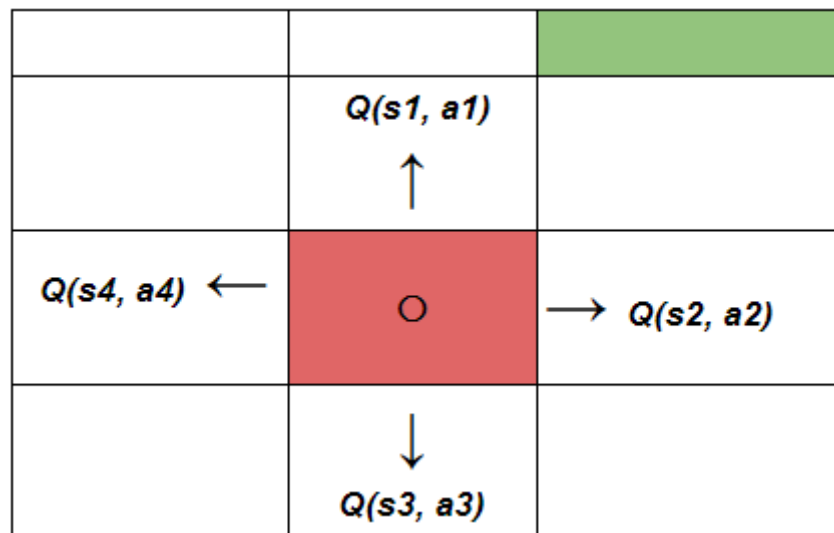# L7.2 Q-Learning

Zonghua Gu 2022

Outline

- Introduction to RL
- Q-Learning
- Deep Q Learning w. Function Approximation

Reinforcement Learning

- Recall: an MDP consists of:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions and rewards $p(r, s'|s, a)$ (w. discount γ)
- But now the model $p(r, s'|s, a)$ is unknown
  - Unknown reward r and next state s', denoted as state transition (s, a, r, s'), if agent takes action a in state s.
  - Agent must learn the optimal policy $\pi(a|s)$ by trial-and-error.

Warm

Cool

Overheated

State ⟶ | Model $p(r, s'|s, a)$ | ⟶ Reward
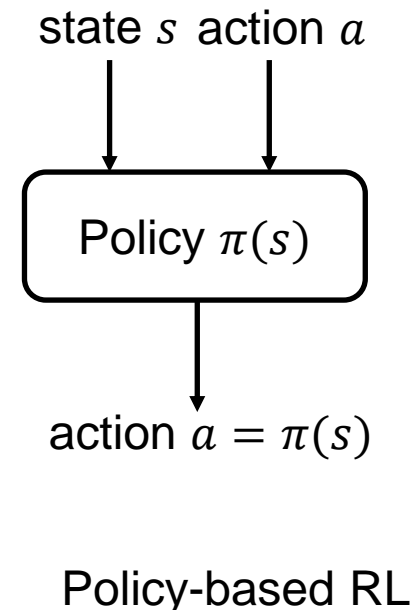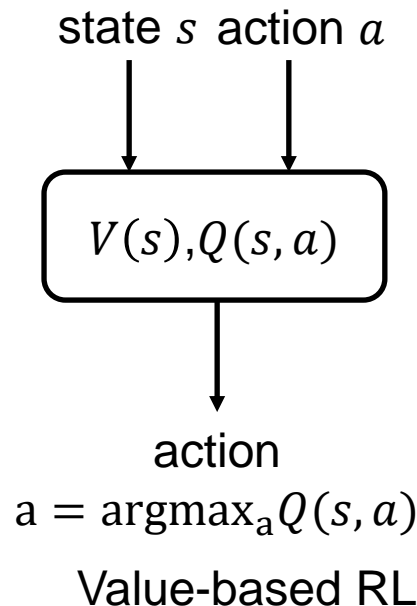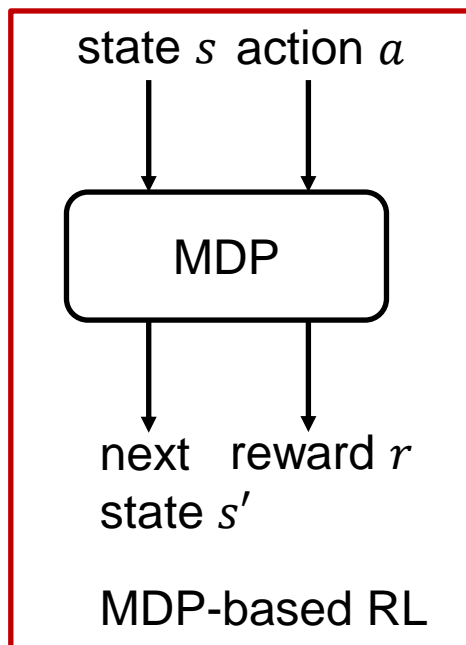Action ⟶ | | ⟶ Next state
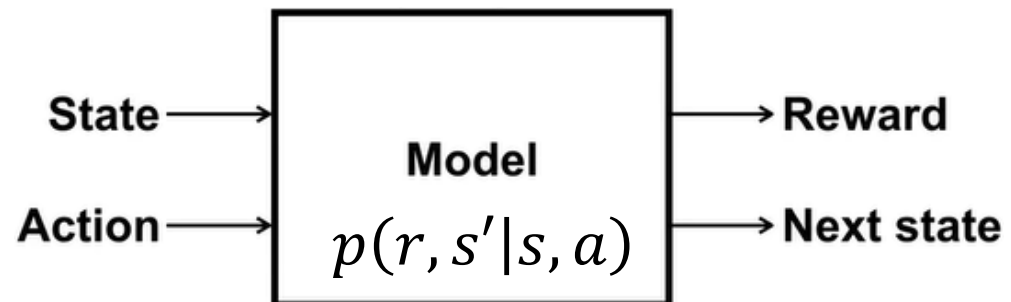
Characteristics of RL

- There is no supervisor, only a reward signal (may be sparse)

- Feedback is delayed, not instantaneous

- Sequential, non i.i.d (Independent and identically distributed) data

 - Agent's actions affect the subsequent data it receives

Model-Based vs. Model-Free

- Model-Based RL: MDP planning
  - Learn MDP $p(r, s'|s, a)$ (given current state s and action a, returns prob distribution of current reward r and next state s'), then plan with Value Iteration or Policy Iteration
- Model-Free RL: Value-based (our focus) and Policy-based
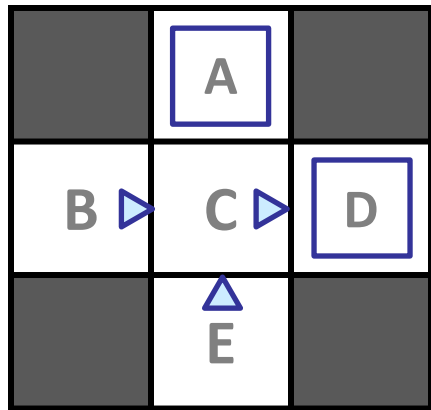 - Learn value function $V(s)$ or $Q(s, a)$, or policy function $\pi(s)$ without learning MDP

state s action a state s action a state s action a

| MDP | $V(s), Q(s, a)$ | Policy $\pi(s)$ |

next reward r action   action $a = \pi(s)$
state s' $a = \text{argmax}_a Q(s, a)$

MDP-based RL   Value-based RL   Policy-based RL

Model-Based RL

- If MDP is not available, we can use Model-Based RL:
- Step 1: Learn empirical MDP model
  - Estimate the model $p(r, s'|s, a)$ by executing some policy π (may be random), and keeping track of outcomes r, s' for each s, a in the observed episodes.
- Step 2: Do planning w. the learned MDP for the optimal policy
  - Dynamic Programming w. Value Iteration or Policy Iteration
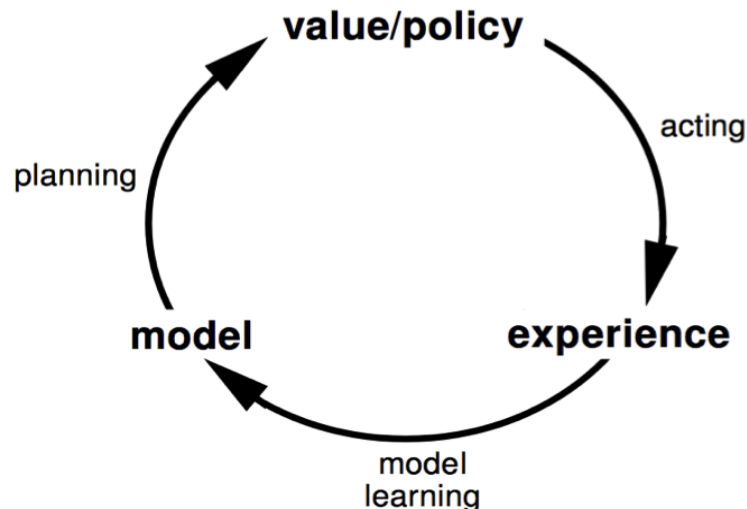
MiniGW: Model Learning

Episode 1

$(B, r, C, -1)$
$(C, r, D, -1)$
$(D, exit, x, 10)$

Episode 2

$(B, r, C, -1)$
$(C, r, D, -1)$
$(D, exit, x, +10)$

Episode 3

$(E, u, C, -1)$
$(C, r, D, -1)$
$(D, exit, x, 10)$

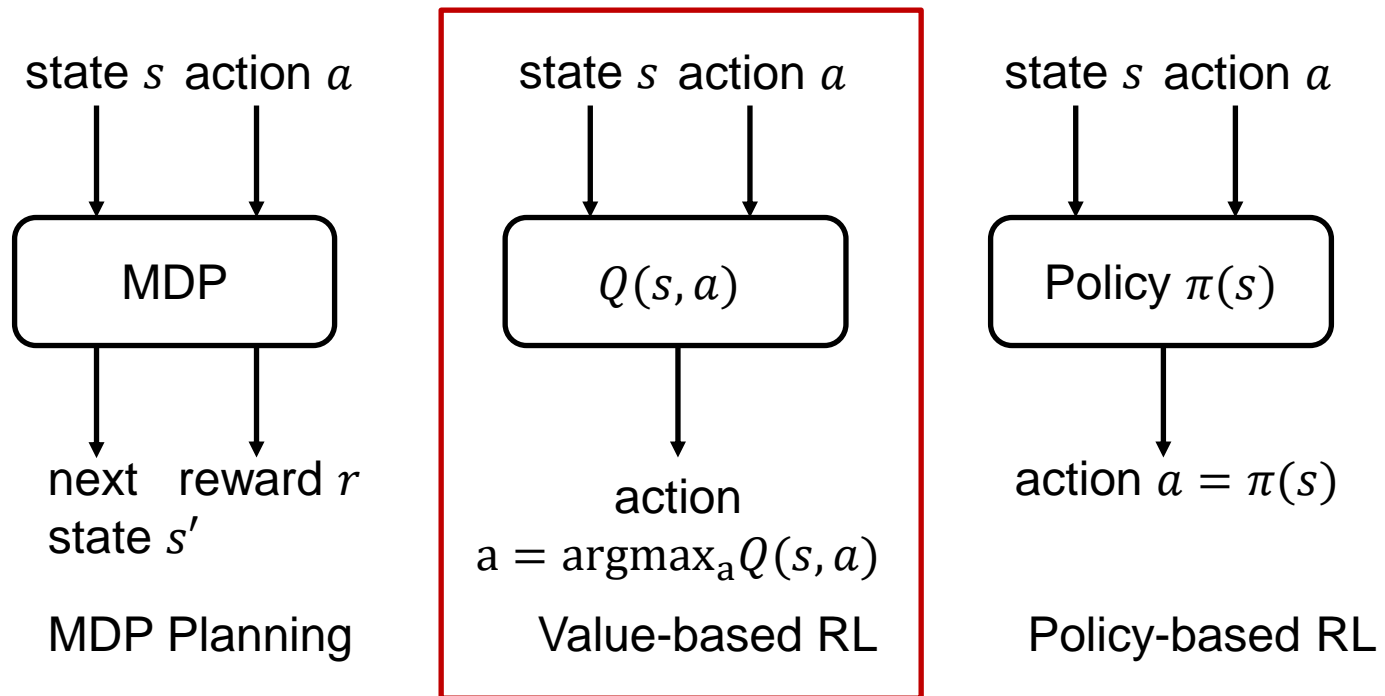Episode 4

$(E, u, C, -1)$
$(C, r, A, -1)$
$(A, exit, x, -10)$

$p(s'|s, a)$

$p(-1, C|B, r) = 1.0$
$p(-1, D|C, r) = 0.75$
$p(-1, A|C, r) = 0.25$
$p(10, x|D, exit) = 1.0$
$p(-10, x|A, exit) = 1.0$

A

B ▷   C ▷   D

E

Assume: $\gamma = 1$

- In the 4 episodes, we see 4 transitions from $(s = C, a = r)$. 3 of them go to next state $s' = D$, and one goes to next state $s' = A$, each w. reward -1. Hence $p(-1, D|C, r) = 0.75; p(-1, A|C, r) = 0.25$.

Value-based RL

state s action a

↓ ↓

MDP

↓ ↓

next reward r
state s'

MDP Planning

state s action a

↓ ↓

$Q(s, a)$

↓

action
$a = \mathrm{argmax}_a Q(s, a)$

Value-based RL

state s action a

↓ ↓

Policy $\pi(s)$

↓

action $a = \pi(s)$

Policy-based RL

Outline

- Introduction to RL
- Q-Learning
- Deep Q Learning w. Function Approximation

Q-Learning

- [BOA] Bellman Optimality Equation for Optimal Action Value Function:
  - $q_*(s, a) = \sum_{r,s'} p(r, s'|s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]$

- Q Learning solves [BOA] by sampling:
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
  - γ is the discount factor in MDP; α is the learning rate
  - (Note slight difference in notation: S_t and S_{t+1} above is the same as S and S' below)

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A) \right]$
        $S \leftarrow S'$
    until S is terminal

Off-policy, Value Iteration: in state S, Q update w. one-step lookahead $Q(S', a)$ by taking $\max_a Q(S', a)$ among all possible actions.

Q-Learning Update Equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New
Q-value
estimation

Former
Q-value
estimation

Learning
Rate

Immediate
Reward

Discounted Estimate
optimal Q-value
of next state

Former
Q-value
estimation

TD Target

TD Error

Q-Function

- $Q(S, A)$ maps from state S and action A to a Q value.

- In Tabular QL, $Q(S, A)$ is a Q-table, where each cell corresponds to a state-action value pair value.

- In Deep QL, $Q(S, A)$ is a Neural Network

Q-Learning Steps

- During training: train a Q-Function (an action-value function), which is a Q-table that contains all the state-action pair values.
  - Given a state and an action, our Q-Function will search into its Q-table the corresponding value.
  - When the training is done, we have an accurate Q-function
- During inference: given the accurate Q-function, the optimal policy is to choose the greedy (best) action that results in the highest Q value $\text{a} = \text{argmax}_\text{a} Q(s, a)$

state s action a

$$Q(s, a)$$

action
$$\text{a} = \text{argmax}_\text{a} Q(s, a)$$

13
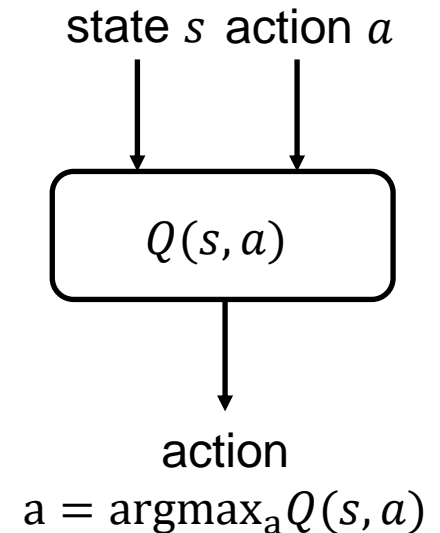
Exploration-Exploitation Dilemma

- The agent has to exploit what it has already experienced in order to obtain reward, but it also has to explore in order to make better action selections in the future
  - Exploitation: to obtain high reward, the agent prefers actions that it has tried in the past and proven to give high reward
  - Exploration: to discover such actions, it has to try actions that it has not selected before

ϵ-Greedy Policy

- ϵ-greedy policy: select a random action w. prob ϵ (exploration); select the greedy action $\underset{a}{\operatorname{argmax}} Q(s, a)$ with prob $1 - \epsilon$ (exploitation)
  - With $|\mathcal{A}(s)|$ possible actions in state s, select each non-greedy action w. prob $\frac{\epsilon}{|\mathcal{A}(s)|}$; the greedy action w. prob $1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$
  - e.g., if $\epsilon = .1$, select each non-greedy action w. prob $\frac{.1}{6} = .017$; the greedy action w. prob $1 - .1 + \frac{.1}{6} = .917$
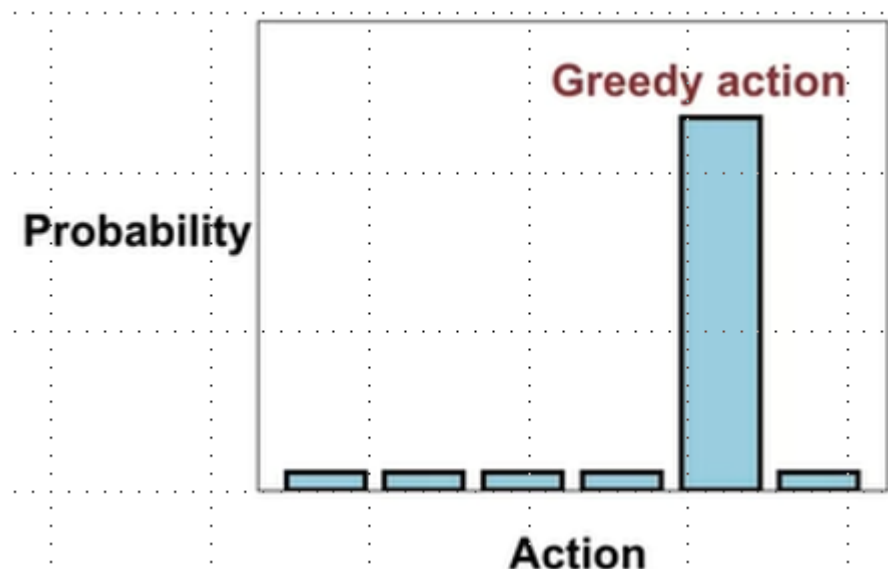
ϵ is Gradually Reduced during Training

- At the beginning of the training, the agent knows very little. We set ϵ to be large (close to 1), so the agent is adventurous and explores a lot

- As the training goes on, the Q-Table gets better and better in its estimations, we progressively reduce ϵ since we will need less and less exploration and more exploitation

- During inference: $\epsilon = 0$ (always take the greedy action)

Epsilon

Training
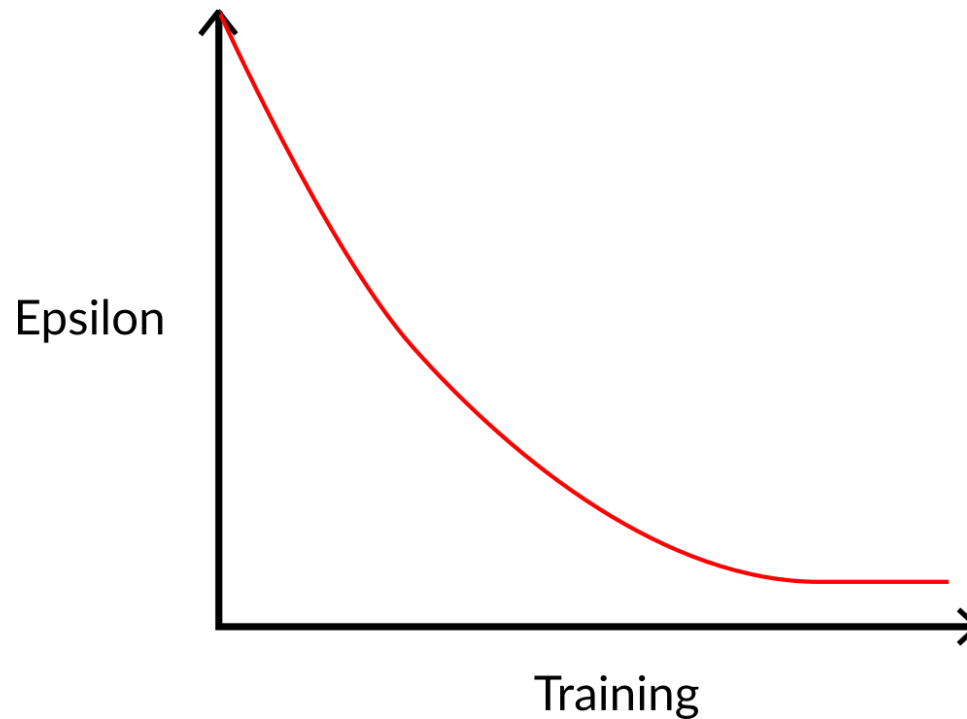
Example: a Maze

- Agent (rat) always starts at the same starting point
- It has 4 possible actions in each grid position (Left, Right, Up, Down)
- Goal: eat the big pile of cheese (at the lower right-hand corner) and avoid the poison
- The episode ends if rat eats the poison, eats the big pile of cheese, or if it spends more than 5 steps.
- Learning rate $\alpha = .01$, discount factor $\gamma = .99$
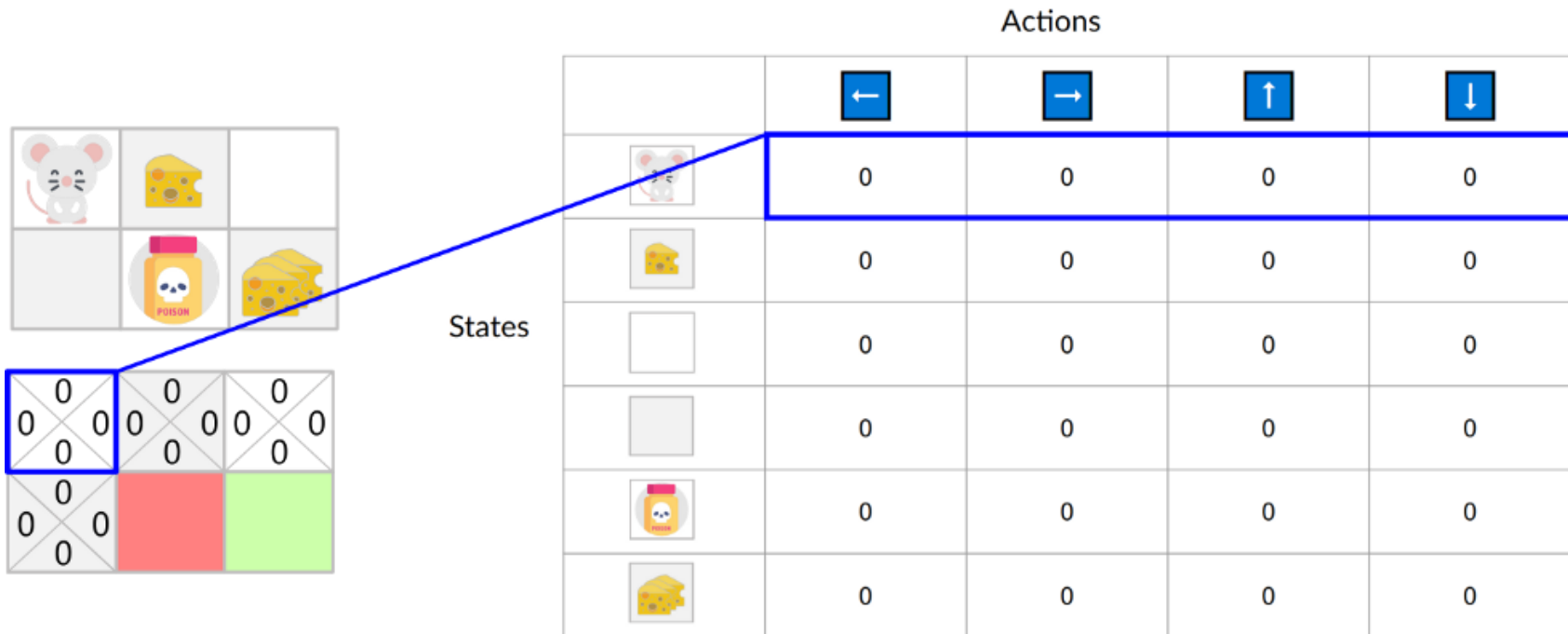
Reward Function

- +0: Going to a state with no cheese in it.
- +1: Going to a state with a small cheese in it.
- +10: Going to the state with the big pile of cheese.
- -10: Going to the state with the poison and thus die.
- +0 If it spends more than five steps.

Initial Q-Table

- All values are 0. This table contains, for each state, the four state-action values.
  - $Q(S, L), Q(S, R), Q(S, U), Q(S, D)$

Actions

	$\leftarrow$	$\rightarrow$	$\uparrow$	$\downarrow$
🐭	0	0	0	0
🧀	0	0	0	0
	0	0	0	0
	0	0	0	0
☠	0	0	0	0
🧀	0	0	0	0

States

Step 1

- Agent takes a random action, and moves right
- It gets a small cheese with +1 reward
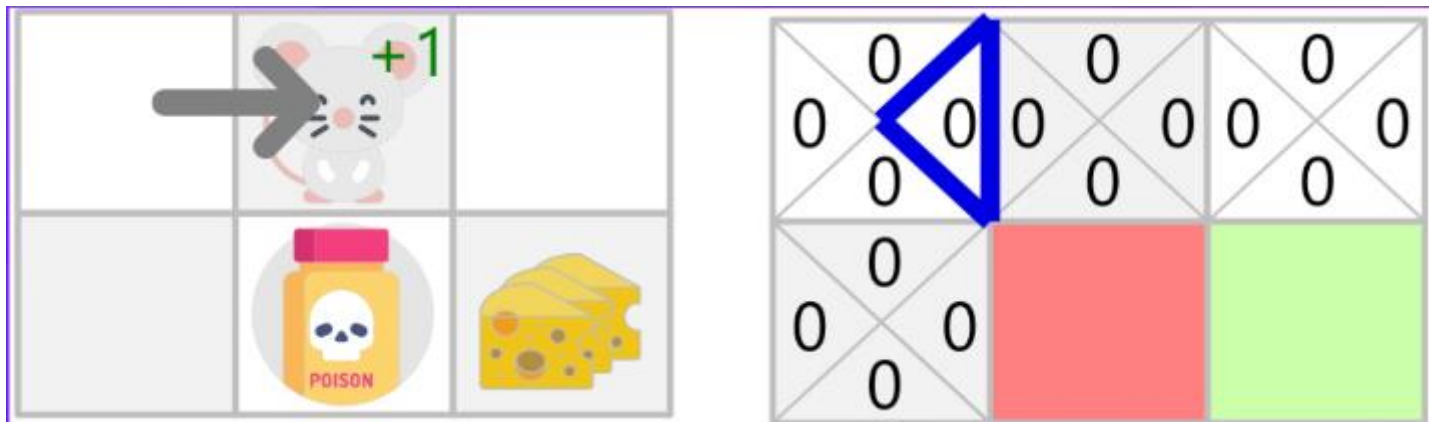
Q-Table Update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$$

Q(Initial state, Right) = 0 + 0.1 * [1 + 0.99 * 0 - 0]
Q(Initial state, Right) = 0.1

	←	→	↑	↓
🐭	0	0.1	0	0
🧀	0	0	0	0
	0	0	0	0
	0	0	0	0
🍯	0	0	0	0
🧀	0	0	0	0

Step 2

- Agent takes a random action, and moves down

- It gets the poison with -10 reward

- Episode ends
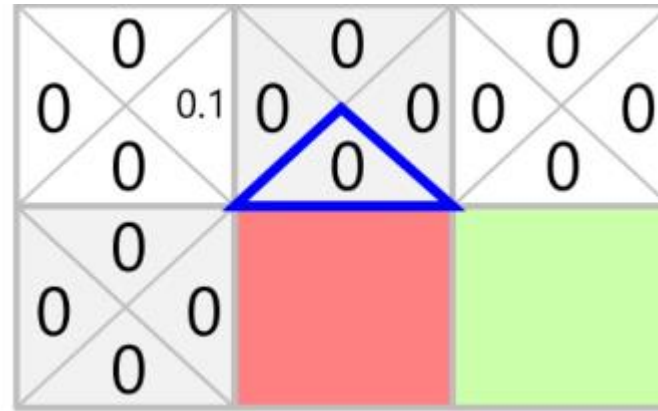
Q-Table Update

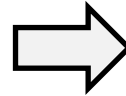$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$$

Q(State 2, Down) = 0 + 0.1 * [-10 + 0.99 * 0 - 0]
Q(State 2, Down) = -1

	←	→	↑	↓
🐭	0	0.1	0	0
🧀	0	0	0	-1
	0	0	0	0
	0	0	0	0
🍯	0	0	0	0
🧀	0	0	0	0

23

After Some Training Episodes

	←	→	↑	↓
🐭	0	0	0	0
🧀	0	0	0	0
⬜	0	0	0	0
⬜	0	0	0	0
☠️	0	0	0	0
🧀	0	0	0	0

⟹

	←	→	↑	↓
🐭	0	10.8	0	0
🧀	0	9.9	0	-10
⬜	0	0	0	10
⬜	-10	0	0	0
☠️	0	0	0	0
🧀	0	0	0	0

Outline

- Introduction to RL

- Q-Learning

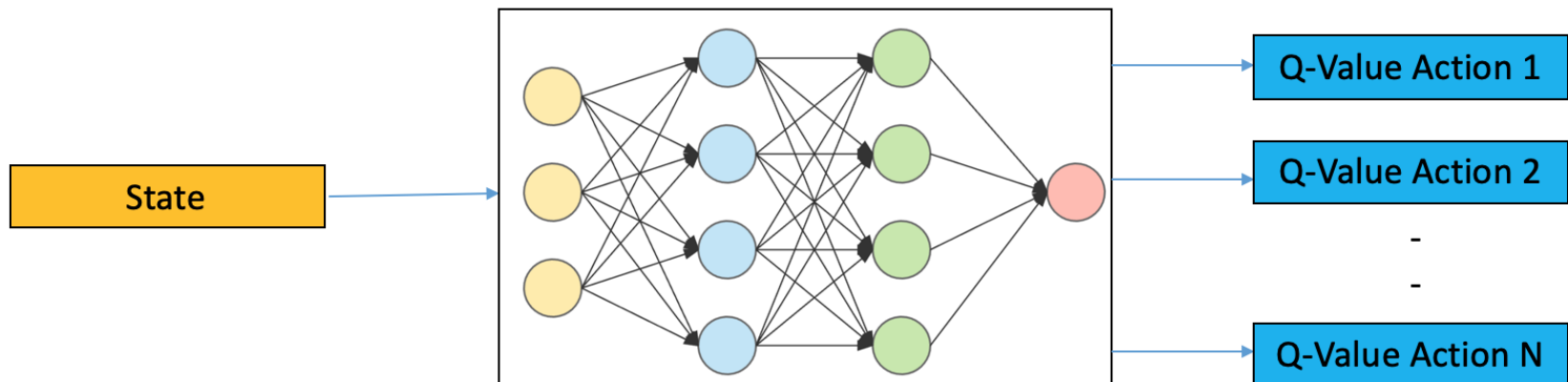- Deep Q Learning w. Function Approximation

Q Learning vs. Deep QL
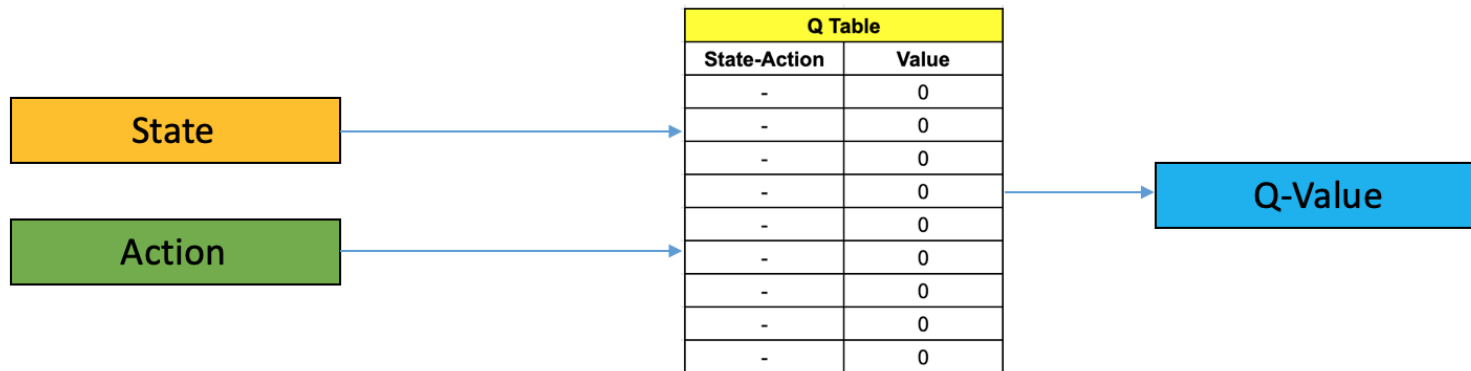
Q Learning

Deep Q Learning

Function Approximations of Value Functions

- Upper:
  - Left: state value function $\hat{v}(s, \mathbf{w})$ with params $\mathbf{w}$
  - Middle: action value function $\hat{q}(s, a, \mathbf{w})$ with params $\mathbf{w}$
  - Right: action value functions $\hat{q}(s, a_i, \mathbf{w})$ with params $\mathbf{w}$, since we need all Q-values for computing greedy policy $\text{argmax}_a \hat{q}(s, a, \mathbf{w})$

- Lower:
  - Use Neural Network as action value functions (corresponds to upper middle and right)
  - Optimized with Gradient descent
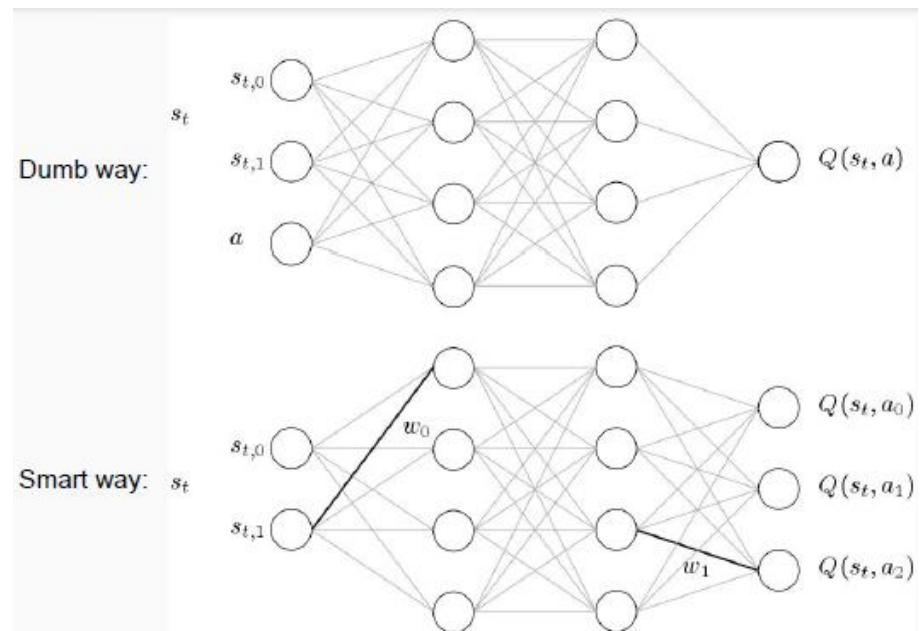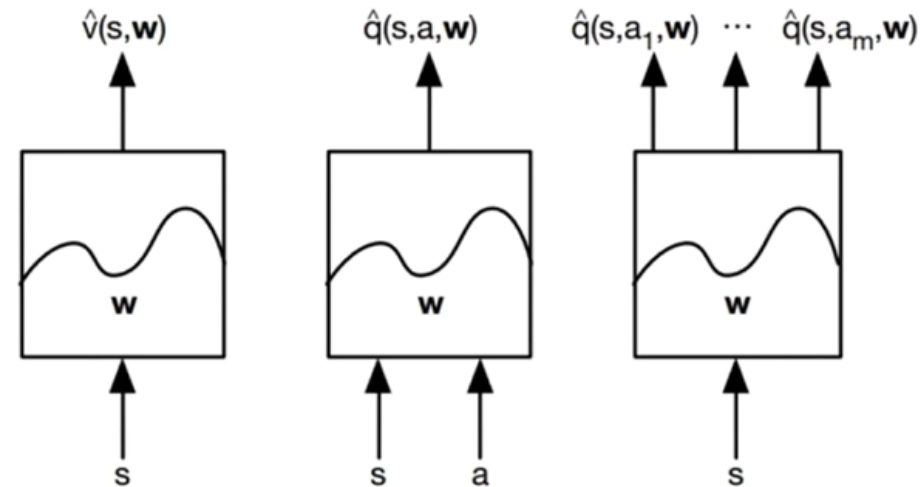
Deep QL Pros and Cons

- Pros:
  - Compresses Q-table with a Neural Network
  - Can handle states not seen during training
- Cons
  - Deterministic: cannot learn stochastic policies
  - Cannot be directly applied to continuous action spaces (need to discretize)
  - Need to separately add ϵ-greedy algorithm to balance exploration vs. exploitation.

DQN Extensions

- Experience replay
  - Save transitions $(S_t, A_t, R_{t+1}, S_{t+1})$ to buffer
  - Randomly sample from replay buffer and apply Q update
- Target network
  - Use a separate Q-network to estimate TD-target
  - Target network is synced infrequently with main network
  - Reduce correlation between Q-value and TD-target
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q^{target}(S_{t+1}, a) - Q(S_t, A_t))$