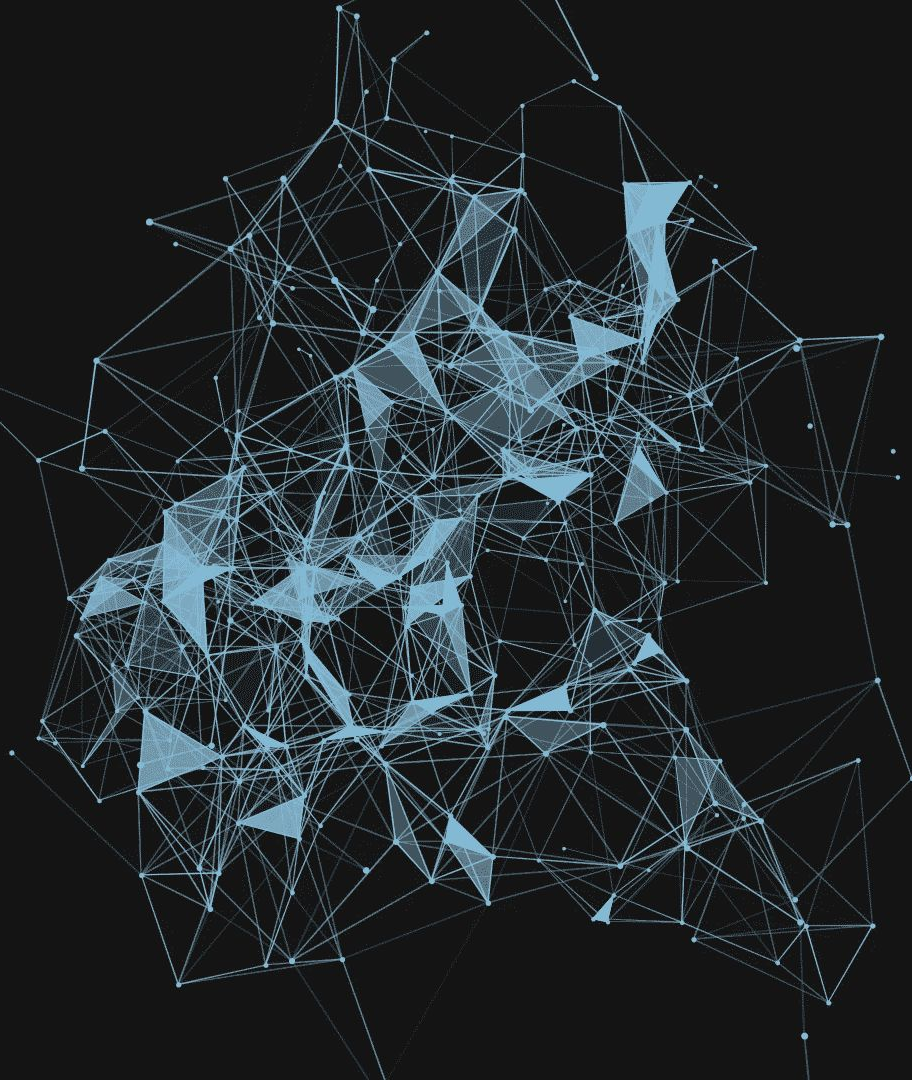


Reinforcement

Learning

Lesson - 3





RENEFENETON
LEARNING

REWARD
FUNCTION
CURVE

REWARD
FUNCTION
CURVE

QUIZ TIME

REWARD
FUNCTION

REWARD
FUNCTION
CURVE

RENEENNSION
LEARNING

QZ-2
TIME

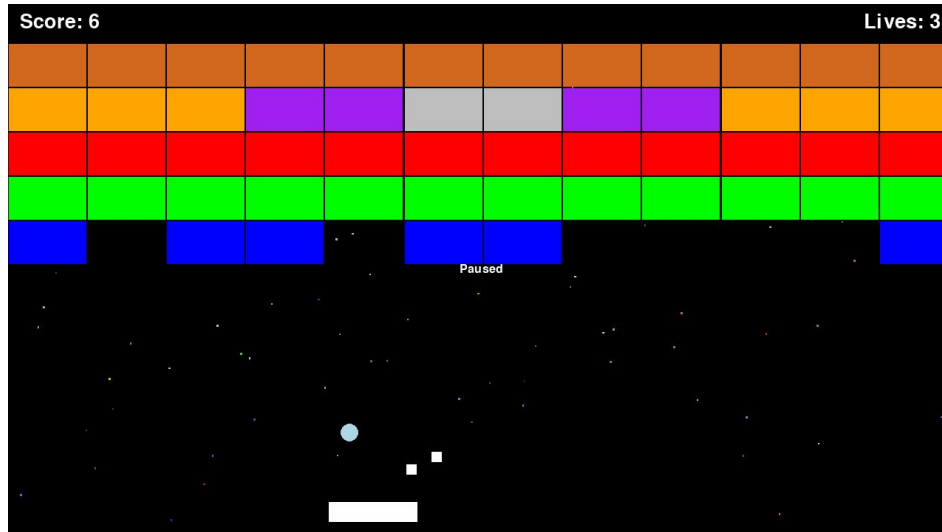
RENEFENETON
LEARNING

QUIZ
TIME

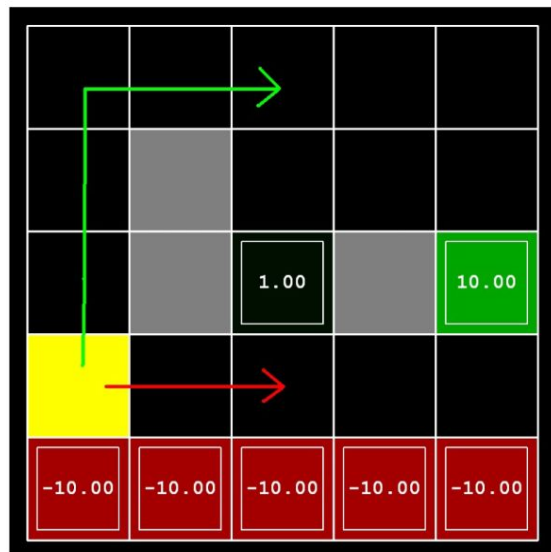
Breakout

Game Scenario:

Imagine you are playing Breakout. Your paddle is positioned near the bottom of the screen, and a ball is bouncing towards it. The ball has already destroyed a few bricks on the top right side of the wall. You have 3 lives left, and the ball is about to hit the paddle.



Exercise 1: Effect of Discount and Noise



(a) Prefer the close exit (+1), risking the cliff (-10)

(1) $\gamma = 0.1$, noise = 0.5

(b) Prefer the close exit (+1), but avoiding the cliff (-10)

(2) $\gamma = 0.99$, noise = 0

(c) Prefer the distant exit (+10), risking the cliff (-10)

(3) $\gamma = 0.99$, noise = 0.5

(d) Prefer the distant exit (+10), avoiding the cliff (-10)

(4) $\gamma = 0.1$, noise = 0

*Credit: Pieter Abbeel
foundation of deep RL*

Exercise 1 Solution

0.00 ▸	0.00 ▸	0.01 ▾	0.01 ▸	0.10 ▾
0.00 ▾		0.10 ▾	0.10 ▸	1.00 ▾
0.00 ▾		1.00 ▴		10.00 ▴
0.00 ▸	0.01 ▸	0.10 ▴	0.10 ▸	1.00 ▴
-10.00	-10.00	-10.00	-10.00	-10.00

(a) Prefer close exit (+1), risking the cliff (-10)

(4) $\gamma = 0.1$, noise = 0

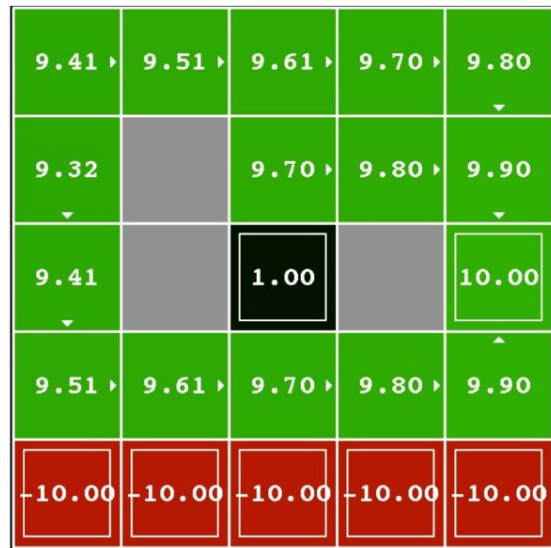
Exercise 1 Solution

0.00	0.00	0.00	0.00	0.03
0.00		0.05	0.03	0.51
0.00		1.00		10.00
0.00	0.00	0.05	0.01	0.51
-10.00	-10.00	-10.00	-10.00	-10.00

(b) Prefer close exit (+1), avoiding the cliff (-10)

(1) $\gamma = 0.1$, noise = 0.5

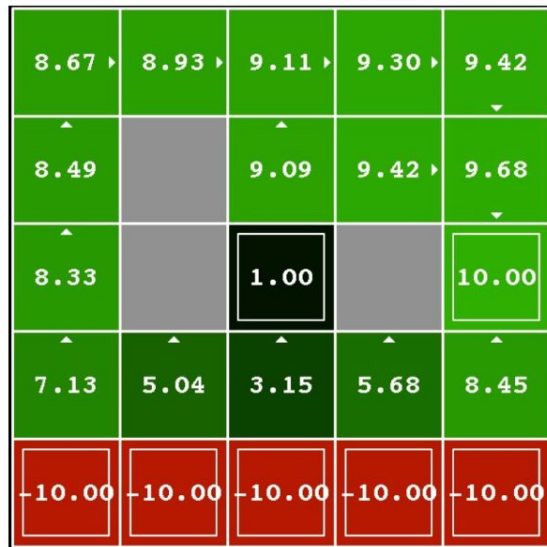
Exercise 1 Solution



(c) Prefer distant exit (+10), risking the cliff (-10)

(2) $\gamma = 0.99$, noise = 0

Exercise 1 Solution



(d) Prefer distant exit (+10), avoid the cliff (-10)

(3) $\gamma = 0.99$, noise = 0.5

Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement

policy-stable $\leftarrow true$

For each $s \in \mathcal{S}$:

old-action $\leftarrow \pi(s)$

$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow false$

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Bellman Equation



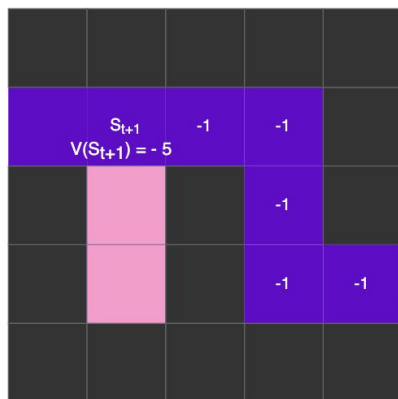
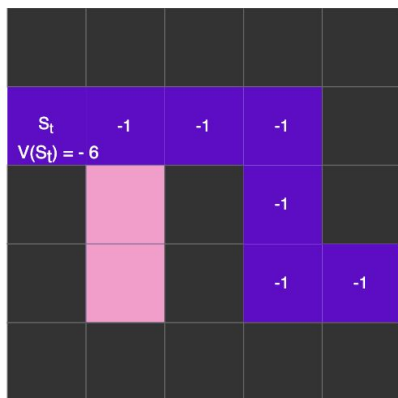
-7	-6	-5	-4	
	-7		-3	
	-8		-2	-1



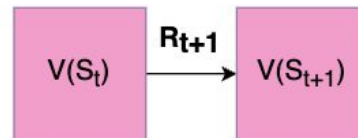
$$v_{\pi}(s) \doteq \mathbb{E}_{\pi}[G_t \mid S_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_{\pi}[G_{t+1} | S_{t+1} = s'] \right]$$

Bellman Equation



$$\begin{aligned}
 v_{\pi}(s) &= \mathbb{E}_{\pi}[G_t \mid S_t = s] \\
 &= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \\
 &= \mathbb{E}_{\pi}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s\right] \\
 &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_{\pi}[G_{t+1} | S_{t+1} = s']\right] \\
 &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_{\pi}(s')\right],
 \end{aligned}$$



$$V(S_t) = R_{t+1} + \gamma * V(S_{t+1})$$

$$V(S_t) = -1 + 1 * (-5) = -6$$

Action-Value function for policy π

Similarly, we define the value of taking action a in state s under a policy π , denoted $q_\pi(s, a)$, as the expected return starting from s , taking the action a , and thereafter following policy π :

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t=s, A_t=a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t=s, A_t=a\right]. \quad (3.13)$$

Backup Diagram

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_{\pi}(s')]$$

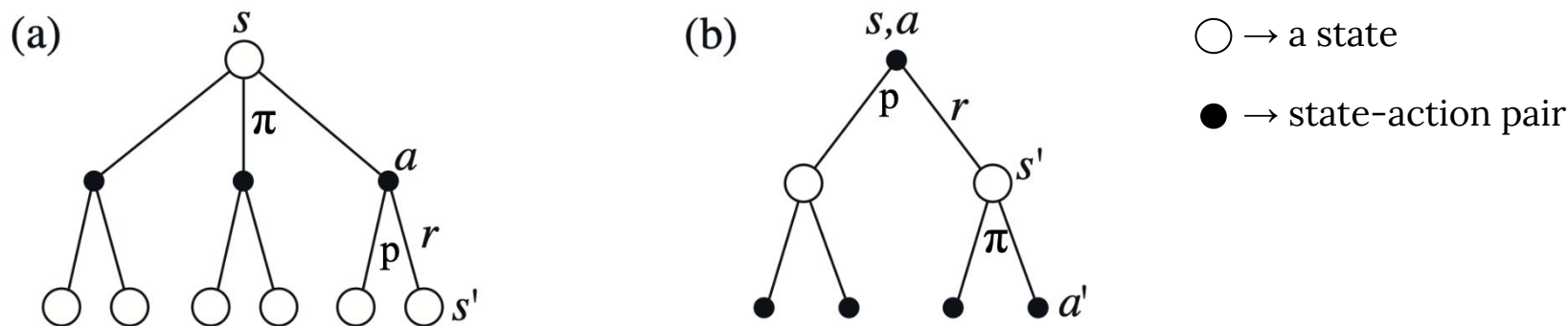
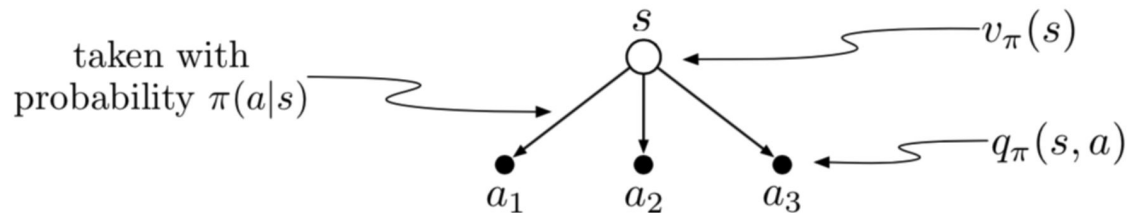


Figure 3.4: Backup diagrams for (a) v_{π} and (b) q_{π} .

What is the Bellman equation for action values for q_{π} ?

$$q_{\pi}(s, a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_{\pi}(s', a') \right]$$

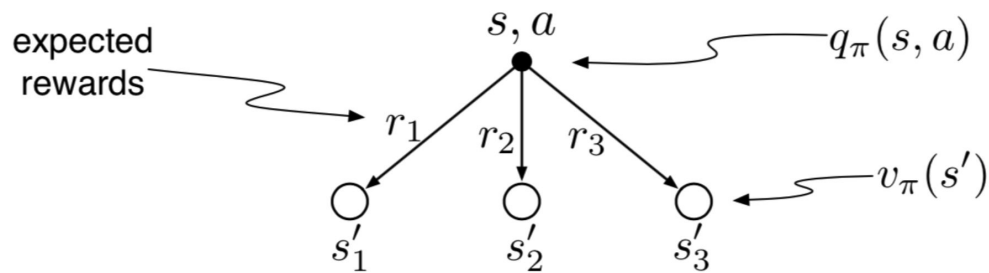
Backup Diagram



Backup diagram for example 3.18

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s, a)$$

Backup Diagram



Backup diagram for example 3.19

$$q_\pi(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')]$$

Optimal Value Functions

Optimal state-value function,

$$v_*(s) \doteq \max_{\pi} v_{\pi}(s),$$

for all $s \in \mathcal{S}$.

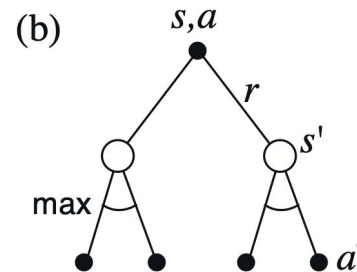
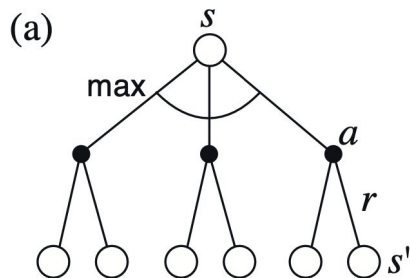
Optimal action-value function,

$$q_*(s, a) \doteq \max_{\pi} q_{\pi}(s, a),$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$.

Optimal Value Functions

Figure 3.7: Backup diagrams for (a) v_* and (b) q_*



$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')].$$

$$q_*(s, a) = \sum_{s', r} p(s', r | s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right].$$

Monte Carlo vs Temporal Difference

Monte Carlo: learning at the end of the episode

Monte Carlo Approach:

Monte Carlo: waits until the end of the episode, then calculates G_t (return) and uses it as a target for its value or policy.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

New value of state t

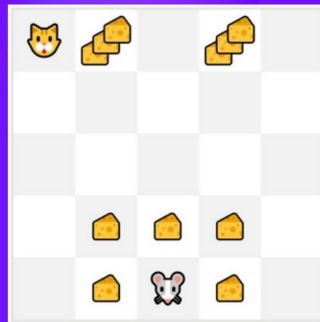
Former estimation
of value of state t
(= Expected return
starting at that state)

Learning
Rate

Return at
timestep
 t

Former estimation
of value of state t
(= Expected return
starting at that
state)

Monte Carlo Approach:



At the end of the episode:

- We have a list of **State, Actions, Rewards, and New States**.
- The agent will **sum the total rewards G_t** (to see how well it did).
- It will then **update $V(st)$** :

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Then **start a new game with this new knowledge**.

By running more and more episodes, the agent will learn to play better and better.

Monte Carlo vs Temporal Difference

Temporal Difference Learning: learning at each step

TD Learning Approach:

Temporal Difference Learning: learning at each time step.

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

New value
of state t

Former
estimation of
value of state
 t

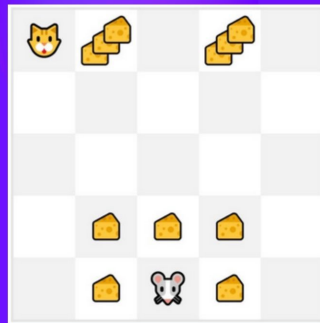
Learning
Rate

Reward

Discounted value of next
state

TD Target

TD Approach:



At the end of one step (State, Action, Reward, Next State):

- We have R_{t+1} and S_{t+1}
 - We update $V(S_t)$:
 - We estimate G_t by adding R_{t+1} and the discounted value of next state.
- TD target : $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Now we continue to interact with this environment with our updated value function. By running more and more steps, the agent will learn to play better and better.

$$NewEstimate \leftarrow OldEstimate + StepSize [Target - OldEstimate].$$

Monte Carlo vs Temporal Difference

Monte Carlo: $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$

TD Learning: $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

Q-Learning

Q-Learning is an off-policy value-based method that uses a TD approach to train its action-value function:

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 $Q(\text{terminal}, \cdot) = 0$

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'

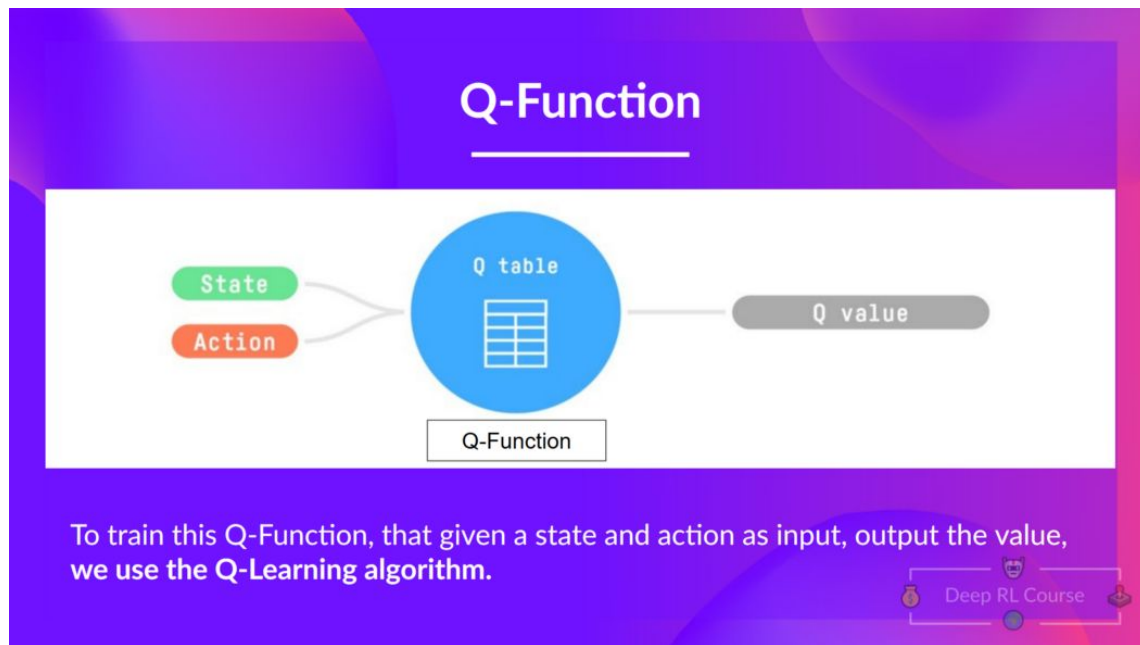
$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

 until S is terminal

Q-Learning

Q-Learning is an off-policy value-based method that uses a TD approach to train its action-value function:



Q-Learning

Step 1: We initialize the Q-table

Q-Learning, Step 1

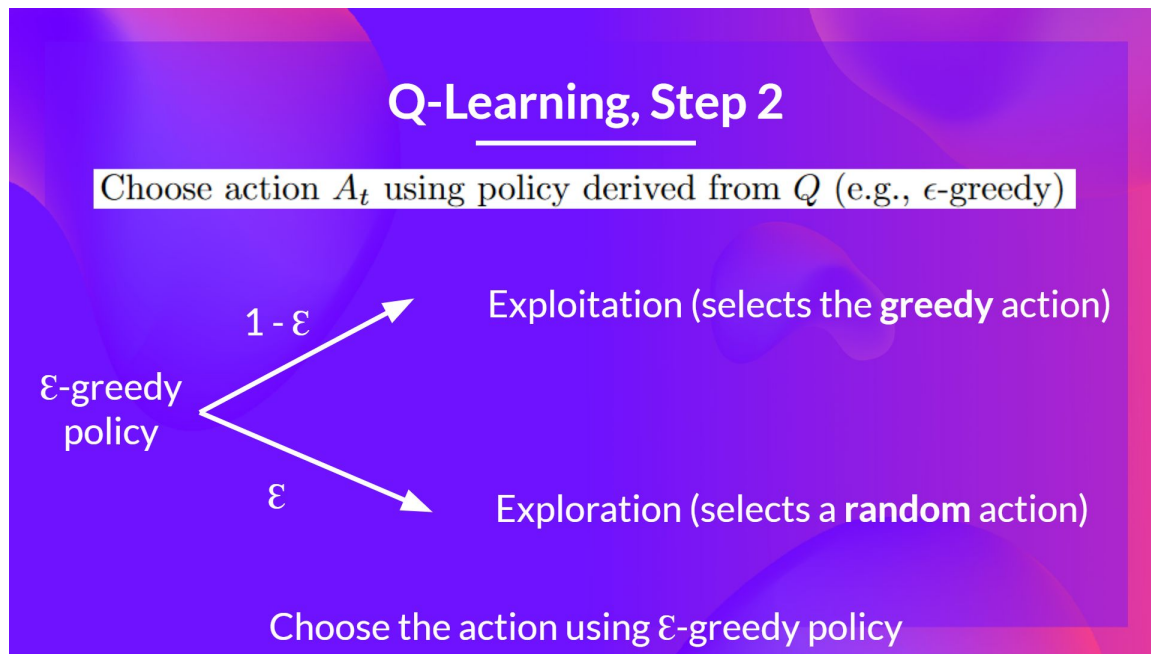
Initialize Q arbitrarily (e.g., $Q(s, a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\text{terminal-state}, \cdot) = 0$)

	←	→	↑	↓
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

We initialize the Q-Table

Q-Learning

Step 2: Choose an action using the epsilon-greedy strategy



Q-Learning

Step 3: Perform action A_t , get reward R_{t+1} and next state S_{t+1}

Q-Learning, Step 3

Take action A_t and observe R_{t+1}, S_{t+1}

Q-Learning

Step 4: Update $Q(S_t, A_t)$

$$\underbrace{Q(S_t, A_t)}_{\text{New Q-value estimation}} \leftarrow \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}} + \underbrace{\alpha}_{\text{Learning Rate}} \underbrace{[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]}_{\text{TD Error}}$$

$\underbrace{\hspace{10em}}_{\text{TD Target}}$