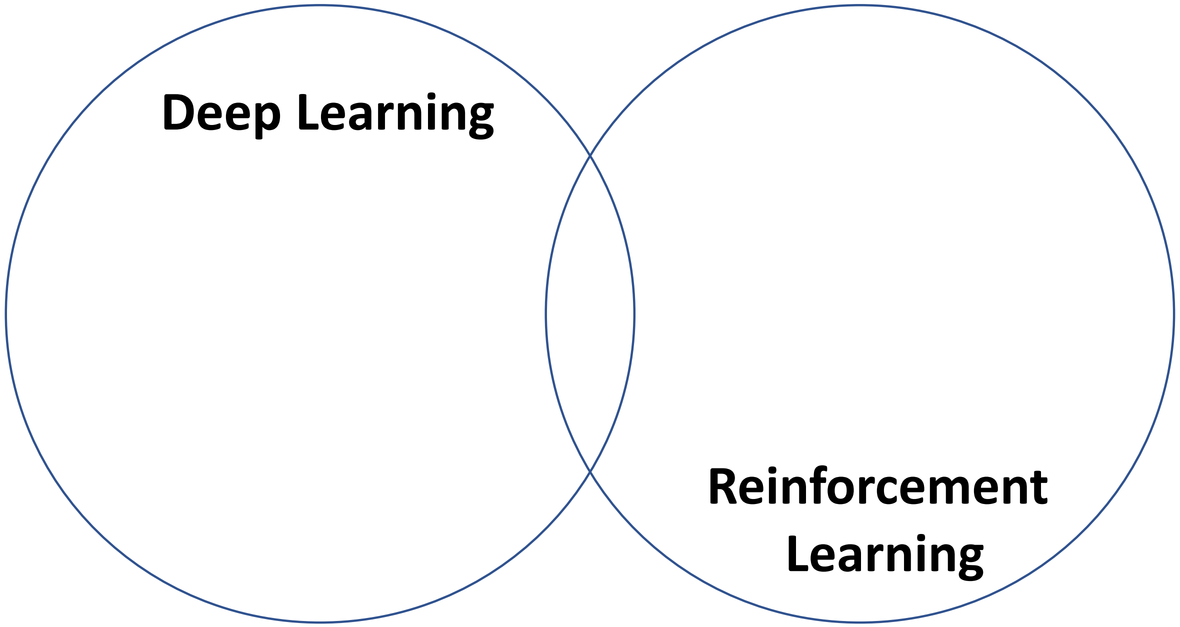
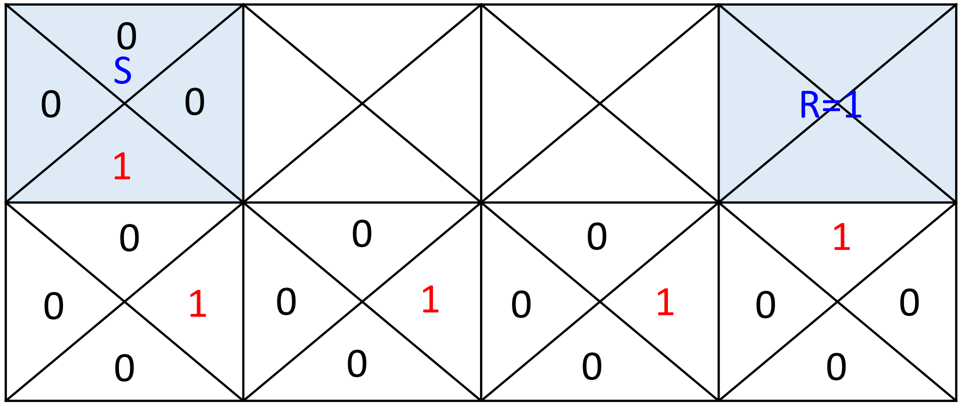
Reinforcement Learning

Reinforcement Learning



Q-Learning

Q-Learning: Greedy action in random direction of Q-map



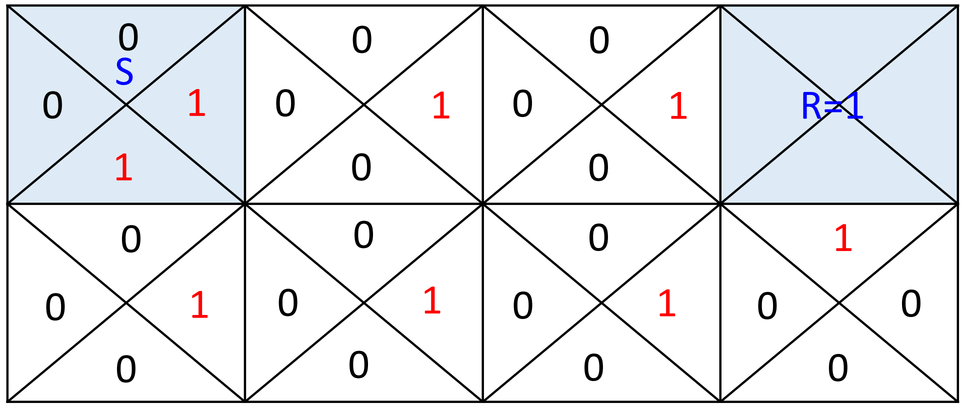
Q-Learning: Exploration

- Exploitation
  - ✓Greedy action using Q-map

- ε-Greedy
  - ✓ε(0,1): random selection in direction + Greedy action

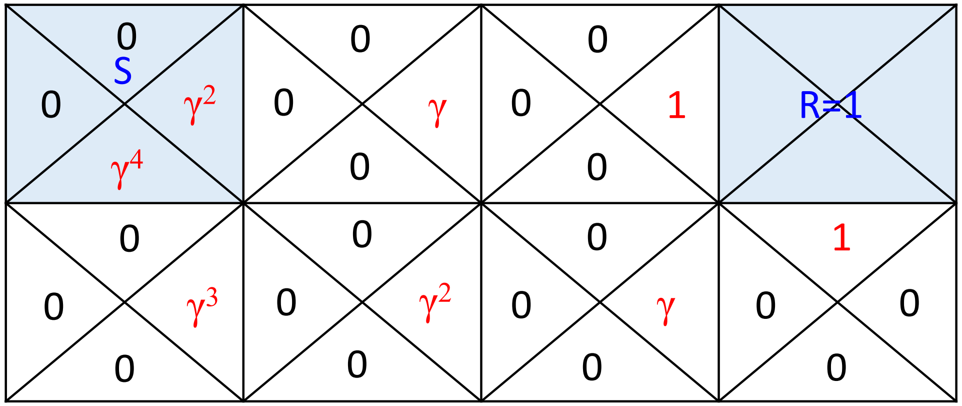
- Exploration
  - ✓Find new paths and goals

- (Decaying)ε-Greedy
  - ✓Variable ε (e.g., 0.9 → 0)



Q-Learning: Discount factor

- Discount factor: γ (0, 1)

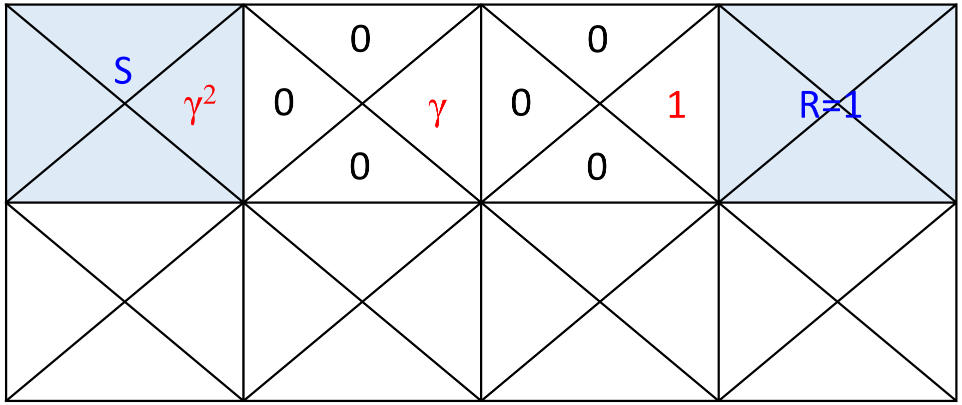


✓ Find efficient paths

✓ Meaning: reward for future state

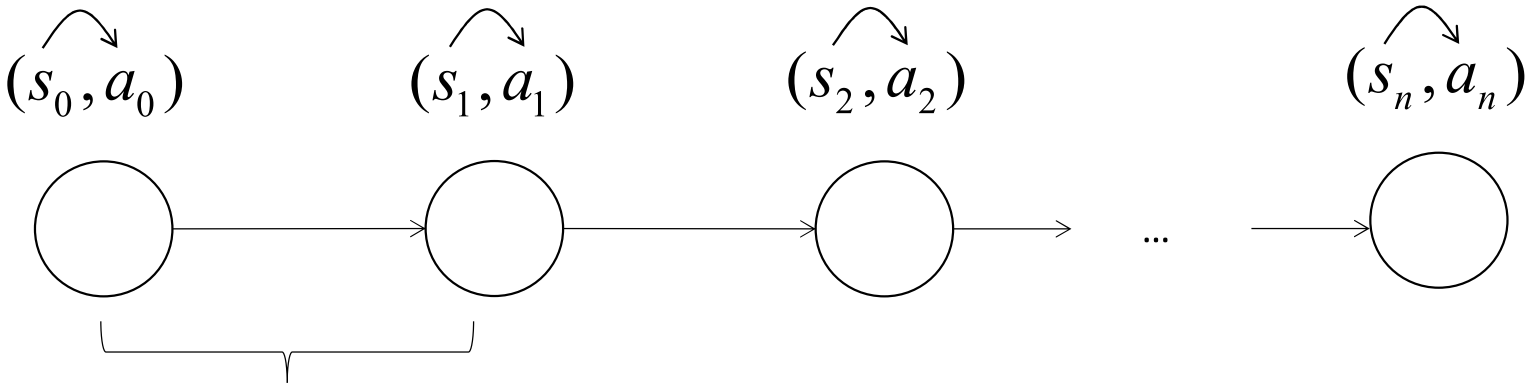
Q-Learning: Q-update

- $Q(s_t, a_t)$



$$Q(s_t,a_t) \leftarrow (1-\alpha)\underbrace{Q(s_t,a_t)}_{\text{Existing value}} + \alpha\left(\underbrace{\overbrace{R_t}^{\text{Reward by } a_t} + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1})}_{\text{Newly updated value}}\right)$$

Markov Decision Process (MDP)



$P(a_1 \mid s_0, a_0, s_1)$ *Prob. of action at state s1 (Policy)*

s$_1$ has the information of s$_0$ and a$_0$ $= P(a_1 \mid s_1)$

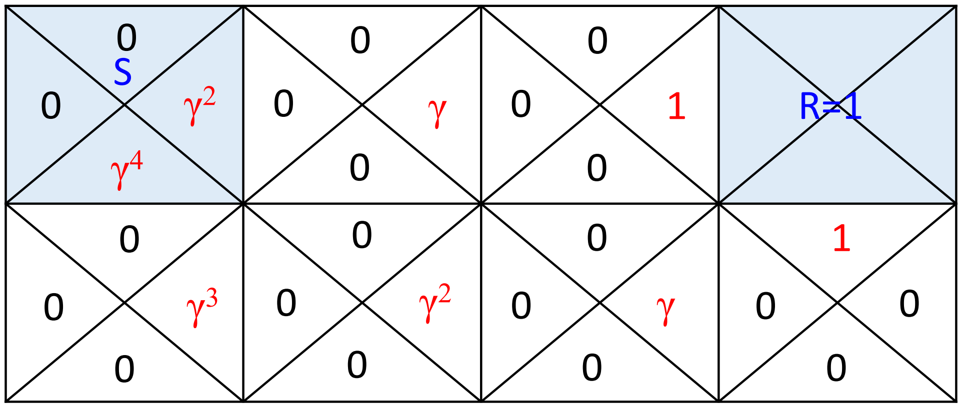
$$P(s_2 \mid s_0, a_0, s_1, a_1) = P(s_2 \mid s_1, a_1)$$

Prob. of state transition from s1 to s2

Markov Decision Process (MDP)

- Goal of Reinforcement Learning
  - ✓ Maximize expected reward or return
    - ➢ Reward: result in action a_t

$$G_t \cong R_t + \gamma R_{t+1} + r^2 R_{t+2} + \cdots$$



Markov Decision Process (MDP)

- State value function:
  - ✓ Expected return value in current state

- Action value function:
  - ✓ Expected return from current action

- Optimal policy
  - ✓ Maximize state value function

Markov Decision Process (MDP)

- State value function $V(s_t)$

$$G_t \cong R_t + \gamma R_{t+1} + r^2 R_{t+2} + \cdots$$

Expected Return

$$\Longrightarrow \quad V(s_t) \cong \int_{a_t : a_\infty} G_t P(a_t, s_{t+1}, a_{t+1}, \cdots \mid s_t) da_t : a_\infty$$

For whole actions and states

Expected value for whole actions and states in S_t

$$*E[f(x)] = \int f(x) p(x) dx$$

Markov Decision Process (MDP)

- Action value function $Q(s_t, a_t)$

$$G_t \cong R_t + \gamma R_{t+1} + r^2 R_{t+2} + \cdots$$

Expected Return

$$\Rightarrow \quad Q(s_t, a_t) \cong \int_{s_{t+1}:a_\infty} G_t P(s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}, \cdots \mid s_t, a_t) ds_{t+1} : a_\infty$$

For whole actions and states

Expected value for whole actions and states from a_t in S_t

Markov Decision Process (MDP)

- Optimal policy: Maximize $V(s_t)$

$$V(s_t) \cong \int_{a_t : a_\infty} G_t P(a_t, s_{t+1}, a_{t+1}, \cdots | s_t) da_t : a_\infty$$

$$\text{Optimal policy} \begin{cases} P(a_t \mid s_t) \\ P(a_{t+1} \mid s_{t+1}) \\ \vdots \\ P(a_\infty \mid s_\infty) \end{cases}$$

Bellman Equation

- Use Bayesian rule $P(x,y) = P(x \mid y)P(y)$

$$P(x,y \mid z) = P(x \mid y,z)P(y \mid z)$$

$$V(s_t) \cong \int_{a_t:a_\infty} G_t \underline{P(a_t, s_{t+1}, a_{t+1}, \cdots \mid s_t)} da_t : a_\infty$$

$$\Rightarrow P(s_{t+1}, a_{t+1}, \cdots \mid s_t, a_t) P(a_t \mid s_t)$$

$$V(s_t) \cong \int_{a_t:a_\infty} G_t P(s_{t+1}, a_{t+1}, \cdots \mid s_t, a_t) P(a_t \mid s_t) da_t : a_\infty$$

$$= \int_{a_t} \boxed{\int_{s_{t+1}:a_\infty} G_t P(s_{t+1}, a_{t+1}, \cdots \mid s_t, a_t) ds_{t+1} : a_\infty} P(a_t \mid s_t) da_t$$

$$= \int_{a_t} Q(s_t, a_t) P(a_t \mid s_t) da_t$$

State value is the mean Q-values at the given state

Bellman Equation

$$V(s_t) \cong \int_{a_t:a_\infty} G_t P(a_t, s_{t+1}, a_{t+1}, \cdots \mid s_t) da_t : a_\infty$$

$$\Rightarrow P(s_{t+1}, a_{t+1}, \cdots \mid s_t, a_t) P(a_t \mid s_t)$$

$$\Rightarrow P(a_{t+1}, \cdots \mid \cancel{s_t}, \cancel{a_t}, s_{t+1}) P(a_t, s_{t+1} \mid s_t)$$

$$= P(a_{t+1}, \cdots \mid s_{t+1}) P(a_t, s_{t+1} \mid s_t)$$

Bellman Equation

$$G_t \cong R_t + \gamma R_{t+1} + r^2 R_{t+2} + \cdots = R_t + \gamma G_{t+1}$$

$$V(s_t) \cong \int_{a_t, s_{t+1}} \boxed{\int_{a_{t+1}:a_\infty}} \left(R_t + \gamma \boxed{G_{t+1}\right) P\left(a_{t+1}, \cdots \mid s_{t+1}\right) da_{t+1} : a_\infty} P\left(a_t, s_{t+1} \mid s_t\right) da_t, s_{t+1}$$

$$= \int_{a_t, s_{t+1}} \left(R_t + \gamma V(S_{t+1})\right) \underline{P\left(a_t, s_{t+1} \mid s_t\right)} da_t, s_{t+1}$$

$$= \int_{a_t, s_{t+1}} \left(R_t + \gamma V(S_{t+1})\right) \underbrace{P\left(s_{t+1} \mid s_t, a_t\right)}_{\text{Transition}} \underbrace{P(a_t \mid s_t)}_{\text{Policy}} da_t, s_{t+1} \Longrightarrow \text{Bellman eq.}$$

Bellman Equation

$$Q(s_t, a_t) \cong \int_{s_{t+1}:a_\infty} G_t \underline{P(s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}, \cdots \mid s_t, a_t)} ds_{t+1} : a_\infty$$

$$\Rightarrow P(a_{t+1}, s_{t+2}, a_{t+2}, \cdots \mid \cancel{s_t}, \cancel{a_t}, s_{t+1}) P(s_{t+1} \mid s_t, a_t)$$

$$= P(a_{t+1}, s_{t+2}, a_{t+2}, \cdots \mid s_{t+1}) P(s_{t+1} \mid s_t, a_t)$$

$$G_t \cong R_t + \gamma R_{t+1} + r^2 R_{t+2} + \cdots = R_t + \gamma G_{t+1}$$

$$Q(s_t, a_t) \cong \int_{s_{t+1}} \boxed{\int_{a_{t+1}:a_\infty}} \left(R_t + \gamma \boxed{G_{t+1}\right) P(a_{t+1}, s_{t+2}, a_{t+2}, \cdots \mid s_{t+1}) da_{t+1} : a_\infty} P\left(s_{t+1} \mid s_t, a_t\right) ds_{t+1}$$

$$= \int_{s_{t+1}} \left(R_t + \gamma V(s_{t+1})\right) P\left(s_{t+1} \mid s_t, a_t\right) ds_{t+1}$$

Bellman Equation

$$Q(s_t, a_t) \cong \int_{s_{t+1}:a_\infty} G_t \underline{P(a_{t+1}, s_{t+2}, a_{t+2}, \cdots | s_t, a_t, s_{t+1}) P(s_{t+1} | s_t, a_t)} ds_{t+1}: a_\infty$$

$$\Rightarrow P(s_{t+2}, a_{t+2}, \cdots | s_{t+1}, a_{t+1}) P(s_{t+1}, a_{t+1} | s_t, a_t)$$

$$Q(s_t, a_t) \cong \int_{s_{t+1}, a_{t+1}} \boxed{\int_{s_{t+2}:a_\infty}} (R_t + \gamma \boxed{G_{t+1}) P(s_{t+2}, a_{t+2}, \cdots | s_{t+1}, a_{t+1}) ds_{t+2}: a_\infty} P(s_{t+1}, a_{t+1} | s_t, a_t) ds_{t+1}, a_{t+1}$$

$$= \int_{s_{t+1}, a_{t+1}} (R_t + \gamma Q(s_{t+1}, a_{t+1})) \underline{P(s_{t+1}, a_{t+1} | s_t, a_t)} ds_{t+1}, a_{t+1}$$

$$= \int_{s_{t+1}, a_{t+1}} (R_t + \gamma Q(s_{t+1}, a_{t+1})) P(a_{t+1} | s_t, a_t, s_{t+1}) P(s_{t+1} | s_t, a_t) ds_{t+1}, a_{t+1}$$

$$= \int_{s_{t+1}, a_{t+1}} (R_t + \gamma Q(s_{t+1}, a_{t+1})) \underbrace{P(a_{t+1} | s_{t+1})}_{\text{Policy}} \underbrace{P(s_{t+1} | s_t, a_t)}_{\text{Transition}} ds_{t+1}, a_{t+1} \Rightarrow \text{Bellman eq.}$$

Optimal Policy

- Maximize the expected return at the current state
- Maximize the state value function

$$V(s_t) \cong \int_{a_t:a_\infty} G_t P(a_t, s_{t+1}, a_{t+1}, \cdots \mid s_t) da_t : a_\infty$$

$$= \int_{a_t} Q(s_t, a_t) P(a_t \mid s_t) da_t$$

$$\underset{P(a_t|s_t)}{\arg\max} \int_{a_t} Q(s_t, a_t) P(a_t \mid s_t) da_t \Longrightarrow \int_{a_t} \underline{Q^*(s_t, a_t)} P(a_t \mid s_t) da_t$$

$a_t^* \cong \arg\max_{a_t} Q^*(s_t, a_t)$ ⇨ Greedy policy ⟹ ε-Greedy for random direction

$P^*(a_t \mid s_t) = \delta(a_t - a_t^*)$ Find a_t to maximize Q^* ⇨ Choose a_t

Monte-Carlo Method

- How to obtain Q*?
  - ✓ Training means making Q*

- Expectation value $E[x] = \boxed{\int_x xP(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} x_i}$

$$Q(s_t, a_t) \cong \int_{s_{t+1}:a_\infty} G_t P(s_{t+1}:a_\infty \mid s_t, a_t) ds_{t+1}:a_\infty = \frac{1}{N}\sum_{i=1}^{N} G_t^{(i)}$$

For many cases *N*, obtain *Q* and find a_t

Temporal Difference (TD)

- Action value function

$$Q(s_t,a_t) \cong \int_{s_{t+1},a_{t+1}} (R_t + \gamma Q(s_{t+1},a_{t+1}))P(s_{t+1} \mid s_t,a_t)P(a_{t+1} \mid s_{t+1})ds_{t+1},a_{t+1}$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}(R_t^{(i)} + \gamma Q(s_{s+1}^{(i)},a_{t+1}^{(i)})) \cong \overline{Q}_N$$

- 1-step TD (Incremental Mete-Carlo Updates)

$$\overline{Q}_N = \frac{1}{N}\left(\overline{Q}_{N-1}(N-1) + R_t^{(N)} + \gamma Q(s_{t+1}^{(N)},a_{t+1}^{(N)})\right)$$

$$= \underline{\overline{Q}_{N-1}} + \frac{1}{N}\left(R_t^{(N)} + \gamma Q(s_{t+1}^{(N)},a_{t+1}^{(N)}) - \underline{\overline{Q}_{N-1}}\right)$$

Incremental Monte-Carlo Updates

- Incremental Mete-Carlo Updates

$$\overline{Q}_N = \frac{1}{N}\left(\overline{Q}_{N-1}(N-1) + R_t^{(N)} + \gamma Q(s_{t+1}^{(N)}, a_{t+1}^{(N)})\right)$$

$$= \overline{Q}_{N-1} + \boxed{\frac{1}{N}}\left(R_t^{(N)} + \gamma Q(s_{t+1}^{(N)}, a_{t+1}^{(N)}) - \overline{Q}_{N-1}\right)$$

$$= (1-\alpha)\overline{Q}_{N-1} + \alpha\left(\underbrace{R_t^{(N)} + \gamma Q(s_{t+1}^{(N)}, a_{t+1}^{(N)})}_{\text{Current sample (i.e., TD target)}}\right)$$

- SARSA algorithm

Reward

$$\overline{Q}_N = (1-\alpha)\overline{Q}_{N-1} + \alpha\left(R_t^{(N)} + \gamma Q(s_{t+1}^{(N)}, a_{t+1}^{(N)})\right)$$

Current State & Action

Next State

Next Action

Temporal Difference (TD)

- SARSA
  - On-Policy
    - Behavior policy == Target policy

- Q-Learning
  - Off-Policy
    - Behavior policy != Target policy

Behavior policy & Target policy

- Behavior policy & Target policy

Given by Behavior policy
$P(a_t | s_t)$

$$Q(s_t, a_t) \cong \int_{s_{t+1}, a_{t+1}} (R_t + \gamma Q(s_{t+1}, a_{t+1}))\overbrace{P(a_{t+1} \mid s_{t+1})}^{\text{Target policy}} P(s_{t+1} \mid s_t, a_t) ds_{t+1}, a_{t+1}$$

$$\approx \overline{Q}_N = (1-\alpha)\overline{Q}_{N-1} + \alpha\left(\underbrace{R_t^{(N)} + \gamma Q(s_{t+1}^{(N)}, a_{t+1}^{(N)})}_{\substack{\text{TD target} \\ \text{(i.e., sample)}}}\right)$$

Off-Policy

- Divide Target policy and Behavior policy
  - ✓ Why?
    - ➢ Can use other target policy by human or other agents
    - ➢ Can obtain a sample (i.e., TD target) by the optimal policy (i.e., greedy action) during exploring
    - ➢ Re-judgement (i.e., re-use) for actions is possible

Behavior policy: $p(a_t|s_t)$

$$Q(s_t,a_t) \cong \int_{s_{t+1},a_{t+1}} (R_t + \gamma Q(s_{t+1},a_{t+1}))\underbrace{P(a_{t+1}\mid s_{t+1})}_{\text{Target policy}} P(s_{t+1}\mid s_t,a_t)\,ds_{t+1},a_{t+1}$$

$$\approx \overline{Q}_N = (1-\alpha)\overline{Q}_{N-1} + \alpha\left(R_t^{(N)} + \gamma Q(s_{t+1}^{(N)},a_{t+1}^{(N)})\right)$$

Q-Learning (revisit)

- Target policy: greedy
- Behavior policy: ε-greedy

$$Q(s_t, a_t) \cong \int_{s_{t+1}, a_{t+1}} (R_t + \gamma Q(s_{t+1}, a_{t+1})) P(a_{t+1} \mid s_{t+1}) P(s_{t+1} \mid s_t, a_t) ds_{t+1}, a_{t+1}$$

$$\approx \overline{Q}_N = (1-\alpha)\overline{Q}_{N-1} + \alpha\left(R_t^{(N)} + \gamma Q(s_{t+1}^{(N)}, a_{t+1}^{(N)})\right)$$

$$\text{Target}: P(a_{t+1} \mid s_{t+1}) = \delta(a_{t+1} - a_{t+1}^{*}), \quad a_{t+1}^{*} = \arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

$$\text{Behavior}: \int_{s_{t+1}, a_{t+1}} (R_t + \gamma Q(s_{t+1}, a_{t+1})) \delta(a_{t+1} - a_{t+1}^{*}) P(s_{t+1} \mid s_t, a_t) ds_{t+1}, a_{t+1}$$

$$= \int_{s_{t+1}} (R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})) P(s_{t+1} \mid s_t, a_t) ds_{t+1}$$

Q-Learning (revisit)

$$\int_{s_{t+1},a_{t+1}} (R_t + \gamma Q(s_{t+1}, a_{t+1}))\delta(a_{t+1} - a^*_{t+1})P(s_{t+1} \mid s_t, a_t)ds_{t+1}, a_{t+1}$$

$$= \int_{s_{t+1}} (R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}))P(s_{t+1} \mid s_t, a_t)ds_{t+1}$$

$$\Rightarrow R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \text{ : TD-target}$$

2Step-TD

- 1-step TD

$$Q(s_t,a_t) \cong \int_{s_{t+1},a_{t+1}} (R_t + \gamma Q(s_{t+1},a_{t+1}))P(a_{t+1} \mid s_{t+1})P(s_{t+1} \mid s_t,a_t)ds_{t+1},a_{t+1}$$

$$\approx \overline{Q}_N = (1-\alpha)\overline{Q}_{N-1} + \alpha\left(R_t^{(N)} + \gamma Q(s_{t+1}^{(N)},a_{t+1}^{(N)})\right)$$

- 2-step TD

$$Q(s_t,a_t) \cong$$

$$\int_{s_{t+1},a_{t+1},s_{t+2},a_{t+2}} (R_t + \gamma R_t + \gamma^2 Q(s_{t+2},a_{t+2}))P(a_{t+2} \mid s_{t+2})P(s_{t+2} \mid s_{t+1},a_{t+1})P(a_{t+1} \mid s_{t+1})P(s_{t+1} \mid s_t,a_t)ds_{t+1},a_{t+1},s_{t+2},a_{t+2}$$

2Step-TD

- Off-policy with importance sampling

$$\int_x xp(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} x_i \qquad x_i \sim p(x)$$

$$= \int_x x\frac{p(x)}{q(x)}q(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} x_i \boxed{\frac{p(x_i)}{q(x_i)}} \qquad x_i \sim q(x)$$

Compensation value ← importance rate

2Step-TD

- TD-target in 2-step TD

$$Q(s_t,a_t) \cong \int_{s_{t+1},a_{t+1},s_{t+2},a_{t+2}} (R_t + \gamma R_t + \gamma^2 Q(s_{t+2},a_{t+2}))P(a_{t+2} \mid s_{t+2})P(s_{t+2} \mid s_{t+1},a_{t+1})\underbrace{P(a_{t+1} \mid s_{t+1})}_{q(a_{t+1} \mid s_{t+1})}P(s_{t+1} \mid s_t,a_t)ds_{t+1},a_{t+1},s_{t+2},a_{t+2}$$

$q(a_{t+1} \mid s_{t+1})$

Not Use : We can obtain similar results

$$\text{TD-target}: \cancel{\frac{p(a_{t+1}^{(N)} \mid s_{t+1})}{q(a_{t+1}^{(N)} \mid s_{t+1})}}\left(R_t^{(N)} + \gamma R_{t+1}^{(N)} + \gamma^2 \max_{a_{t+2}} Q(s_{t+2}^{(N)},a_{t+2})\right)$$

$a_t^{(N)} \sim q$ $\quad a_{t+1}^{(N)} \sim q$ $\quad a_{t+2}^{(N)} \sim p$

Q-learning Code

```
import numpy as np

n_states = 12  # Number of states in the grid world
n_actions = 4  # Number of possible actions (up, down, left, right)
goal_state = 11  # Goal state

Q_table = np.zeros((n_states, n_actions))

learning_rate = 0.8
discount_factor = 0.95
exploration_prob = 0.2
epochs = 100
```

Q-learning Code

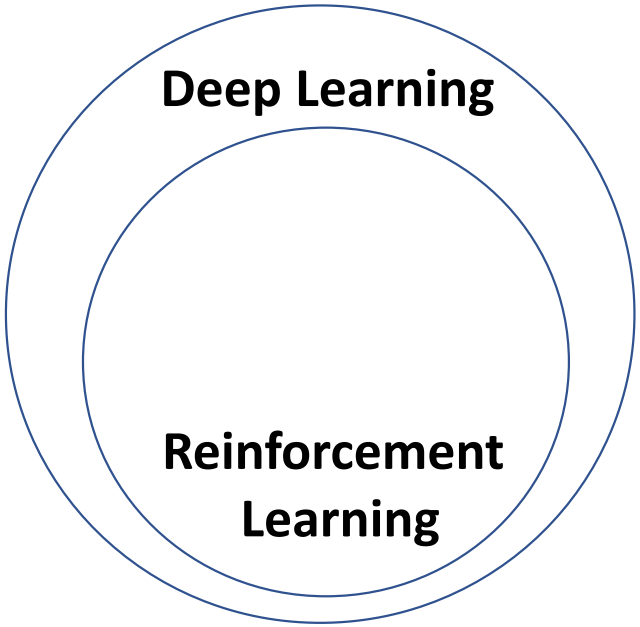
```
for epoch in range(epochs):
    current_state = np.random.randint(0, n_states)  # Start from a random state

    while current_state != goal_state:
        # Choose action with epsilon-greedy strategy
        if np.random.rand() < exploration_prob:
            action = np.random.randint(0, n_actions)  # Explore
        else:
            action = np.argmax(Q_table[current_state])  # Exploit

        # Simulate the environment (move to the next state)
        # For simplicity, move to the next state
        next_state = (current_state + 1) % n_states

        # Define a simple reward function (1 if the goal state is reached, 0 otherwise)
        reward = 1 if next_state == goal_state else 0

        # Update Q-value using the Q-learning update rule
        Q_table[current_state, action] += learning_rate * (reward +
                        discount_factor * np.max(Q_table[next_state]) - Q_table[current_state, action])

        current_state = next_state  # Move to the next state

# After training, the Q-table represents the learned Q-values
print("Learned Q-table:")
print(Q_table)
```

Q-learning Results

```
Learned Q-table:
[[0.37616543 0.47823792 0.59873694 0.        ]
 [0.63024941 0.         0.60503942 0.        ]
 [0.66342043 0.53073304 0.65809562 0.50961706]
 [0.6983373  0.55866984 0.6927506  0.6927506 ]
 [0.73509189 0.72921116 0.73385176 0.7292102 ]
 [0.77378094 0.7428297  0.77251646 0.76759069]
 [0.81450625 0.80799003 0.81449582 0.781926  ]
 [0.857375   0.85710064 0.85595476 0.85736402]
 [0.9025     0.90244224 0.90244224 0.901056  ]
 [0.94999749 0.94998776 0.95       0.95      ]
 [1.         0.99999949 0.99999949 0.99999998]
 [0.         0.         0.         0.        ]]
```

Deep Q Network (DQN)

Q-Learning +DNN



Q-Learning

$$Q(s_t, a_t) \cong \int_{s_{t+1}, a_{t+1}} (R_t + \gamma Q(s_{t+1}, a_{t+1})) P(a_{t+1} \mid s_{t+1}) P(s_{t+1} \mid s_t, a_t) ds_{t+1}, a_{t+1}$$

Target policy Transition

$$P(a_{t+1} \mid s_{t+1}) = \delta(a_{t+1} - a^*_{t+1}), \quad a^*_{t+1} = \arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

$$= \int_{s_{t+1}} (R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})) P(s_{t+1} \mid s_t, a_t) ds_{t+1}$$

TD target

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

Q-Learning

- Bellman equation as an iterative update

$$Q(s_t, a_t) \leftarrow \mathrm{E}_{s_{t+1} \sim P(s_{t+1}|s_t, a_t)} \left(R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \right)$$

- Q will converge to Q*

$$\mathrm{target}(s_{t+1}) = R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(\mathrm{target}(s_{t+1}) - Q(s_t, a_t) \right)$$

  ✓ Q-learning converges to optimal policy if all state-action pairs seen frequently enough

- ε-Greedy as Behavior policy

Problem in Q-learning

- Not scalable
  - ✓ Must compute Q-table for every state-action pair
- Use a function approximator to estimate Q(s,a)
  - ✓ Learn about some small number of training states from experience
  - ✓ Generalize that experience to new, similar situations

Deep Q Network (DQN)

- Regression with DNN
  - ✓ Find Q using Regression
    - ➢ Why?
    - ➢ To apply various states

  - ✓ Q_W : Q with weight W; By regression

$$\vec{w} \leftarrow \vec{w} - \boxed{\alpha} Grad$$

Learning rate

Difference
among samples

Approximate Q-Learning

$$\text{target}(s_{t+1}) = R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(\text{target}(s_{t+1}) - Q(s_t, a_t)\right)$$

- Linear function approximator (i.e., Regression)

$$Q_w(s,a) = w_0 f_0(s,a) + w_1 f_1(s,a) + \cdots + w_n f_n(s,a)$$

- Exact Q

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[difference\right]$$

- Approximate Q

$$w_i \leftarrow w_i + \alpha\left[difference\right] f_i(s,a)$$

Linear function approximator

- Approximate Q $$w_i \leftarrow w_i + \alpha\left[difference\right] f_i(s,a)$$

- Loss function $$loss = \frac{1}{2}(y - \hat{y})^2$$

- Difference

$$y - \hat{y} = y - \left(w_0 f_0(s,a) + w_1 f_1(s,a) + \cdots + w_n f_n(s,a)\right) = y - \sum_i w_i f_i(s,a)$$

$$\frac{\partial l(w)}{\partial w_k} = -\left(y - \sum_i w_i f_i(s,a)\right) f_k(s,a)$$

$$w_k \leftarrow w_i + \alpha\left(y - \sum_i w_i f_i(s,a)\right) f_k(s,a)$$

$$w_k \leftarrow w_i + \alpha\left[difference\right] f_k(s,a)$$

Deep Q Network (DQN)

- Difference

$$difference = \left[R(s_t) + \gamma \max_{a_{t+1}} Q_w(s_{t+1}, a_{t+1}) \right] - Q_w(s_t, a_t)$$

- Loss function

$$l(w) = \left[\frac{1}{2} \left(\mathrm{target}(s_{t+1}) - Q_w(s_t, a_t) \right)^2 \right]$$

- TD update

$$\begin{aligned} w_{k+1} &\leftarrow w_k - \alpha \nabla_w l(w) \\ &\leftarrow w_k - \alpha \nabla_w \left[\frac{1}{2} \left(\mathrm{target}(s_{t+1}) - Q_w(s_t, a_t) \right)^2 \right] \\ &\leftarrow w_k - \alpha \left[\mathrm{target}(s_{t+1}) - Q_w(s_t, a_t) \right] \nabla_w Q_w(s_t, a_t) \end{aligned}$$

Deep Q Network 2013

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Deep Q Network 2015

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

DQN Code

```
import gym
import collections
import random

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

learning_rate = 0.0005
gamma = 0.9
buffer_limit = 10000
batch_size = 64

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

DQN Code

```python
class ReplayBuffer():
    def __init__(self):
        self.buffer = collections.deque(maxlen=buffer_limit)
    def put(self, transition):
        self.buffer.append(transition)
    def sample(self, n):
        mini_batch = random.sample(self.buffer, n)
        s_list, a_list, r_list, s_prime_list, done_mask_list = [], [], [], [], []

        for transition in mini_batch:
            s, a, r, s_prime, done_mask = transition
            s_list.append(s)
            a_list.append([a])
            r_list.append([r])
            s_prime_list.append(s_prime)
            done_mask_list.append([done_mask])

        return torch.tensor(s_list, dtype=torch.float), torch.tensor(a_list), torch.tensor(r_list),\
                          torch.tensor(s_prime_list, dtype=torch.float), torch.tensor(done_mask_list)
    def size(self):
        return len(self.buffer)
```

DQN Code

```python
class Qnet(nn.Module):
    def __init__(self):
        super(Qnet, self).__init__()
        self.fc1 = nn.Linear(4, 256)
        self.fc2 = nn.Linear(256, 2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

    def sample_action(self, obs, epsilon):
        out = self.forward(obs)
        prob = random.random()
        if prob < epsilon:
            return random.randint(0,1)
        else:
            return out.argmax().item()
```

DQN Code

```
def train(q, q_target, memory, optimizer):
    for i in range(10):
        s, a, r, s_prime, done_mask = memory.sample(batch_size)

        q_out = q(s)
        q_a = q_out.gather(1,a)
        max_q_prime = q(s_prime).max(1)[0].unsqueeze(1)
        target = r + gamma * max_q_prime * done_mask
        loss = F.smooth_l1_loss(q_a, target)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```python
def main():
    env = gym.make('CartPole-v1')
    q = Qnet()
    q_target = Qnet()
    q_target.load_state_dict(q.state_dict())
    memory = ReplayBuffer()

    print_interval = 20
    score = 0.0
    optimizer = optim.Adam(q.parameters(), lr=learning_rate)

    for n_epi in range(1000):
        epsilon = max(0.01, 0.08 - 0.01*(n_epi/200)) # Linear annealing form 8% to 1%
        s, _ = env.reset()
        done = False

        while not done:
            a = q.sample_action(torch.from_numpy(s).float(), epsilon)
            s_prime, r, done, truncated, info = env.step(a)
            done_mask = 0.0 if done else 1.0
            memory.put((s, a, r, s_prime, done_mask))
            s = s_prime
            score += r
            if done:
                break

        if memory.size() > 2000:
            train(q, q_target, memory, optimizer)

        if n_epi%print_interval==0 and n_epi!=0:
            q_target.load_state_dict(q.state_dict())
            print("n_episode :{}, score : {:.1f}, n_buffer : {}, eps : {:.1f}%".format(
                                                            n_epi, score/print_interval, memory.size(), epsilon*100))
            score = 0.0
```

DQN - Results

```
n_episode :500, score : 58.4, n_buffer : 10000, eps : 5.5%
n_episode :520, score : 81.7, n_buffer : 10000, eps : 5.4%
n_episode :540, score : 93.5, n_buffer : 10000, eps : 5.3%
n_episode :560, score : 98.2, n_buffer : 10000, eps : 5.2%
n_episode :580, score : 94.5, n_buffer : 10000, eps : 5.1%
n_episode :600, score : 83.9, n_buffer : 10000, eps : 5.0%
n_episode :620, score : 86.5, n_buffer : 10000, eps : 4.9%
n_episode :640, score : 83.2, n_buffer : 10000, eps : 4.8%
n_episode :660, score : 85.5, n_buffer : 10000, eps : 4.7%
n_episode :680, score : 102.3, n_buffer : 10000, eps : 4.6%
n_episode :700, score : 125.3, n_buffer : 10000, eps : 4.5%
n_episode :720, score : 128.8, n_buffer : 10000, eps : 4.4%
n_episode :740, score : 179.3, n_buffer : 10000, eps : 4.3%
n_episode :760, score : 162.4, n_buffer : 10000, eps : 4.2%
n_episode :780, score : 168.2, n_buffer : 10000, eps : 4.1%
n_episode :800, score : 135.7, n_buffer : 10000, eps : 4.0%
n_episode :820, score : 133.3, n_buffer : 10000, eps : 3.9%
n_episode :840, score : 134.6, n_buffer : 10000, eps : 3.8%
n_episode :860, score : 145.1, n_buffer : 10000, eps : 3.7%
n_episode :880, score : 170.6, n_buffer : 10000, eps : 3.6%
n_episode :900, score : 199.9, n_buffer : 10000, eps : 3.5%
n_episode :920, score : 193.4, n_buffer : 10000, eps : 3.4%
n_episode :940, score : 186.1, n_buffer : 10000, eps : 3.3%
n_episode :960, score : 169.6, n_buffer : 10000, eps : 3.2%
n_episode :980, score : 214.7, n_buffer : 10000, eps : 3.1%
Complete
```

E N D