Q-learning

Петров Тимур

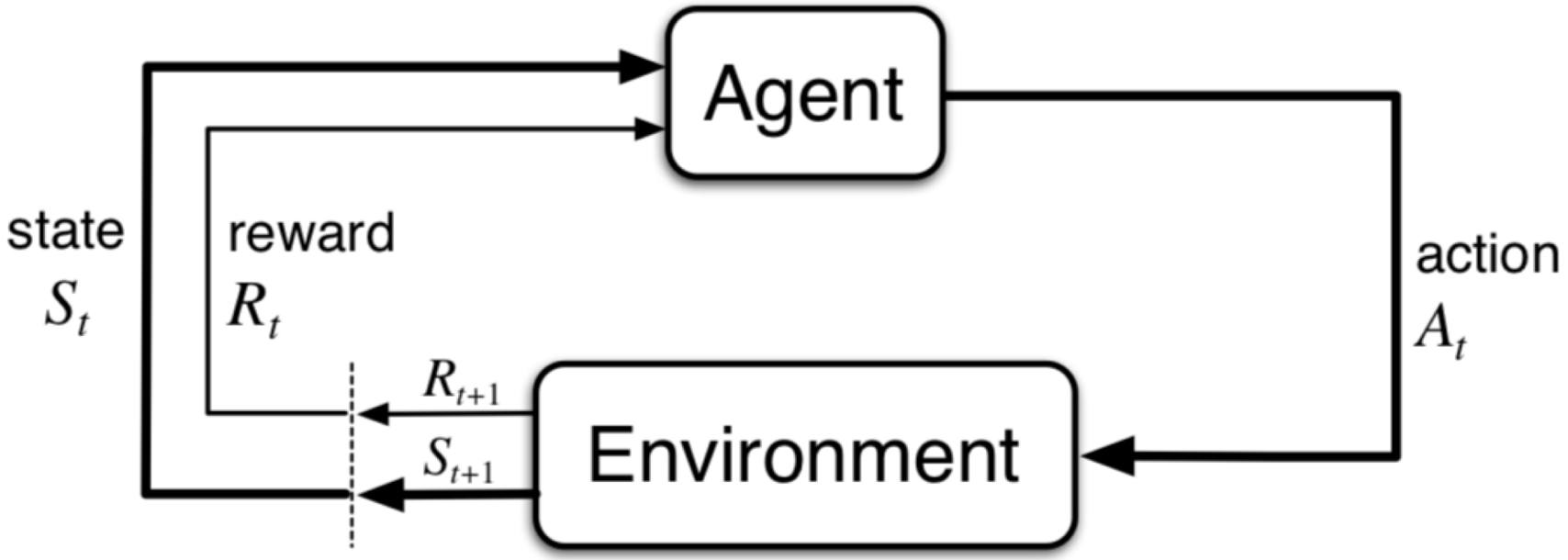
НИУ ВШЭ

30 ноября, 2018

План доклада

- Парадигмы решения задач RL:
  - Dynamic Programming (DP)
  - Monte-Carlo methods (MC)
  - Temporal-difference learning (TD)
- SARSA
- Q-learning & Double Q-learning
- Применение Q-learning:
  - FF-Q
  - Nash-Q
  - Correlated Q-learning

Напоминание



$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

Bellman equation (1)

$\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

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

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S}$$

Bellman equation (2)

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

$$v_*(s) \doteq \max_\pi v_\pi(s)$$

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r \,|\, s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big]. \end{aligned}$$

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \,|\, s, a)\Big[r + \gamma v_*(s')\Big] \end{aligned}$$

Dynamic Programming (1)

Идея: при заданной политике π с помощью оценки v_π можно улучшить политику: изменив какое-то одно действие увеличить оценку v_π . Так можно проделать для всех действий

Плюсы:

-Всегда сходится к оптимальному значению

-Алгоритм работает с оглядкой на предыдущие (и будущие) действия

Минусы:

- Надо знать среду (а именно распределение p)

Dynamic Programming (2)

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 $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   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

Monte-Carlo Methods (MC)

Идея: обучение с помощью повторений и подсчета среднего значения для каждого действия (получаем распределение из многочисленного повторения одного и того же эпизода)

Плюсы:

- Не нужно ничего знать про среду (в отличие от DP)
- Работает при возможности симуляции среды
- Быстро работает для небольших моделей

Минусы:

- Плохо сходится (невозможно понять, когда мы достигли оптимума)
- Проблема exploration/exploitation
- Подсчет идет вне зависимости от предыдущих действий

Temporal-difference learning (1)

Проблема DP: нужно знать среду (для вычисления p)

Проблема MC: плохо изучает и не обращает внимание на предыдущие действия

А что, если попытаться совместить эти два метода?



Temporal-difference learning (2)

TD(0):

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

Input: the policy π to be evaluated
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

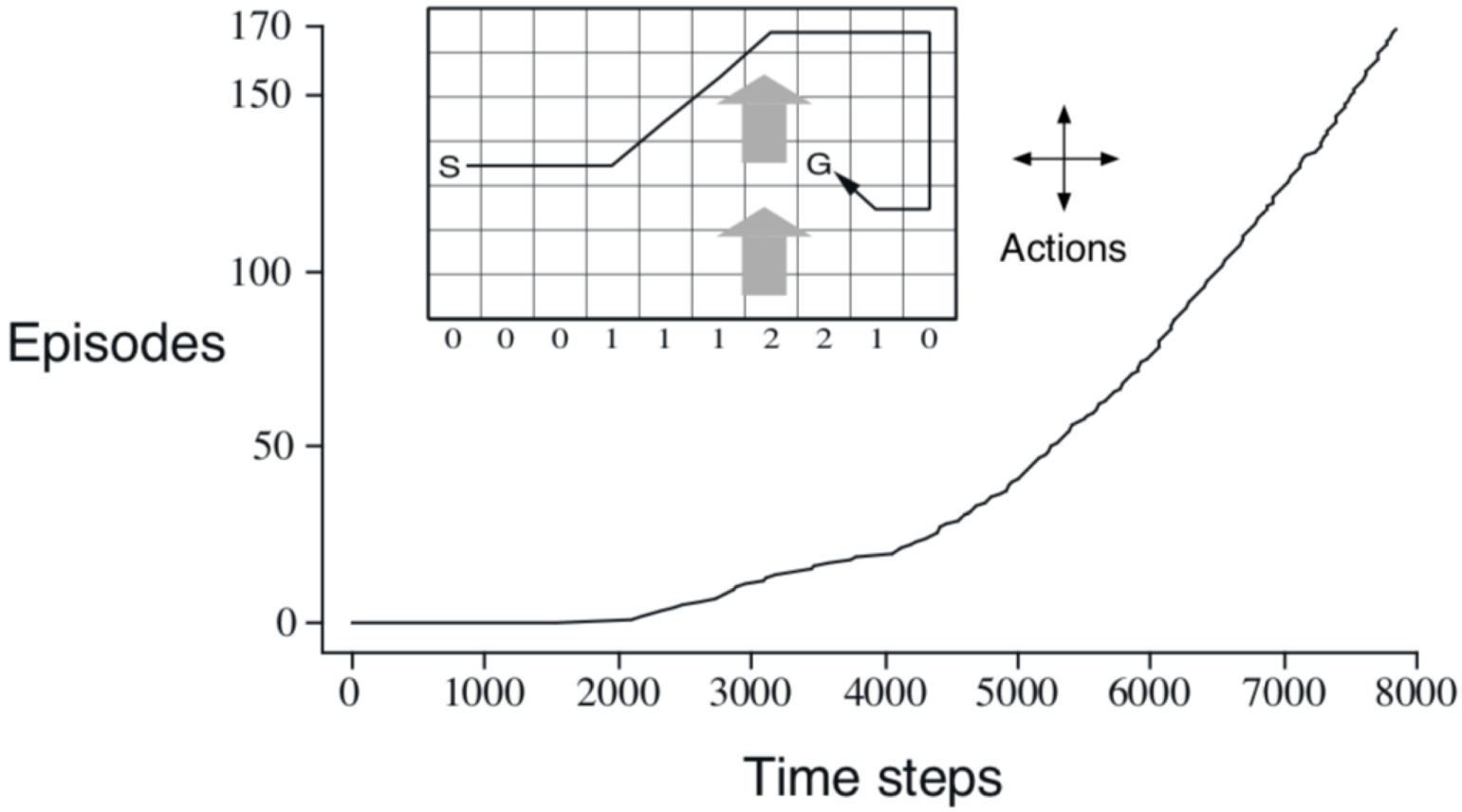
SARSA (On-policy TD control)

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

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

SARSA (2)

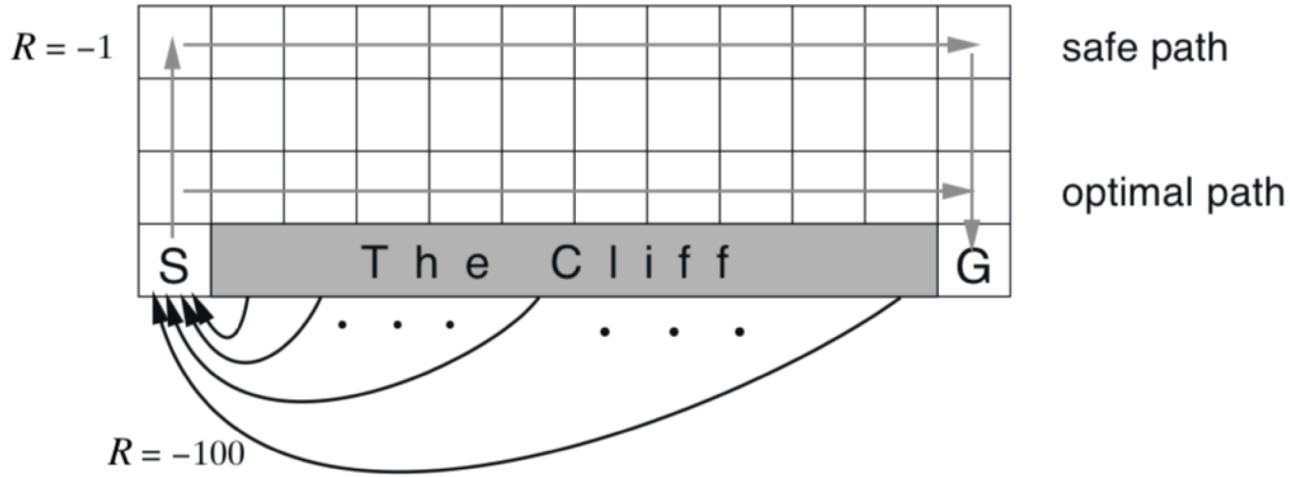

170
150
100
50
0
Episodes
S
G
0 0 0 1 1 1 2 2 1 0
Actions
0
1000
2000
3000
4000
5000
6000
7000
8000
Time steps

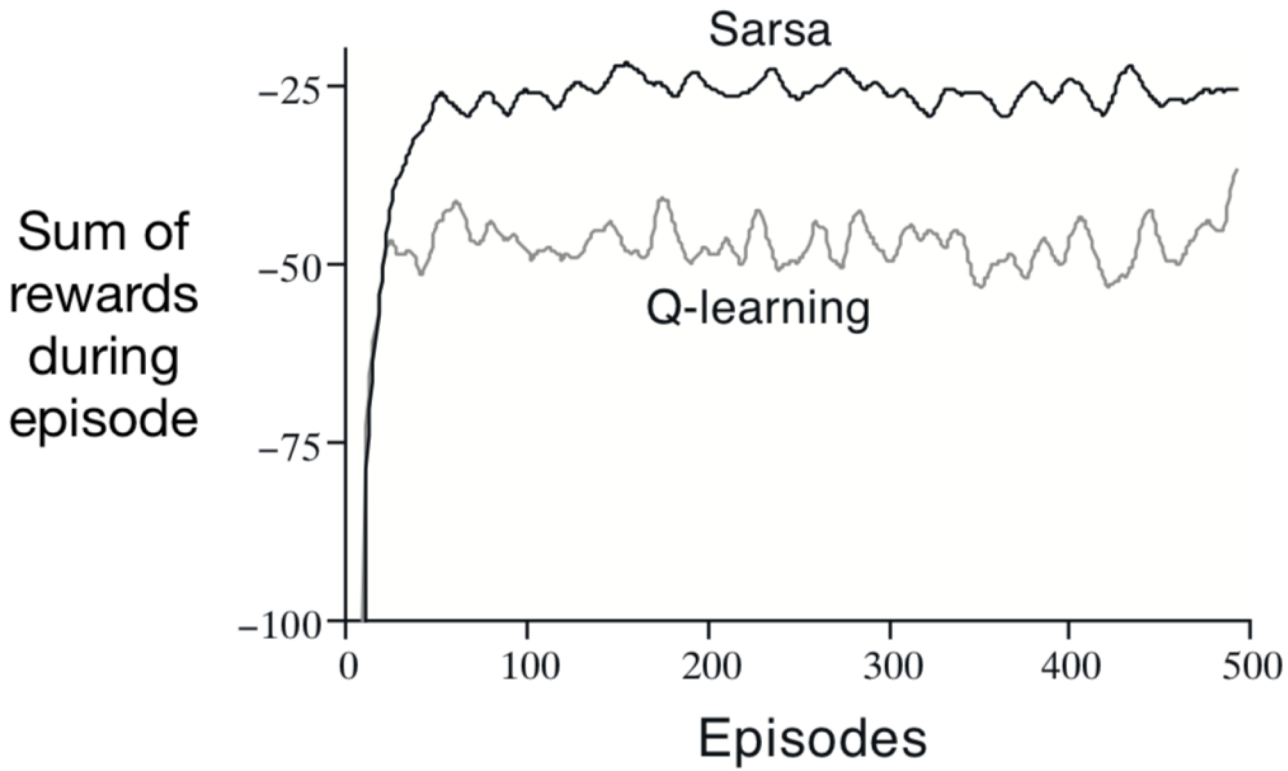
Q-learning (Off-policy TD control)

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

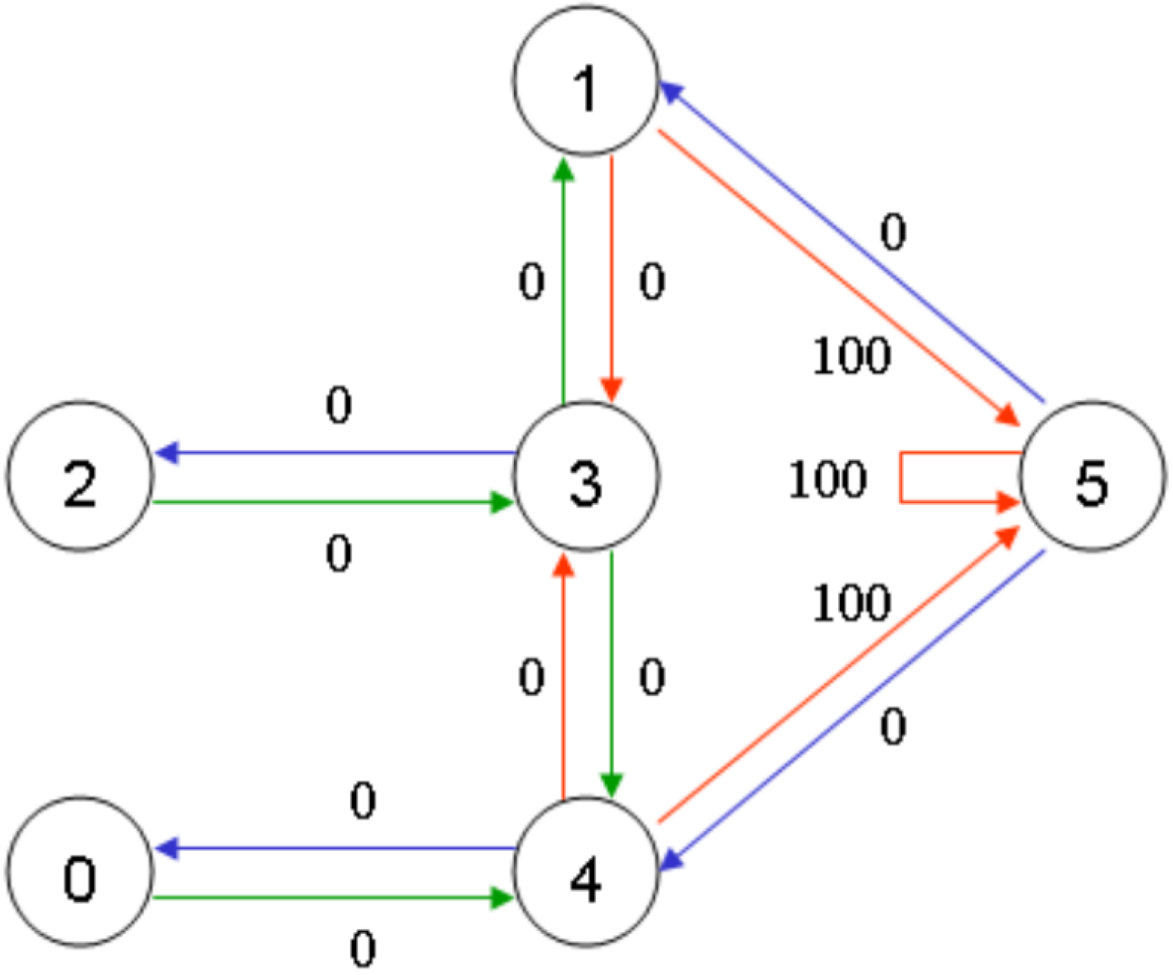
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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


R = −1
safe path
optimal path
S
The Cliff
G
R = −100


Sarsa
Q-learning
Sum of rewards during episode
−25
−50
−75
−100
0
100
200
300
400
500
Episodes

Q-learning (example)



Action

State	0	1	2	3	4	5
0	−1	−1	−1	−1	0	−1
1	−1	−1	−1	0	−1	100
2	−1	−1	−1	0	−1	−1
3	−1	0	0	−1	0	−1
4	0	−1	−1	0	−1	100
5	−1	0	−1	−1	0	100

$R =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$Q =$

Q-learning (example)

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \qquad Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

alpha = 1, gamma = 0.8

Q-learning convergence

Theorem

Given bounded rewards $|r_n| \leq \mathcal{R}$, learning rates $0 \leq \alpha_n < 1$, and

$$\sum_{i=1}^{\infty} \alpha_{n^i(x,a)} = \infty, \quad \sum_{i=1}^{\infty} [\alpha_{n^i(x,a)}]^2 < \infty, \quad \forall x, a,$$

then $\mathcal{Q}_n(x, a) \to \mathcal{Q}^*(x, a)$ as $n \to \infty$, $\forall x, a$, with probability 1.

Double Q-learning

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q_\cdot(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
        Take action A, observe R, S'
        With 0.5 probabilility:

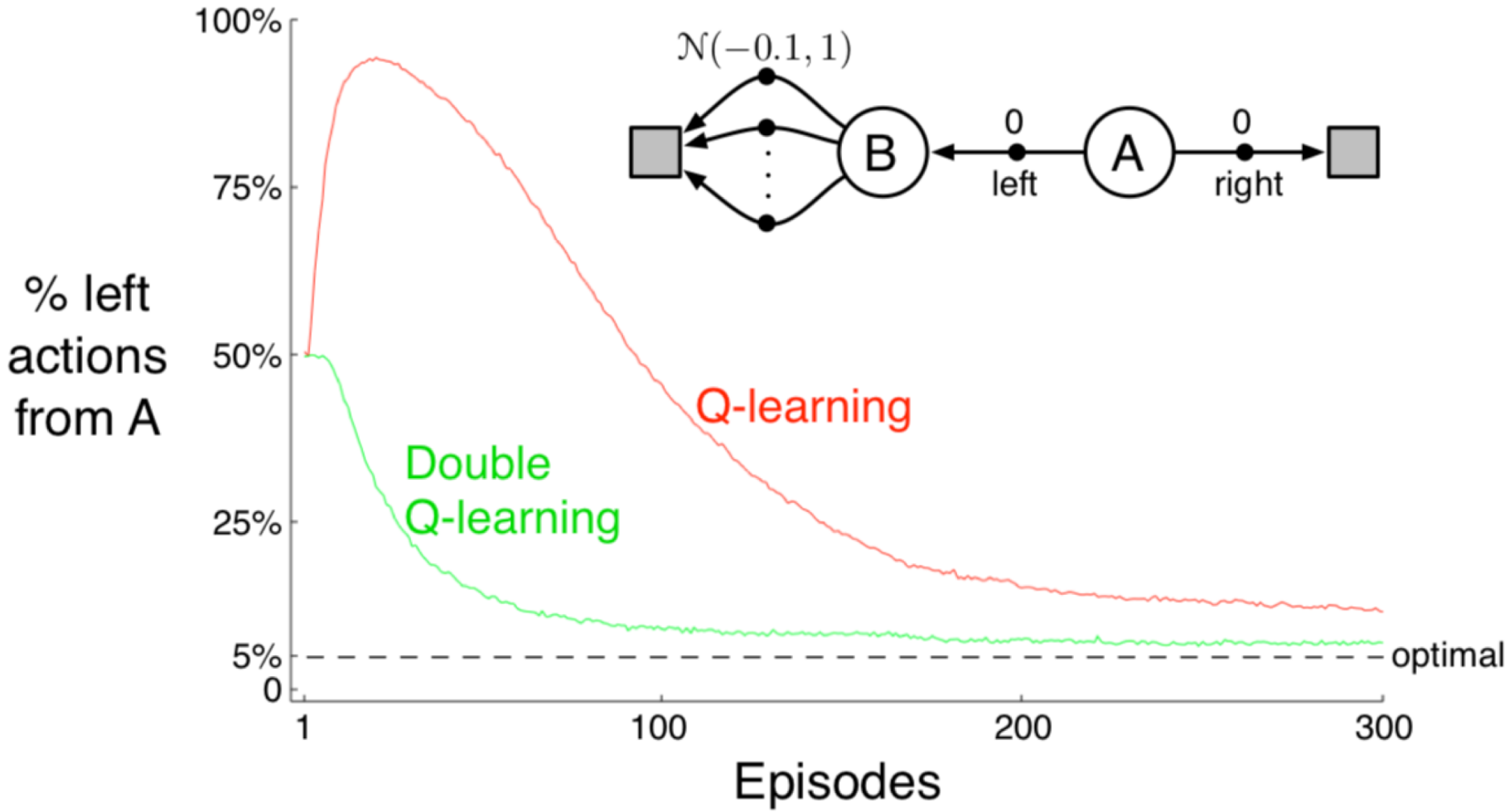
$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

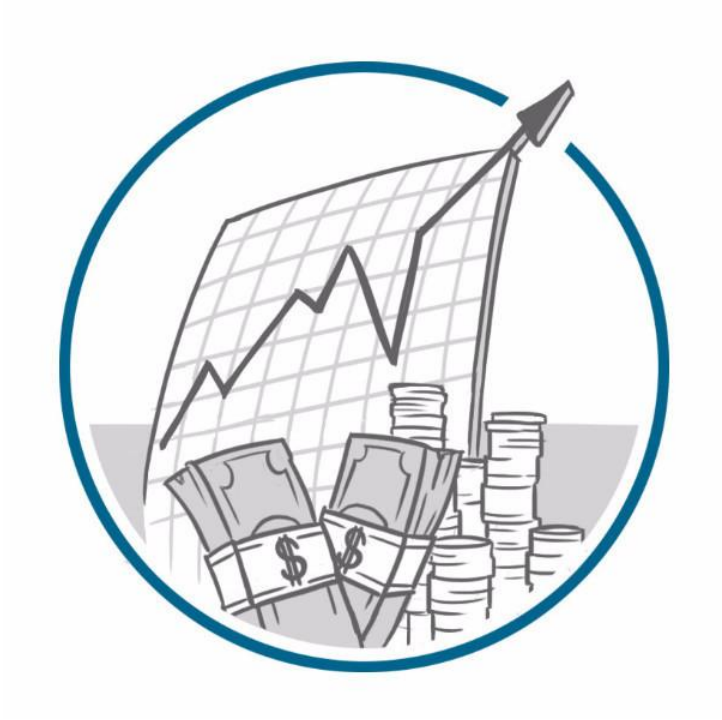
        else:

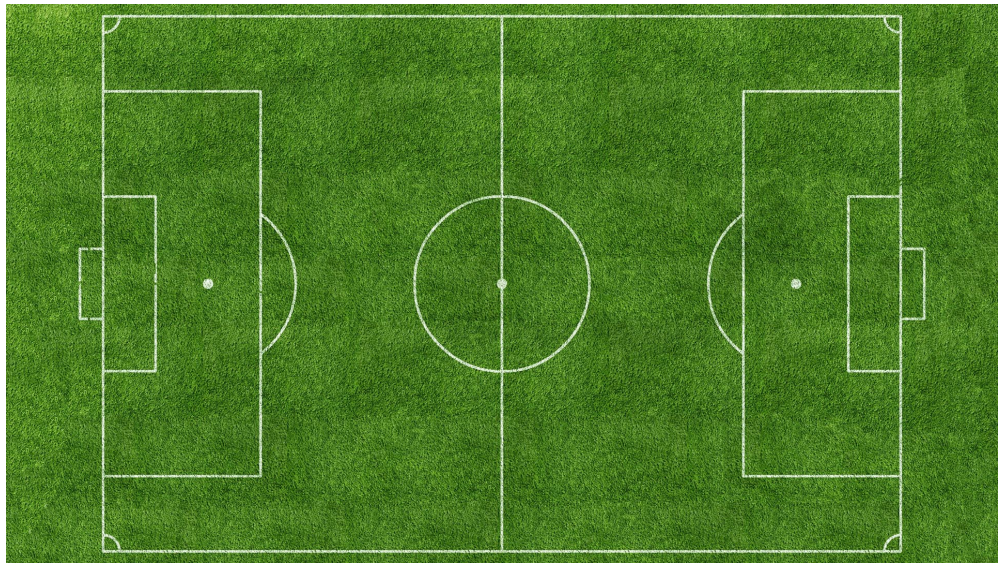
$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

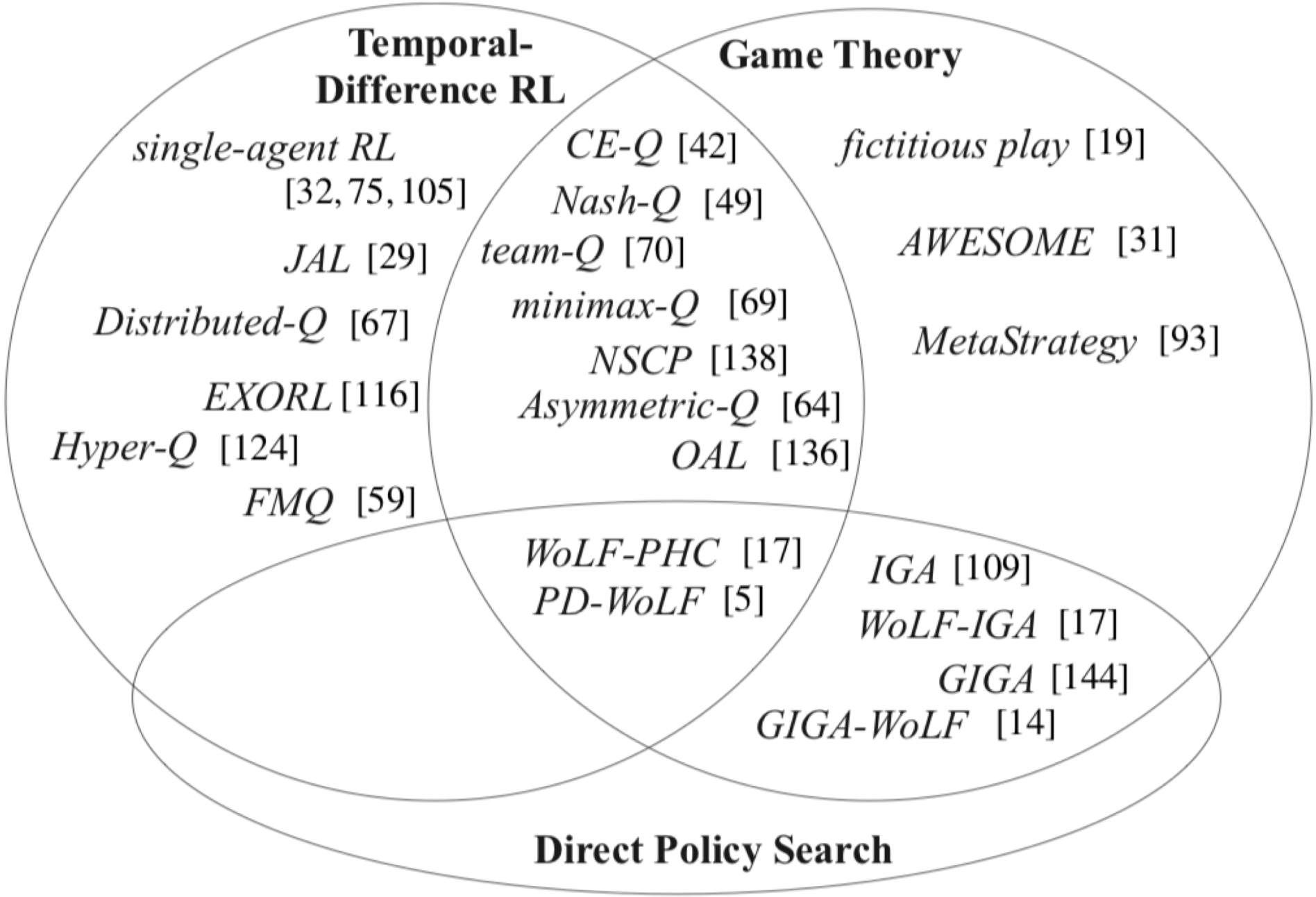
        $S \leftarrow S'$
    until S is terminal

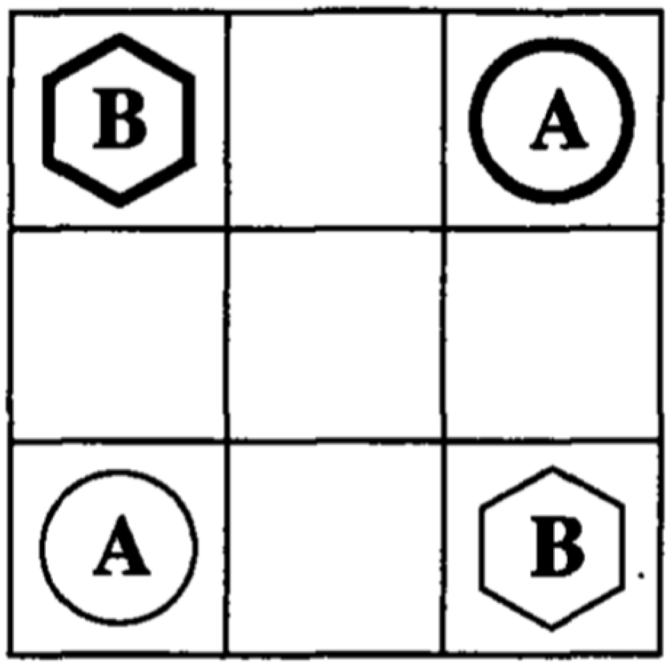
Double Q-learning (2)



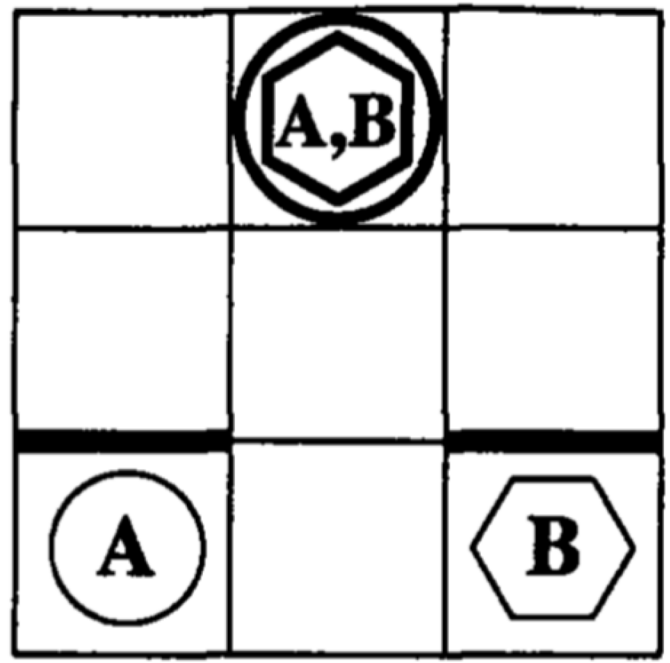




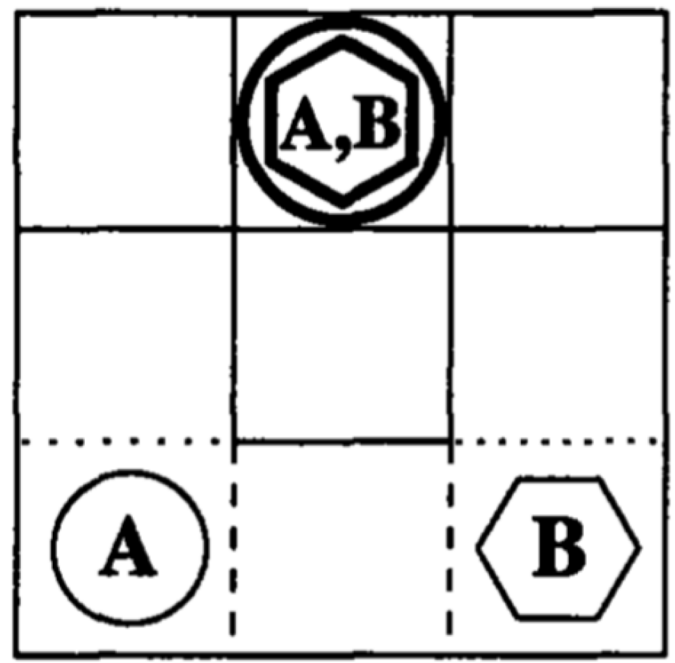

Temporal-Difference RL
Game Theory
Direct Policy Search
single-agent RL [32, 75, 105]
JAL [29]
Distributed-Q [67]
EXORL [116]
Hyper-Q [124]
FMQ [59]
CE-Q [42]
Nash-Q [49]
team-Q [70]
minimax-Q [69]
NSCP [138]
Asymmetric-Q [64]
OAL [136]
fictitious play [19]
AWESOME [31]
MetaStrategy [93]
WoLF-PHC [17]
PD-WoLF [5]
IGA [109]
WoLF-IGA [17]
GIGA [144]
GIGA-WoLF [14]



(a) Grid Game 1



(b) Grid Game 2



(c) Grid Game 3

Nash-Q

- Nash Q-function:

$$Q_*^i(s, a^1, \ldots, a^n) = r^i(s, a^1, \ldots, a^n) + \beta \sum_{s' \in S} p(s'|s, a^1, \ldots, a^n) v^i(s', \pi_*^1, \ldots, \pi_*^n),$$

- Update rule:

$$Q_{t+1}^i(s, a^1, \ldots, a^n) = (1 - \alpha_t) Q_t^i(s, a^1, \ldots, a^n) + \alpha_t \left[r_t^i + \beta Nash Q_t^i(s') \right]$$

$$Nash Q_t^i(s') = \pi^1(s') \cdots \pi^n(s') \cdot Q_t^i(s')$$

FF-Q (Friend-Foe Q-learning)

Initialize $Q(s, \langle a, o \rangle)$ and $\pi(s)$ arbitrarily

Initialize s
loop
$a \leftarrow$ probabilistic outcome $\pi(s)$ {Mixed with exploration policy}

Take action a, observe reward r, next state s' and opponent action o

$Q(s, \langle a, o \rangle) \leftarrow Q(s, \langle a, o \rangle) + \alpha\big(r + \gamma V(s') - Q(s, \langle a, o \rangle)\big)$
where
if Playing against foe **then**
$V(s) = \max_{\pi' \in PD(A)} \min_{o' \in O} \sum_{a' \in A} \pi(s, a')\, Q(s, \langle a', o' \rangle)$
$\pi(s) \rightarrow \arg \max_{\pi' \in PD(A)} \min_{o' \in O} \sum_{a' \in A} \pi(s, a')\, Q(s, \langle a', o' \rangle)$
else
$V(s) = \max_{a' \in A, o' \in O} Q(s, \langle a', o' \rangle)$
$\pi(s, a) = \begin{cases} 1 & a = \arg\max_{a' \in A} \{ \max_{o' \in O} Q(s, \langle a', o' \rangle) \} \\ 0 & \text{otherwise} \end{cases}$
end if

$s \leftarrow s'$
end loop

Correlated-Q

- uCE-Q: maximize the *sum* of the players' rewards:

$$\sigma \in \max_{\sigma \in CE} \sum_i \sum_{a \in A} \sigma(a) Q_i(s,a)$$

- eCE-Q: maximize the *minimum* of the players' rewards:

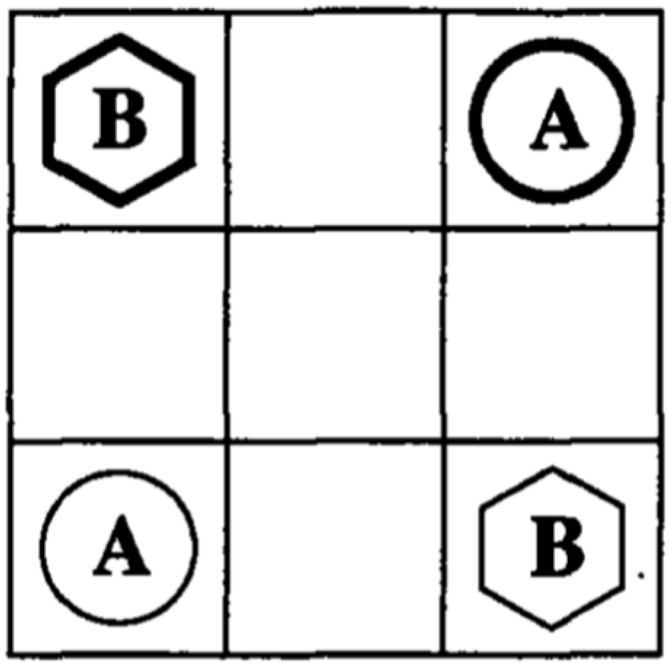
$$\sigma \in \max_{\sigma \in CE} \min_i \sum_{a \in A} \sigma(a) Q_i(s,a)$$

- rCE-Q: maximize the *maximum* of the players' rewards:

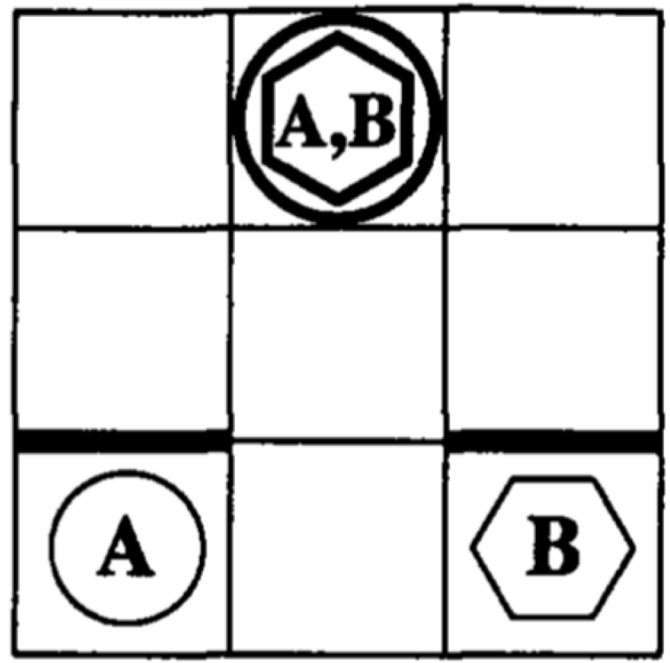
$$\sigma \in \max_{\sigma \in CE} \max_i \sum_{a \in A} \sigma(a) Q_i(s,a)$$

- lCE-Q: maximize the *maximum* of each player i's rewards:

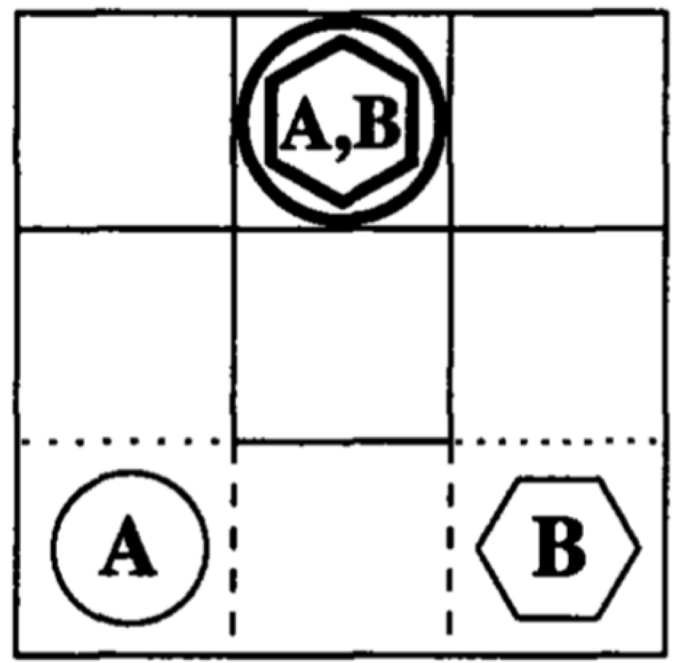
$$\sigma = \Pi_i \sigma_i, \sigma_i \in \max_{\sigma \in CE} \sum_{a \in A} \sigma(a) Q_i(s,a)$$



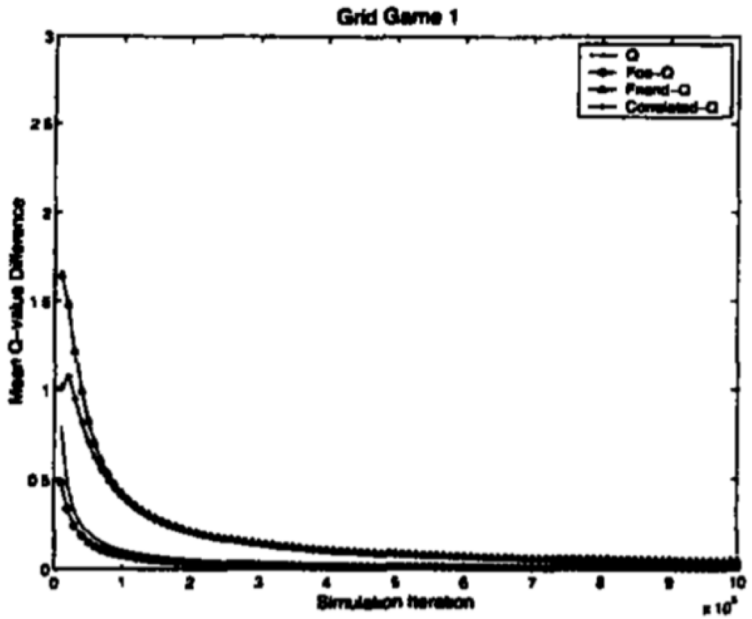
(a) Grid Game 1



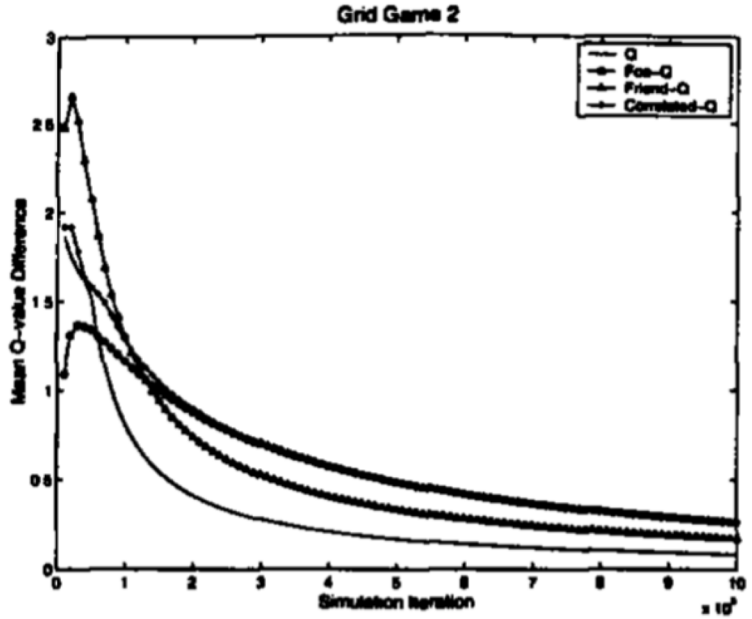
(b) Grid Game 2



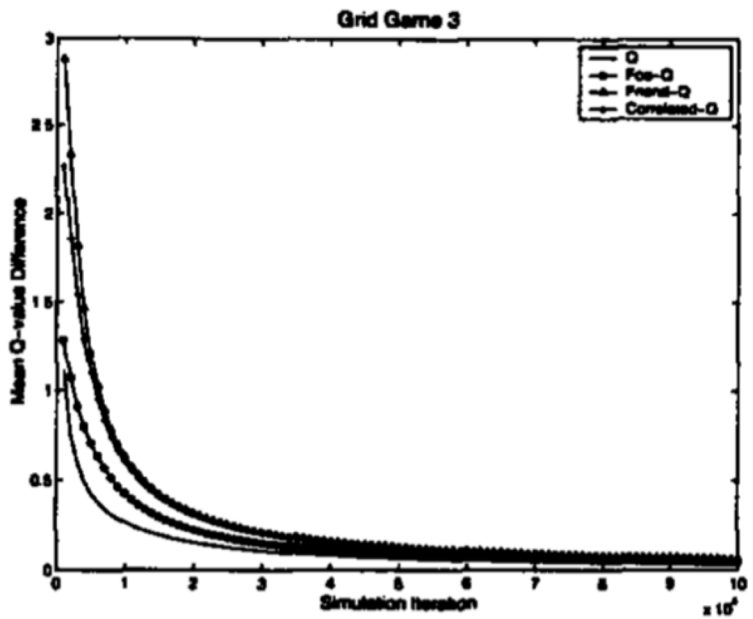
(c) Grid Game 3



(a) Grid Game 1



(b) Grid Game 2



(c) Grid Game 3

Figure 2: Convergence in the grid games: all algorithms are converging.

Grid Games	GG1		GG2		GG3	
Algorithm	Score	Games	Score	Games	Score	Games
Q	100,100	2500	49,100	3333	100,125	3333
Foe-Q	0,0	0	67,68	3003	120,120	3333
Friend-Q	100, 100	2500	$-\infty, -\infty$	0	$-\infty, -\infty$	0
Correlated-Q	100,100	2500	50,100	3333	117,117	3333

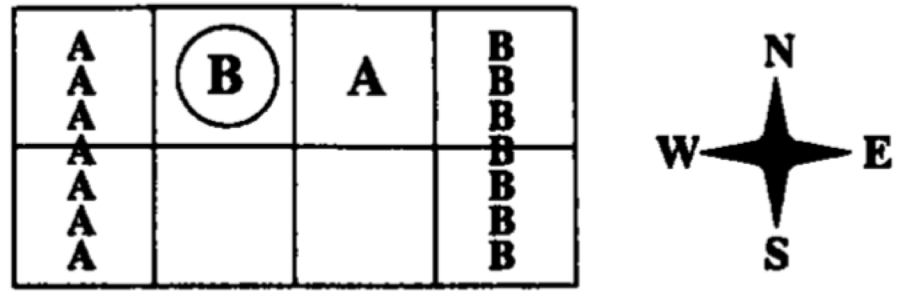
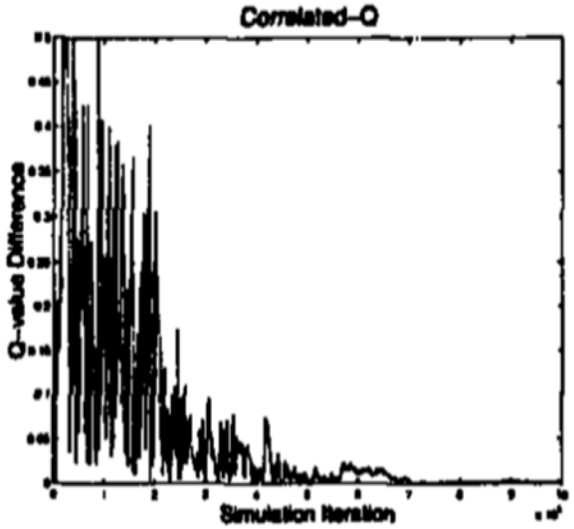
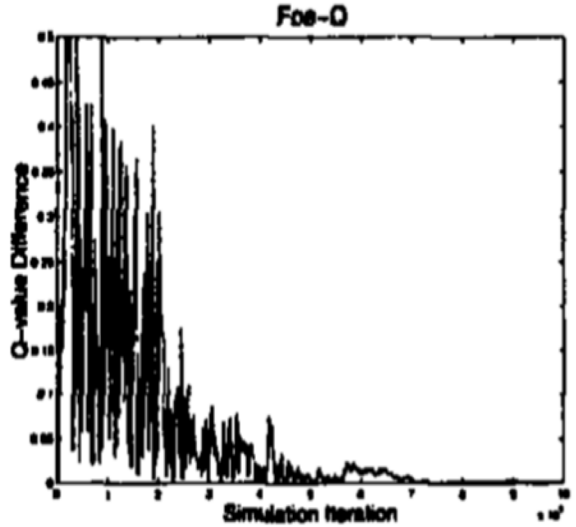


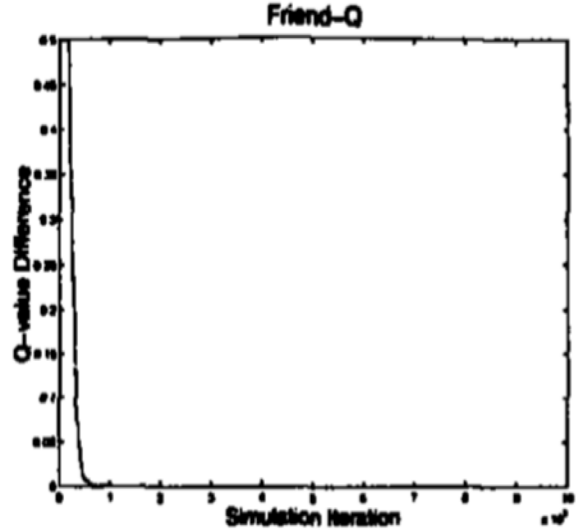
Figure 3: Soccer Game. State s.



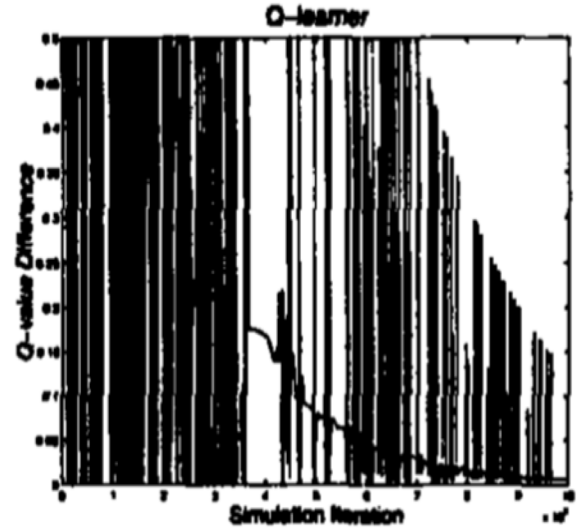
(a) Correlated-Q



(b) Foe-Q



(c) Friend-Q



(d) Q-learning

Выводы

- Q-learning – достаточно простой, но очень эффективный метод решения задач RL, не зависящий от среды и имеющий преимущества и DP, и MC
- Q-learning сходится почти наверное к оптимальному решению задачи (а модификация в виде double q-learning позволяет в некоторых случаях сходиться быстрее)
- Различные модификации обучения Q-learning позволяют решать различные задачи, например, стохастические игры с накопленной суммой

Ссылки на источники

- Introduction to RL: http://incompleteideas.net/book/bookdraft2018jan1.pdf
- Q-learning (Technical Note): https://link.springer.com/content/pdf/10.1007%2FBF00992698.pdf
- Nash-Q: http://www.jmlr.org/papers/volume4/hu03a/hu03a.pdf
- Correlated-Q: http://www.aaai.org/Papers/Symposia/Spring/2002/SS-02-02/SS02-02-012.pdf
- MARL: http://www.dcsc.tudelft.nl/~bdeschutter/pub/rep/10_003.pdf