q-learning)

agent no-coordinate

(move independently)

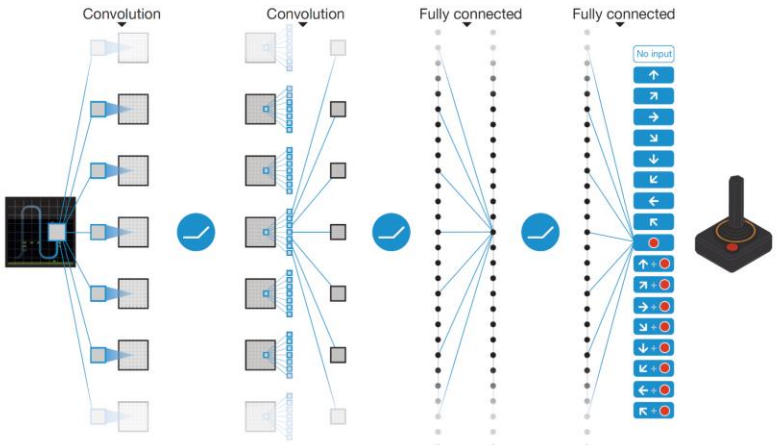
ECE 277

Cheolhong An

Deep Reinforcement learning: RL + Deep Learning

Learned to play 49 games for the Atari 2600 game console, without labels or human input, from self-play and the score alone

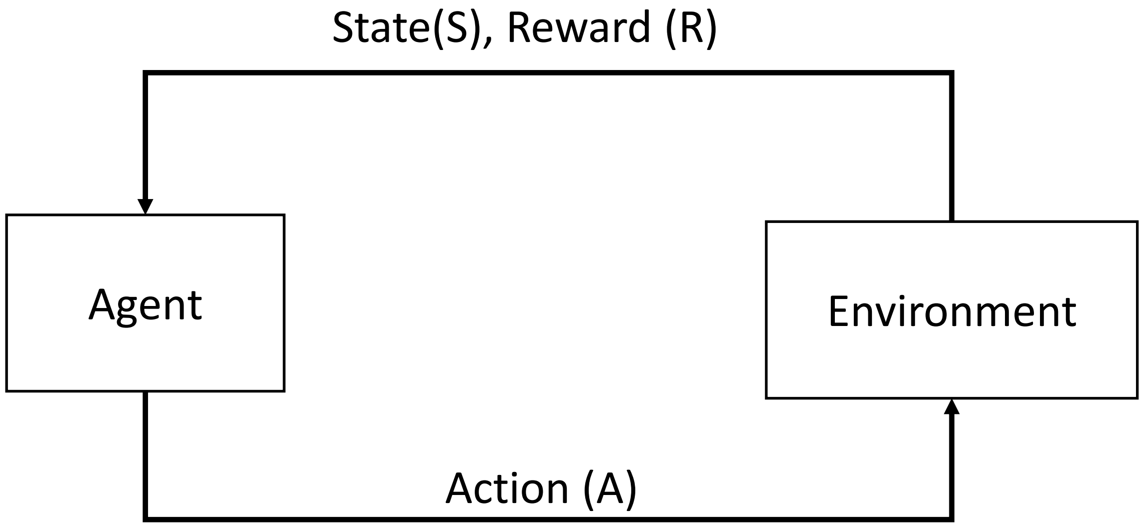
mapping raw screen pixels

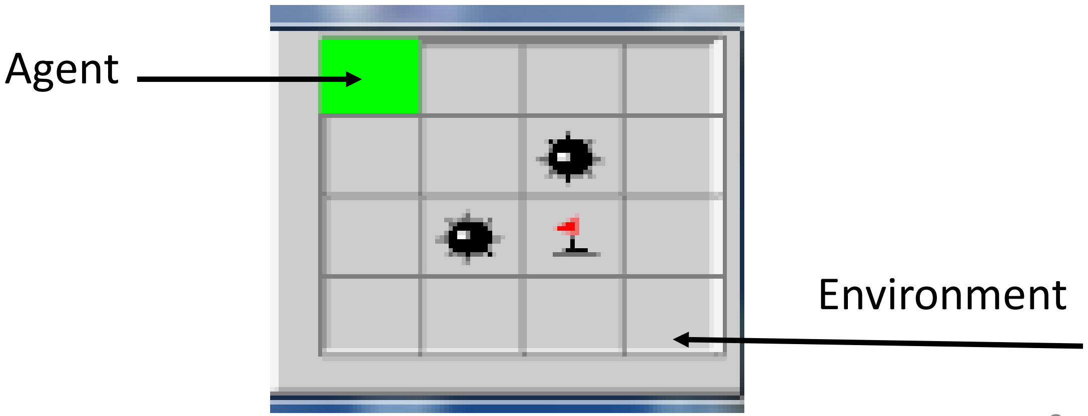


to predictions of final score for each of 18 joystick actions

Reinforcement learning: Q-learning (single agent)

- Goal
  - catch flag
- Environment
  - 4x4 grid world
  - two mines and one flag
- Status
  - (x,y) position of an agent
- Reward
  - catch flag: +1
  - step mine: -1
  - otherwise: 0
- Action
  - 0: right, 1: down , 2: left, 3: up
- Episode end
  - Agent reaches one of mines or a flag
  - Every episode restarts from (0,0)

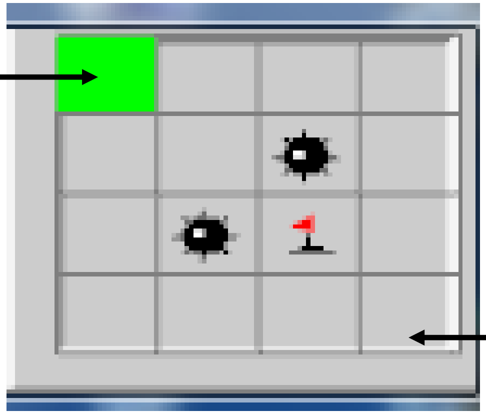




Multiagent RL

The number of agents = 1
Environment size = 4x4 (16)
The number of mines = 2
The number of a flag = 1

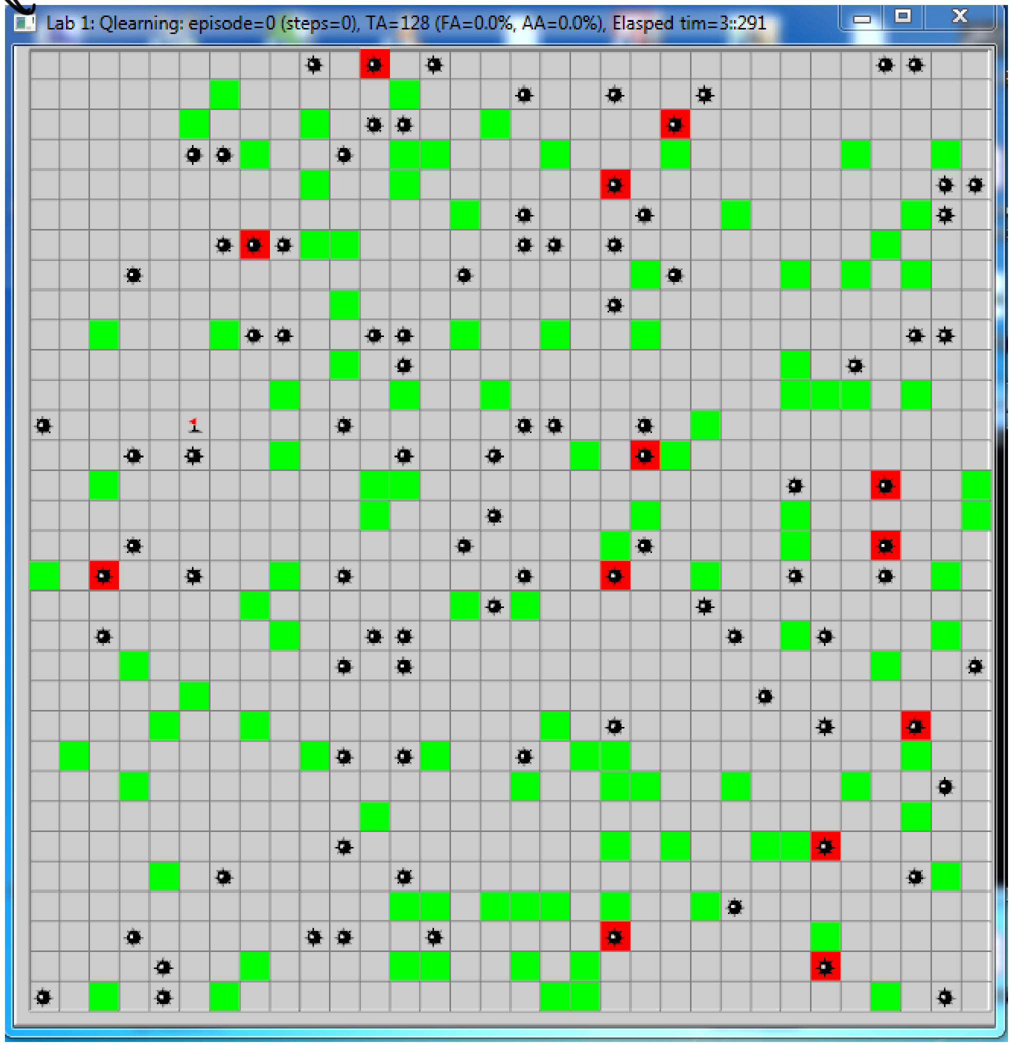
Agent



Environment

Invoke 할때마다 changing

The number of agents = 128
Environment size = 32x32 (1024)
The number of mines = 96
The number of a flag = 1



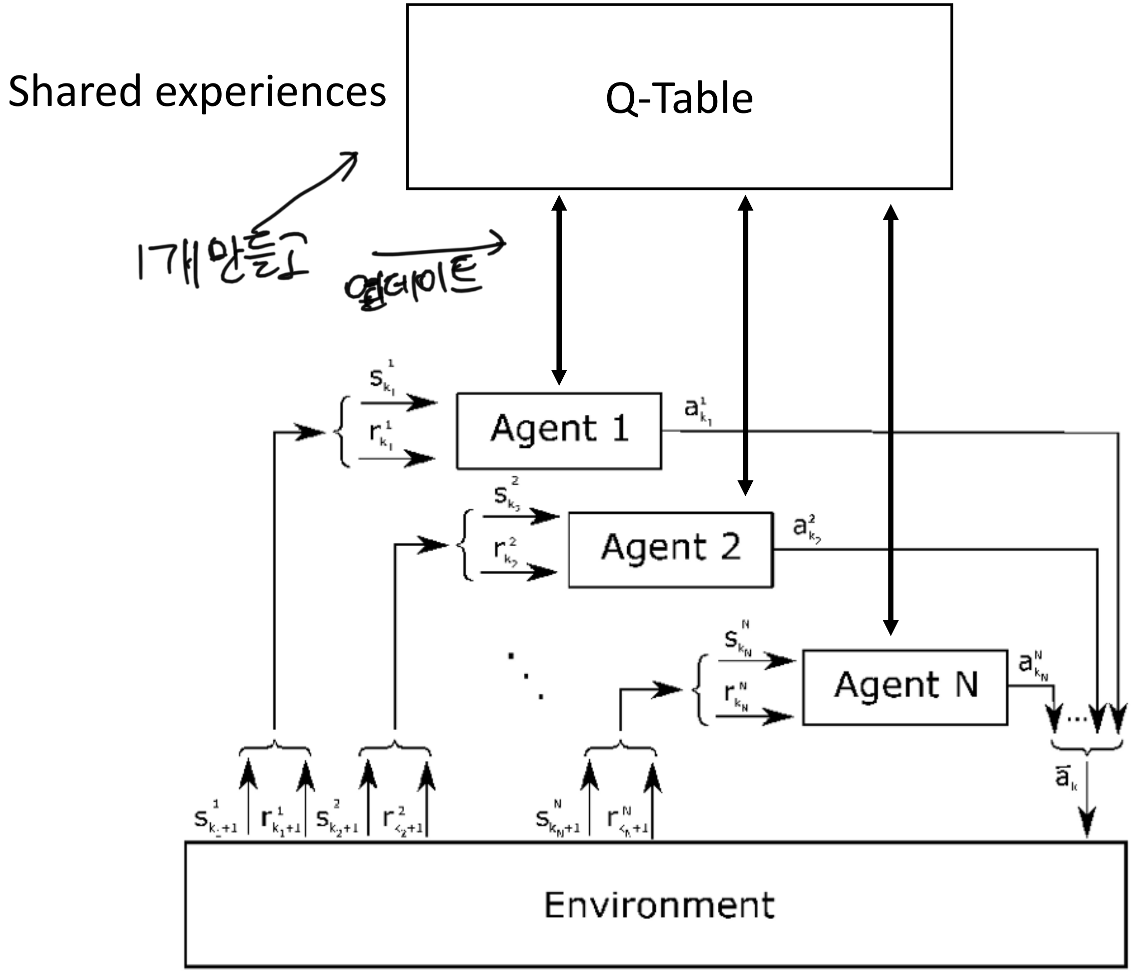
Asynchronous Multiagent RL environment

- S_n: current state
  env.d_state[m_sid]: *int2 for each agent
  *int2 cstate = env.d_state[m_sid];
  cstate[agent_id]
- S_{n+1}: next state
  env.d_state[m_sid^1]: *int2 for each agent

- Each state indicates of a position
  d_state[agent_id].x: x position
  d_state[agent_id].y: y position

- Reward:
  env.d_reward: *float for each agent
  d_reward[agent_id]

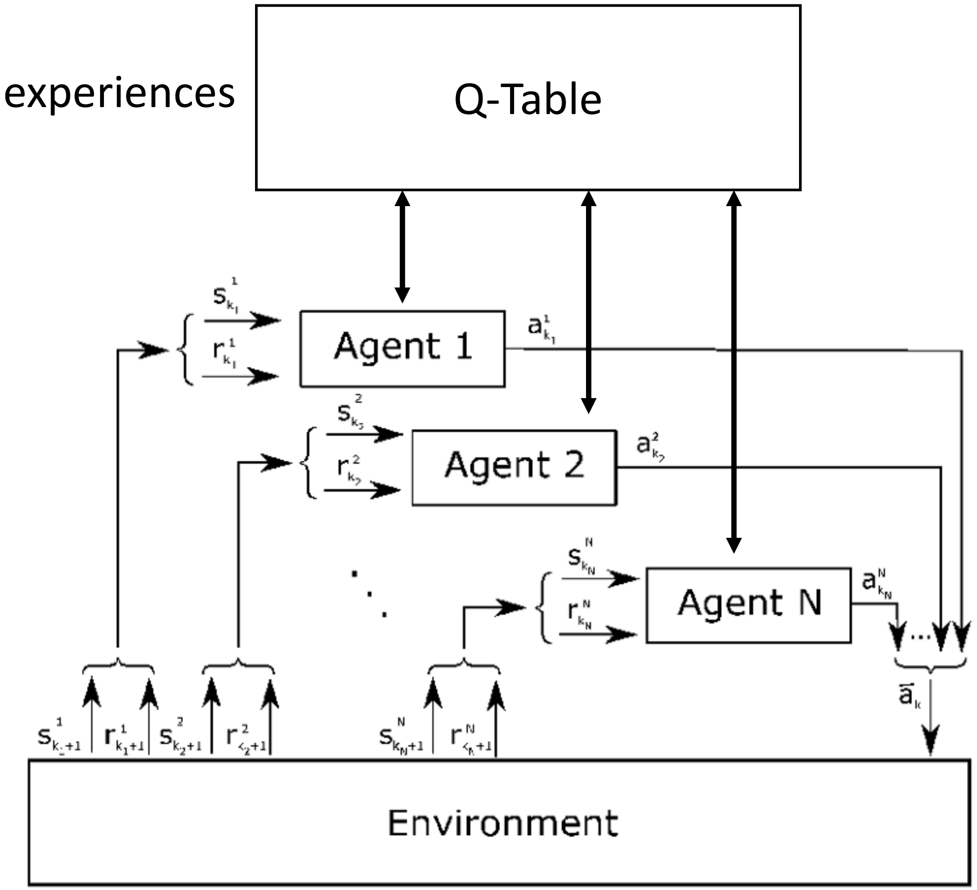
  flag: +1, mine: -1, otherwise: 0
  if any agent receives a non-zero reward, the agents should become inactive status.
  Agent should maintain its own list to prevent from updating Qtable.
  Do not cheat, environment also maintains its own agent status (no more change to the agents)



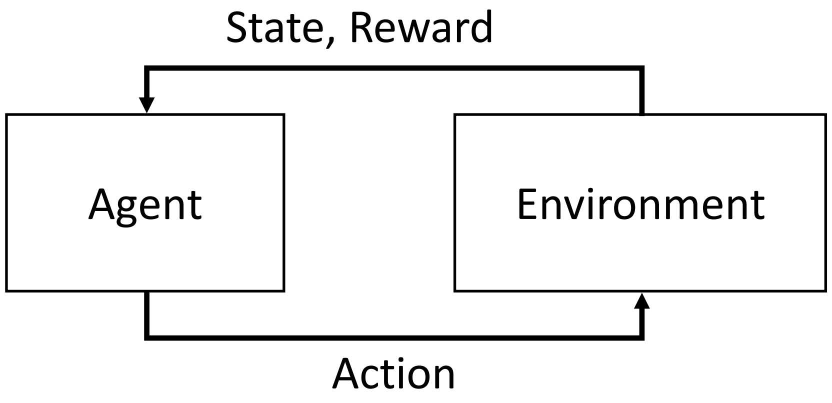
"Hogwild" approach

Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent

Shared experiences



Asynchronous Q-learning



Initialize $Q(s, a) = 0, \forall s \in S, a \in A(s)$ ← agent_init()

Repeat (for each episode):

  Initialize S

  Repeat (for each step of episode:)

    $\left.\begin{array}{l}\text{Choose A from current state } S \text{ using policy derived from Q (e.g. } \epsilon\text{-greedy)}\\ \text{Take action A}\end{array}\right\}$ agent_action()

    $\left.\begin{array}{l}\text{Observe next state } S' \text{ and } R\\ Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]\end{array}\right\}$ Asynchronous agent_update()

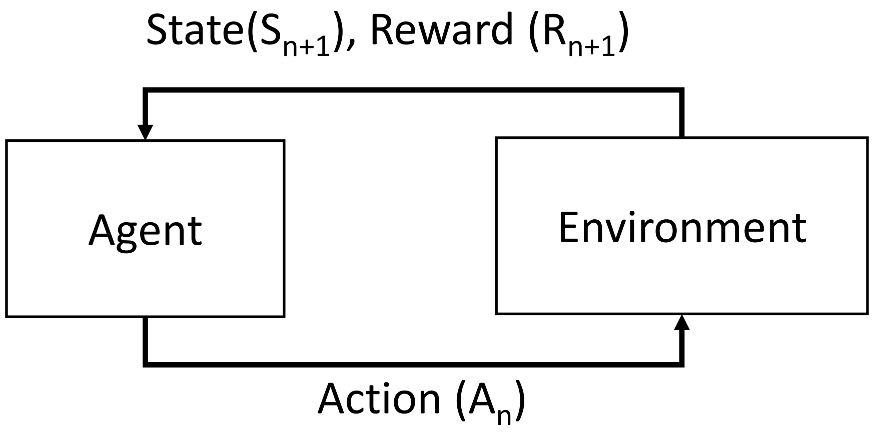
    $S \leftarrow S'$

  Until S is terminal

R.S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

Agent: Action

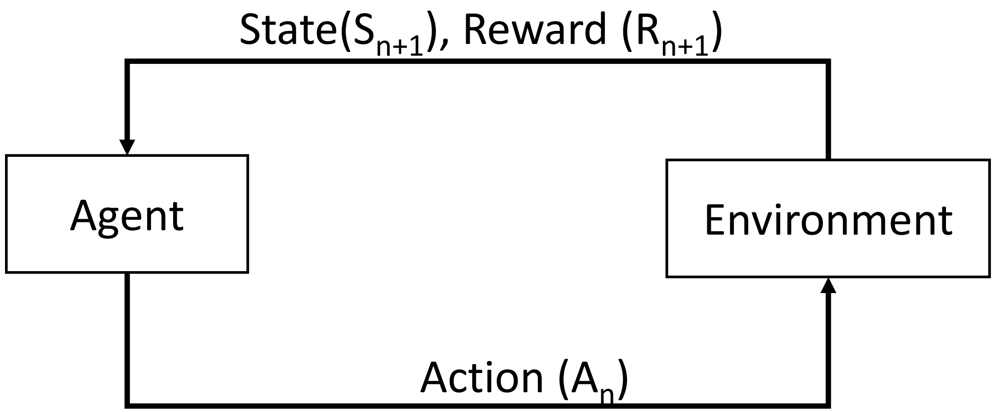
- Policy: ε-Greedy
- Action
  0: right, 1: down , 2: left, 3: up
- S_n: current state
- You need to decrease ε every episode
  ex) initial value: 1.0 -> 0.1 (minimum value)
  Exploration is more important from beginning
- Exploitation: Make the best decision given current information
- Exploration: Gather more information



$$
\begin{aligned}
&\textbf{if} \ (\text{uniform}(0,1) < \epsilon) \\
&\quad \text{a} = \text{uniform}(0, \ \#\text{actions}) \leftarrow \text{Exploration} \\
&\textbf{else} \\
&\quad \text{a} = \arg\max_{a'} Q(S_n, a') \leftarrow \text{Exploitation (Greedy policy)}
\end{aligned}
$$

Agent: update Q-table: Q-learning

- Update Q-table
- R_{n+1}: reward
- S_{n+1}: next state
- S_n: current state



α : learning rate

γ : discount factor

$$Q(S_n, A_n) = Q(S_n, A_n) + \begin{cases} \alpha\Big(R_{n+1} + \gamma \max_{a'} Q(S_{n+1}, a') - Q(S_n, A_n)\Big), & R_{n+1} = 0 \\ \alpha\Big(R_{n+1} - Q(S_n, A_n)\Big), & R_{n+1} \neq 0 \end{cases}$$

Source code view

```
if (m_episode == 0 && m_steps==0) {// only for first episode
    env.reset(m_sid);
    agent_init(); // initQ table + self initialization
}else {
    active_agent = checkstatus(board, env.m_state, flag_agent);

    if (m_newepisode) {
        env.reset(m_sid);
        agent_init_episode(); // set all agents in active status
        float epsilon = agent_adjustepsilon(); // adjust epsilon
        m_steps = 0;
        printf("EP=%4d, eps=%4.3f\n", m_episode, epsilon);
        m_episode++;
    }else {
        short* action = agent_action(env.d_state[m_sid]);
        env.step(m_sid, action);
        agent_update(env.d_state[m_sid], env.d_state[m_sid ^ 1], env.d_reward);

        m_sid ^= 1;
        episode = m_episode;
        steps = m_steps;
    }}
m_steps++;  env.render(board, m_sid); return m_newepisode;
```

Algorithm view

Initialize $Q(s, a) = 0, \forall s \in S, a \in A(s)$

Repeat (for each episode):

  Initialize S

  Repeat (for each step of episode:)

    Choose A from current state S using policy derived from Q (e.g. ϵ-greedy)

    Take action A

    Observe next state S' and R

    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

    $S \leftarrow S'$

  Until S is terminal

References

- David Silver, Reinforcement Learning lecture slides, 2015.