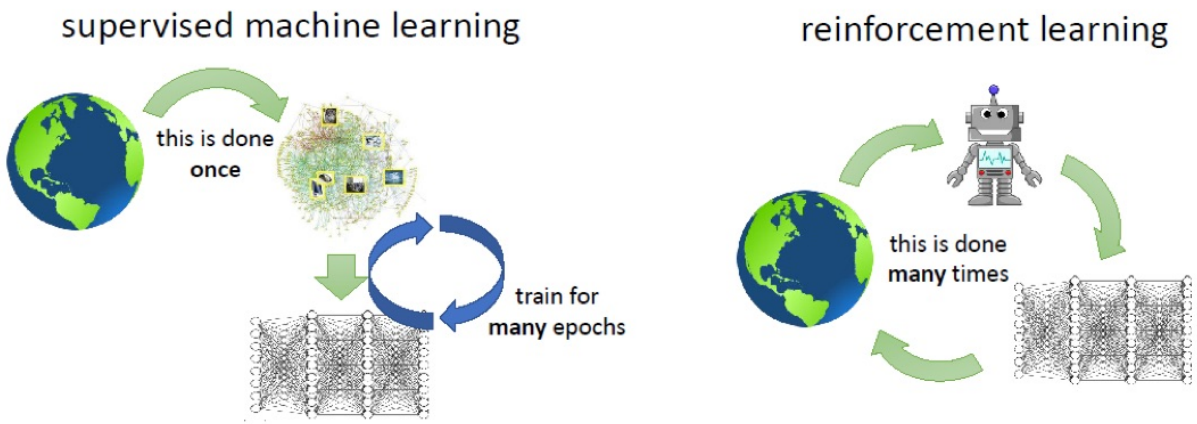
2024-11-16

Part 4 : Intro to Reinforcement Learning

- **Supervised learning**: Learn how to predict from labelled data $\{\mathbf{x}_i, y_i\}_{i=1}^N$.
- **Unsupervised learning**: Understand unlabelled data $\{\mathbf{x}_i\}_{i=1}^N$, learn how to generate and manipulate data.
  In both cases, the data are collected beforehand.



- **Reinforcement Learning**: Learn how to act from experiences $\{(s, a, r, s')\}$ with the environment, which are collected by the learning agent itself.
  So, RL is a kind of **active learning**.

* Hide and Seek video

Multi agent reinforcement learning

* This course: Single-agent RL

  * L14: Fundamentals of RL
  * L15: Value-based deep RL
  * L16: Police-based deep RL

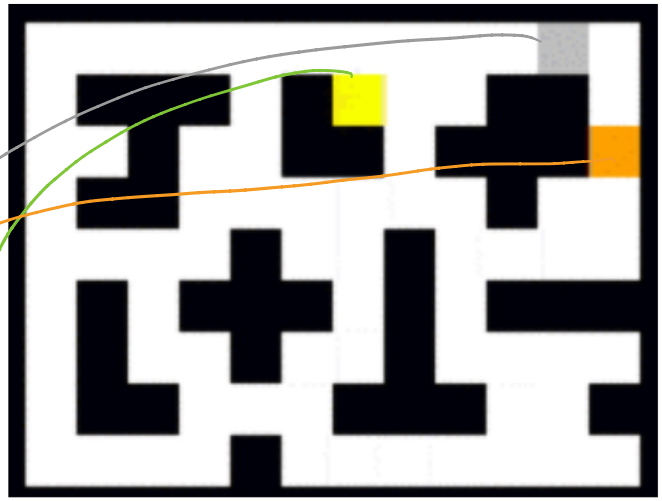
Applications of RL

* Games: AlphaGo, Naruto Mobile (Tencent), ...
* LLM alignment
* Robotics, self-driving cars

....

Lecture 14: Fundamentals of Reinforcement Learning

A concrete example



- A cat (orange), a mouse (gray), a piece of cheese (yellow) in a maze environment.
- Actions for the mouse and the cat: Move up, down, left and right.
- The mouse gets a **reward** at each time step:
  - -100: if eaten by the cat
  - 50: if eats the cheese
  - -1: otherwise

* cheese : Respawn at random location after each game

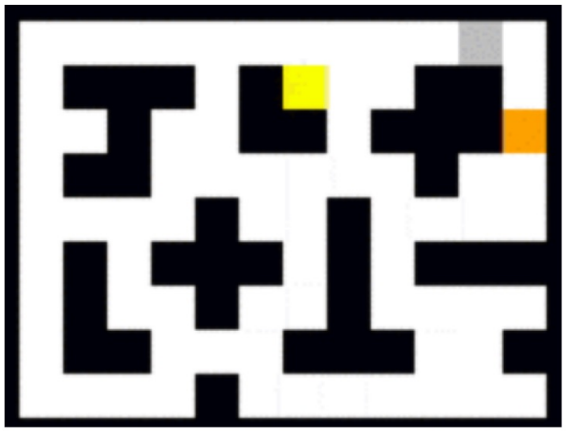
* mouse : Try to get cheese, avoid cat

* cat : Try to catch mouse, preprogrammed & fixed

objective: Make mouse smart using a reward function

partially observable

fully observed



- At each step, the mouse knows the current situation s.
  - The maze, locations of cat and cheese.
  - s is called a **state**.
  - **State space** S: set of all possible states.
  - The mouse needs to pick an **action** a from a **set** A **of possible actions**.
- Needs to learn **policy** π: $S \to A$
  - $\pi(s)$ is the action to take in situation s.

Trial & error

Mouse learns by interacting with environment

* If gets cheese, repeat the preceding actions

* If eaten by cat, avoid the preceding actions

Videos : * After 15,000 iterations of training : Not very smart

* After 150,000 iterations of training : Quite smart

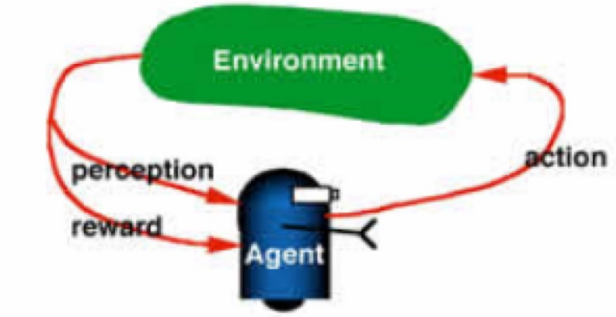
Reinforcement Learning and Markov Decision Process

- The cat is preprogrammed.
- If the mouse knows how that program works, then it could figure out the best policy. No need to learning from experiences.
- In general, if an agent has a **MDP (Markov Decision Process)** model of its environment, it can figure out the optimal policy.
- Next:
  - What are MDPs? How to derive optimal policies from MDPs?
- Later:
  - What to do if we don't have an MDP model of environment?
  - Answer: reinforcement learning.

Outline

Future Independent of past given present

- **Markov Decision Process (MDP)** is a model about how an agent interacts with its environment.
- At each step, Discrete time points
  - Environment is in some **state** s.
  - Depending on what s is, the agent takes an **action** a:
    - Environments moves to another state s'.
    - Agents gets an **immediate reward/penality** r.
  - Repeat



Specification of an MDP

* state space S
* Action space A

world dynamics

* Transition probability

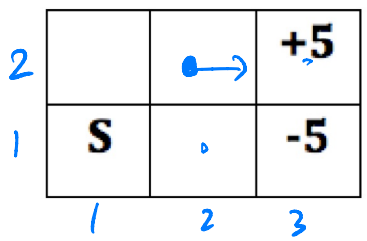
$$p(s' \mid s, a)$$

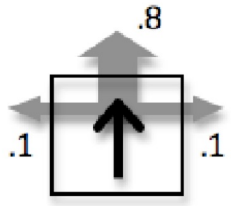
next state, current state, current action

* Reward function:

$$r(s, a, s')$$

Question 4: Consider an agent that acts in the gridworld shown below. The agent always starts in state $(1, 1)$, marked with the letter S. There are two terminal goal states, $(3, 2)$ with reward $+5$ and $(3, 1)$ with reward -5. Rewards are 0 in non-terminal states. (The reward for a state is received as the agent moves into the state.) The transition function is such that the intended agent movement (North, South, West, or East) happens with probability 0.8. With probability 0.1 each, the agent ends up in one of the states perpendicular to the intended direction. If a collision with a wall happens, the agent stays in the same state.





s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
$(2,2)$	E	$(3,2)$	0.8	5
		$(2,1)$	0.1	0
		$(2,2)$	0.1	0

in one step

Expected reward if taking action E at state $(2,2)$

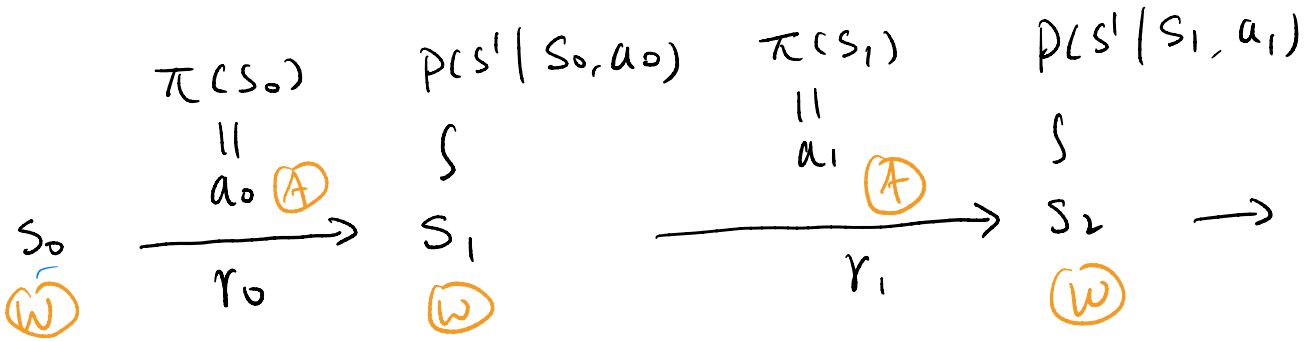
$r((2,2), E) = 5 \times 0.8 + 0 \times 0.1 + 0 \times 0.1$

$r(s, a) = \sum_{s'} p(s' \mid s, a)\, r(s, a, s')$

MDP specification: $p(s' \mid s, a)$, $r(s, a)$

Decision process guided by policy $\pi : S \to A$

Game btw agent & world



Discount factor $0 < \gamma < 1$, 0.95

Discounted cumulative (total) reward

$$R^{\pi}(s_0) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$$

← Average over multiple runs

$1 tomorrow worth less than $1 today

$$V^{\pi}(s) = E[R^{\pi}(s)]$$

Total reward for following policy π starting from state s

State value function

optimal policy, planning, RL

Theorem: Exists π^* s.t.

$$V^{\pi^*}(s) \geq V^{\pi}(s) \quad \forall \pi, \forall s$$

π^* : optimal policy

$V^*(s) = V^{\pi^*}(s)$: optimal state value function

Task 1: planning

$p(s'|s,a), r(s,a) \Rightarrow \pi^*$

Task 2: Don't know $p(s'|s,a), r(s,a)$

Experience tuples $\{(s,a,r,s')\} \Rightarrow \pi^*$

Task 1 planning

$$p(s'|s,a), r(s,a) \Longrightarrow \pi^*$$

① Estimate V^*

② $V^* \Longrightarrow \pi^*$

Step 1 of Task 1 Estimate V^*

* Initialize $V_0 = 0$
* Repeat until converge:

$$V_k \Longrightarrow V_{k+1}$$ ← improved estimation

If take a

improve ↗

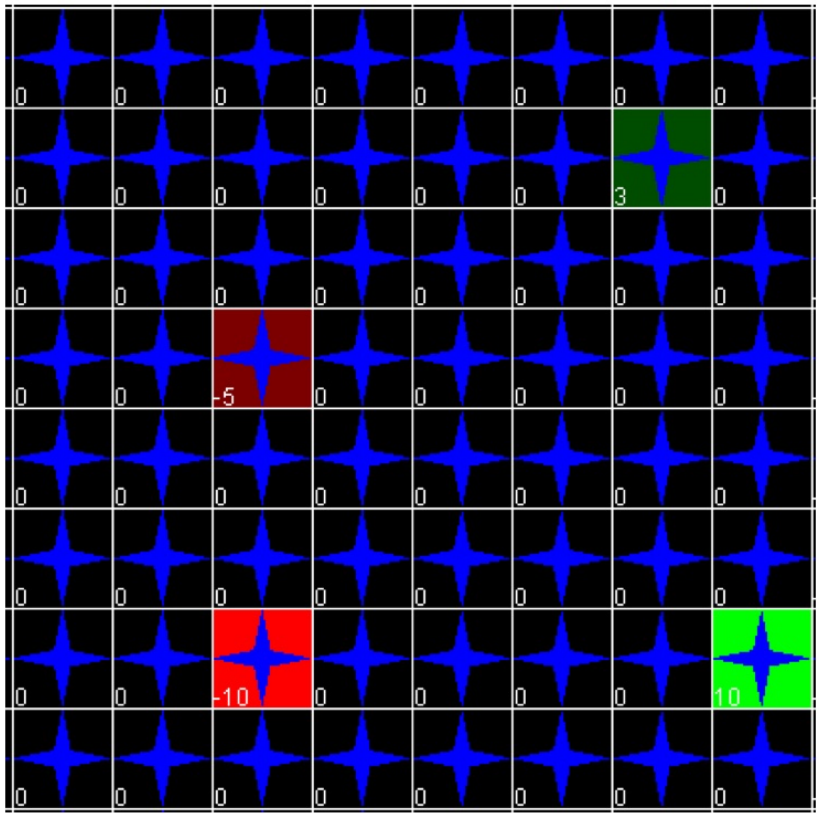
current reward

future reward

V_{k+1} = Bellman operator (V_k)

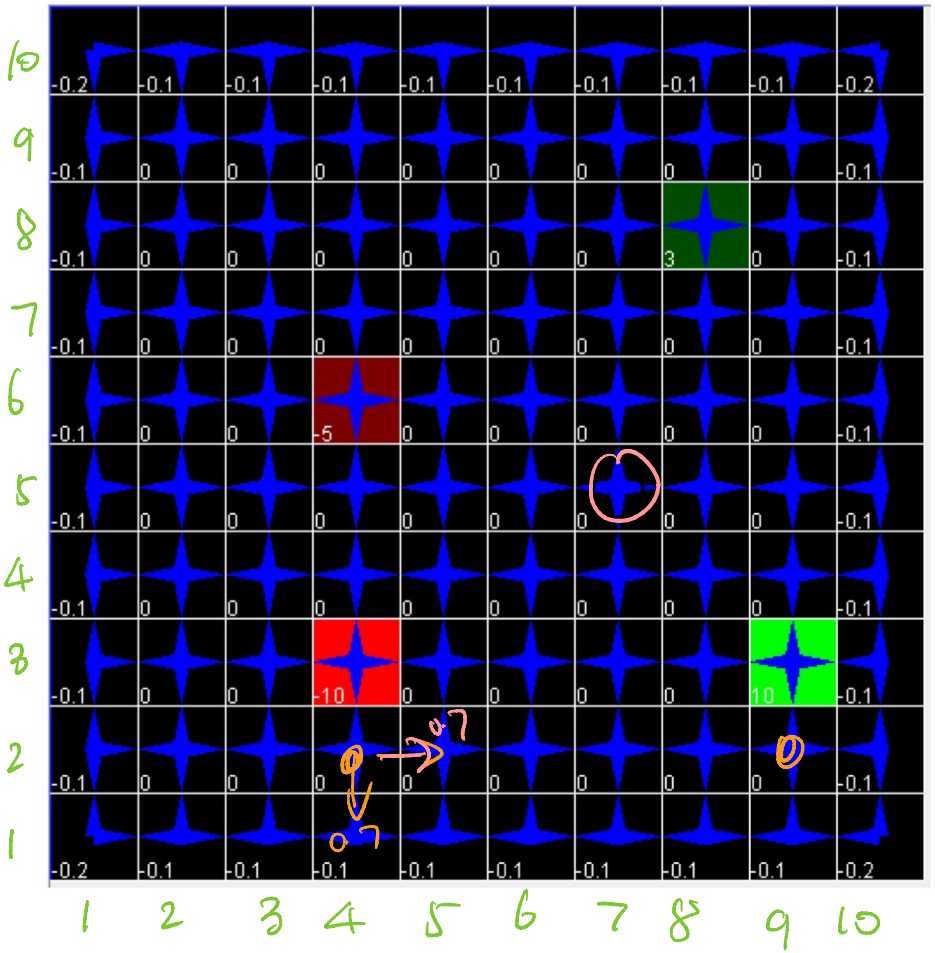
Algo: Value iteration

MDP example



- S: 8×8 grid
- A: 4 actions: up, down, left and right.
- Transition probability:
  - 0.7 chance move in intended direction, 0.1 chance in each of the other 3 directions.
  - Does not move when bumping against the wall.
- Reward: 4 reward states (reward obtained when leaving those states); -1 for bumping into walls.

V_1



V_2

$S = (9,2)$

a { up ≈ 6.3 ✓
down ≈ 0
left ≈ 0
right ≈ 0

$S = (4,2)$

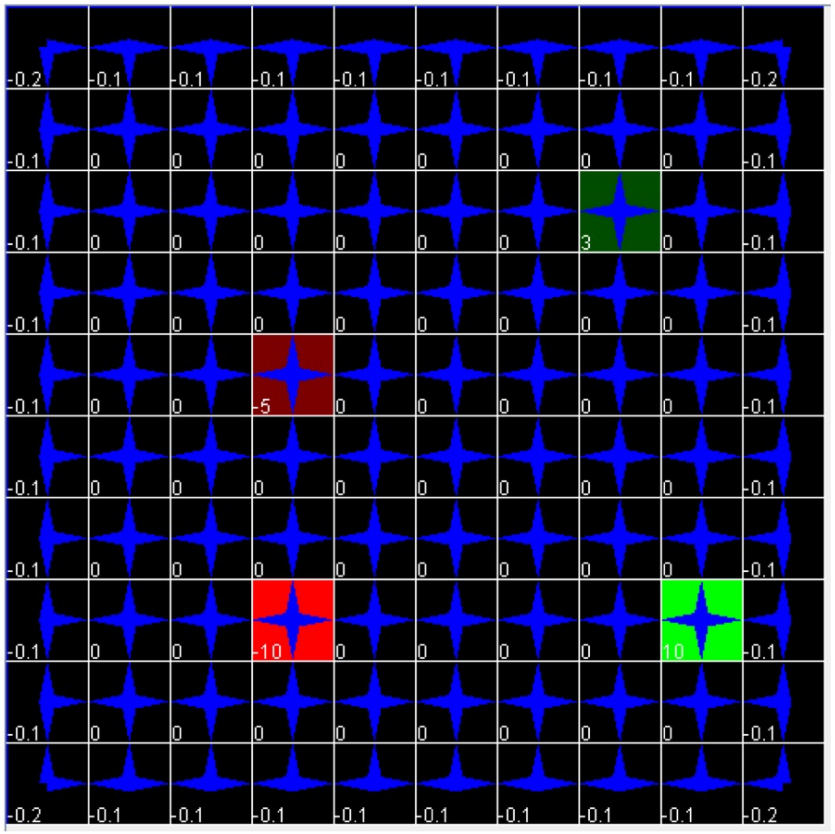
a { up ≈ -6.3
down small negative
left smaller negative ✓
right smaller negative ✓

$S = (7,5)$

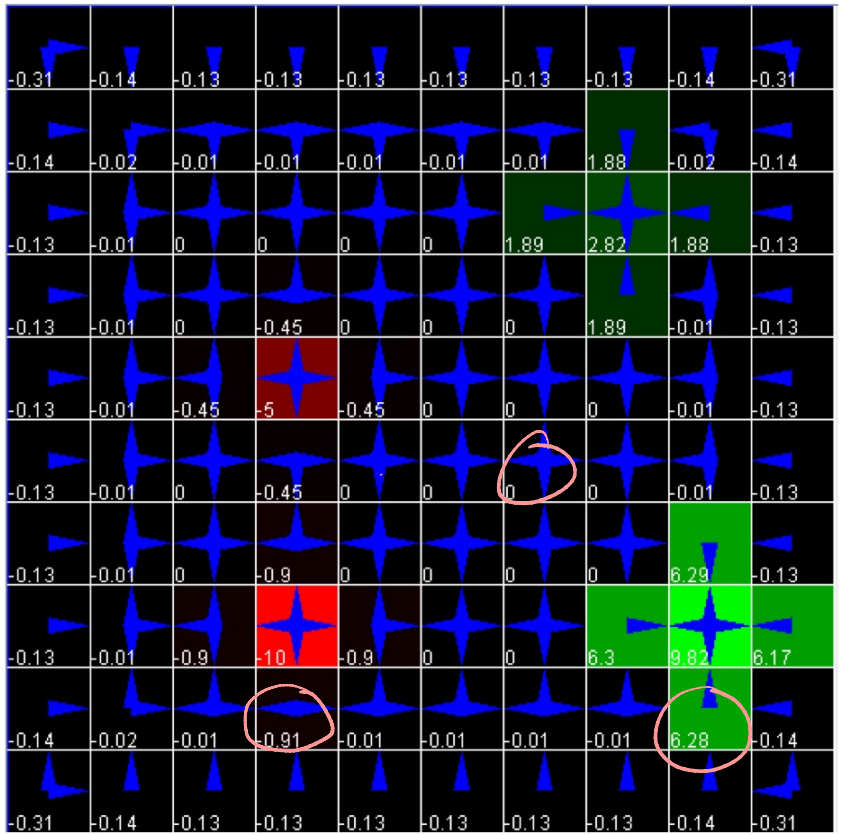
a { up $= 0$ ✓
down $= 0$ ✓
left $= 0$ ✓
right $= 0$ ✓

$$V_{k+1}(s) = \max_a \{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_k(s') \}$$

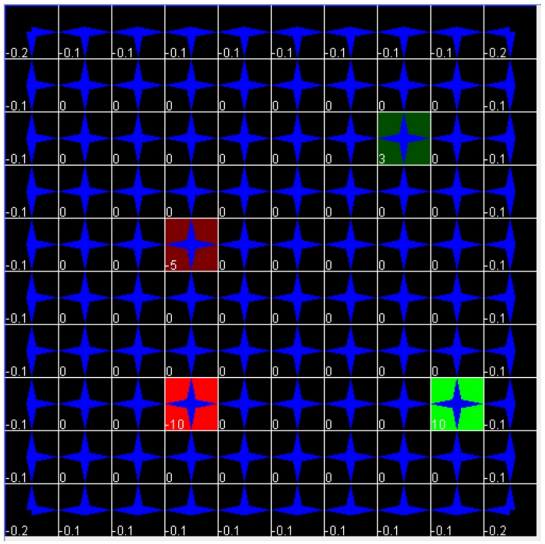
V_1


-0.2 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.2
-0.1 0 0 0 0 0 0 0 0 -0.1
-0.1 0 0 0 0 0 0 3 0 -0.1
-0.1 0 0 0 0 0 0 0 0 -0.1
-0.1 0 0 -5 0 0 0 0 0 -0.1
-0.1 0 0 0 0 0 0 0 0 -0.1
-0.1 0 0 0 0 0 0 0 0 -0.1
-0.1 0 0 -10 0 0 0 0 10 -0.1
-0.1 0 0 0 0 0 0 0 0 -0.1
-0.2 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.2

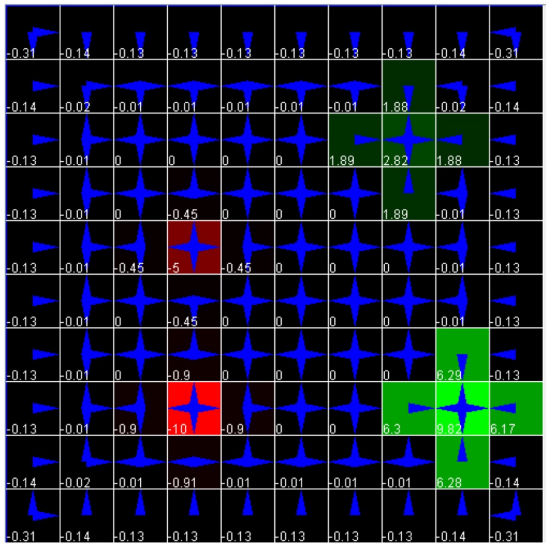
V_2


-0.31 -0.14 -0.13 -0.13 -0.13 -0.13 -0.13 -0.13 -0.14 -0.31
-0.14 -0.02 -0.01 -0.01 -0.01 -0.01 -0.01 1.88 -0.02 -0.14
-0.13 -0.01 0 0 0 0 1.89 2.82 1.88 -0.13
-0.13 -0.01 0 -0.45 0 0 0 1.89 -0.01 -0.13
-0.13 -0.01 -0.45 -5 -0.45 0 0 0 -0.01 -0.13
-0.13 -0.01 0 -0.45 0 0 0 0 -0.01 -0.13
-0.13 -0.01 0 -0.9 0 0 0 0 6.29 -0.13
-0.13 -0.01 -0.9 -10 -0.9 0 0 6.3 9.82 6.17
-0.14 -0.02 -0.01 -0.91 -0.01 -0.01 -0.01 -0.01 6.28 -0.14
-0.31 -0.14 -0.13 -0.13 -0.13 -0.13 -0.13 -0.13 -0.14 -0.31

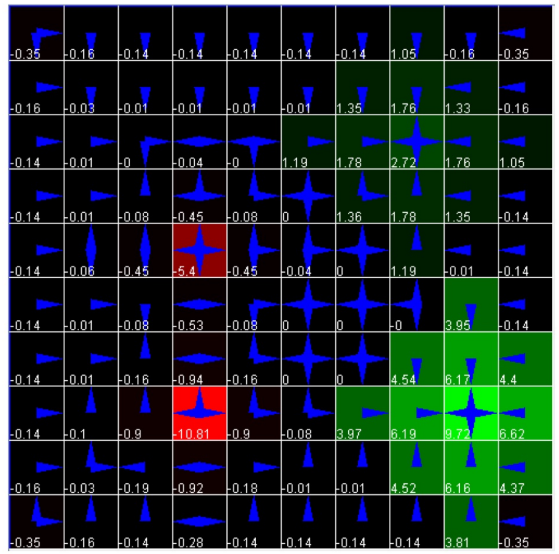
V_1



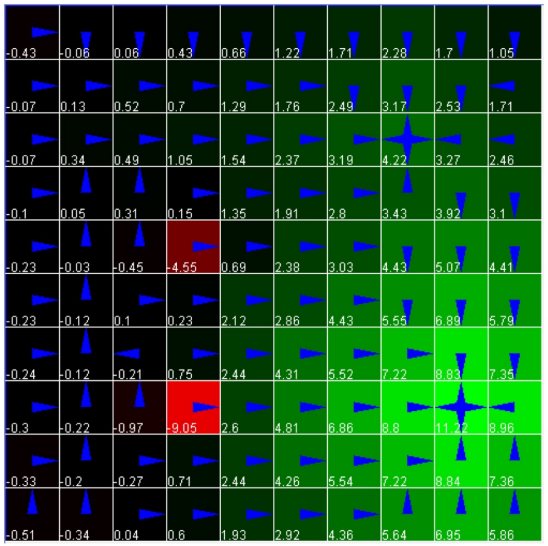
V_2



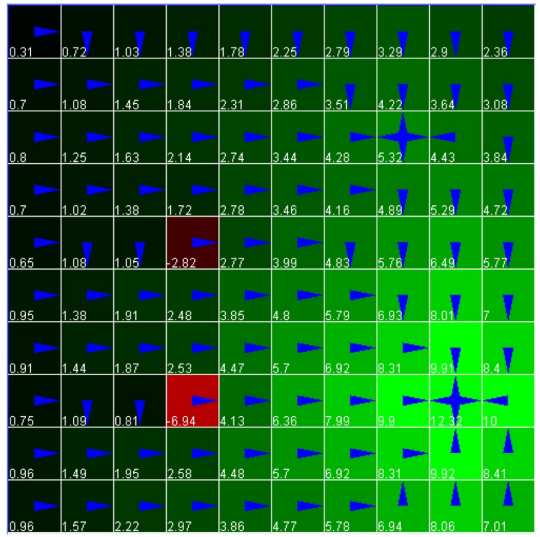
V_3



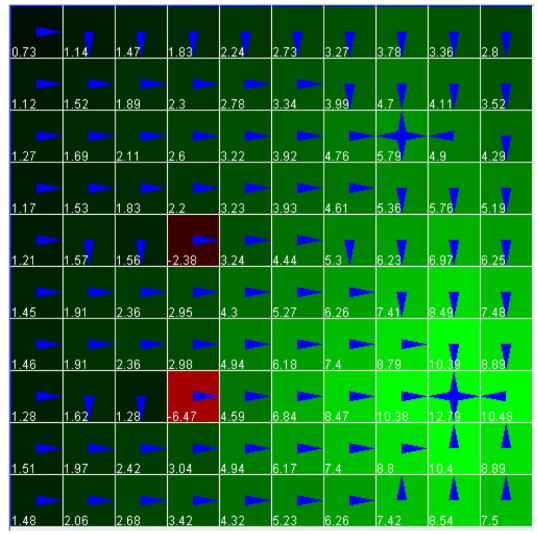
V_{10}



V_{20}



V_{30}



Quality of value function & policy improve with K

- **Value Iteration (VI)**:
  - Pick $V_0(s)$, $k = 0$
  - Repeat:
    - $V_{k+1}(s) = \max_a \{r(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s')\}$
    - $k = k + 1$

Theorem: $\lim_{k \to \infty} V_k(s) = V^*(s)$

$$\max_s |V_{k+1}(s) - V_k(s)| \leq \gamma \max_s |V_k(s) - V_{k-1}(s)|$$

Contraction property of Bellman Operator

Bellman's optimality Equation

$$V_{k+1}(s) = \max_a \{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_k(s') \}$$

$k \to \infty$

$$V^*(s) = \max_a \{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^*(s') \}$$

Q function

$$V^*(s) = \max_a \{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^*(s') \}$$

optimal state value function

Reward for

* Take a in s, then
* Act optimally

optimal state-value function : $Q^*(s,a)$

$$V^*(s) = \max_a Q^*(s,a)$$

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

step 2 of Task 1:

$V^* \Rightarrow \pi^*$

Value iteration in terms of Q function

$Q_K(s', a')$

* $Q_0 = 0$
* Repeat : $Q_K \Rightarrow Q_{K+1}$

$$Q_{K+1}(s,a) \qquad r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q_K(s',a')$$

current future ↖ improve

Theorem : $\lim_{K \to \infty} Q_K(s,a) = Q^*(s,a)$

$$\pi_K(s) = \arg\max_a Q_K(s,a)$$

Greedy policy based on Q_K

Value iteration in terms of Q function

Q_K - table

	a_1	a_2	a_3	π_{Q_K}
s_1	0.5	2.1	-1	a_2
s_2	0.9	0.1	0.5	a_1
s_3	-1	0.1	1.7	a_3

Outline

Task 1 : planning

$p(s'|s,a), r(s,a) \Rightarrow \pi^*$

Task 2 : Don't know $p(s'|s,a), r(s,a)$

Experience tuples $\{(s,a,r,s')\} \Rightarrow \pi^*$

Q - learning

Task 2 $\{(s, a, r, s')\} \Rightarrow \pi^*$

* Initialize $Q(s, a) = 0$
* collect (s, a, r, s')

Very small 10^{-4}

$$Q(s, a) \leftarrow Q(s, a) + \alpha \Delta$$

* No need of gradient descent
* No need for $p(s'|s,a)$, $r(s,a)$

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

TD target : improved estimation

Current

s	a	s'	$p(s'\|s,a)$	$r(s,a,s')$
(2,2)	E	(3,2)	0.8	5
		(2,1)	0.1	0
		(2,2) in one step	0.1	0

$Q((2,2), E)$ improve

$Q((2,1), E)$

$Q((2,2), N)$

improved over Q_K

$$Q_{K+1}(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q_K(s',a')$$

$s_1', s_2', \ldots, s_m' \sim p(s'|s,a)$

$$\approx r(s,a) + \gamma \frac{1}{m} \sum_{j=1}^{m} \max_{a'} Q_K(s_j', a')$$

what if $m = 1$

$$\approx r(s,a) + \gamma \max_{a'} Q(s_1', a')$$

* high variance
* Unbiased

How to improve $Q(s,a)$ using (s,a,r,s')

Answer: $r + \gamma \max_{a'} Q(s', a')$

Time Difference (TD) target

Q-Learning

- **Initialize** $Q(s, a)$
- **Repeat**
  - Collect experience tuple (s, a, r, s')

$$Q(s, a) \leftarrow Q(s, a) + \alpha[(r(\cancel{s, a}) + \gamma \max_{a'} Q(s', a')) - Q(s, a)]$$

TD target

Q-learning converges to $Q^*(s, a)$ if each (s, a) pair is updated infinitely often.

$$(1-\alpha)\, Q(s,a) + \alpha\,(r + \gamma \max_{a'} Q(s',a'))$$

The exploration vs exploitation tradeoff

- The agent needs to collect data for learning by exploring the environment.
- Exploration
  - Explore new parts of state space so as to gain more experiences.
  - Reward might not maximized.
- Exploitation
  - Make use of experience gained so far to maximize reward.
  - Might not gain new experiences
- ϵ-greedy policy:
  - With small probability ϵ, chose an action at random.
  - With probability $1 - \epsilon$, chose the action with the highest reward according to current estimates.

- Terminal/absorbing states: Cannot leave once entered.
- Example: 'Game Over'.
- To continue training, need to restart the game.
- Episode: The process from initial state to terminal state.

Decide

$\arg\max_a Q(s,a)$

The Q-Learning Algorithm

Hyper parameters

- ϵ: 0.1
- α: 10^{-4}
- γ: 0.9, 0.95

- Initialize $Q(s, a)$ arbitrarily.
- Repeat (for each episode)
  - Pick initial state s.
  - **Repeat**
    - Choose a for the state s (ϵ-greedy with $\arg\max_a Q(s, a)$)
    - Take action a, observe r and s'
    - Update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

$$s \leftarrow s'$$

  - **until** s is terminal

large γ means look further into future

t	0	1	2	3	...
γ^t	1	0.5	0.25	0.125	
	1	0.9	0.81	0.72	...

WA4 Q4 (b)

HA 10

* Demo of expected behavior : Random agent
  Q-learning agent

* provided code

  * Game board, random agent

* To do : policy.py, @ LearningPolicy class

  * Set hyperparameters
  * Decide : greedy policy
  * update_q_table : @ Learning algorithm
  * Do training