# Lecture 23: Multi-agent Reinforcement Learning

## Multi Agent Q-learning Template

MultiQ(StochastiGame, $f, \gamma, \alpha, T$)

Inputs equilibrium selection function $f$

discounting factor $\gamma$

learning rate $\alpha$

total training time $T$

Outputs state – value functions $V_i^*$

action – value functions $Q_i^*$

Initialize $s, a_1, \ldots, a_n$ and $Q_1, \ldots, Q_n$

for $t = 1\!:\!T$

1. select actions $\vec{a} = (a_1, \ldots, a_n)$in state $s$
2. observe rewards $r_1, \ldots, r_n$ and next state $s'$
3. for $i = 1$ to $n$ (for each agent)

   (a) $V_i(s') = f_i(Q_1(s', \vec{a}), \ldots, Q_n(s', \vec{a}))$

   (b) $Q_i(s, \vec{a}) = (1 - \alpha_i) Q_i(s, \vec{a}) + \alpha_i [r_i + \gamma V_i(s')]$
4. agent choose actions action $a_1', \ldots, a_n'$
5. $s = s', a_1 = a_1', \ldots, a_n = a_n'$
6. adjust learning rate $\alpha = (\alpha_1, \ldots, \alpha_n)$

## Multi Agent Reinforcement Learning (MARL)

### Multi Agent Q-learning Template

Equilibrium selection function $f$ : $V_i(s') = f_i(Q_1(s', \vec{a}), \ldots, Q_n(s', \vec{a}))$

- We going to study the following equilibrium concept:
  - Value function based (Bellman function based)
    - Single agent Q-learning
    - Independent Q learning by multiple agents
    - Nash-Q learning (Hu and Wellman 1998)
    - Minmax-Q learning (Littman 1994)
    - Friend-or-Foe Q learning (Littman 2001)
    - Correlated Q learning (Greenwald and Hall 2003)
  - Policy gradient methods (direct search for policy)
    - Wind-or-Learn-Fast Policy Hill Climbing (WOLF-PHC) (Policy gradient method)

## Single agent Q learning

$$\text{(a) } V(s') = f\big(Q(s',a)\big) = \max_a Q(s',a)$$

$$\text{(b) } Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma V(s')]$$

$$= (1-\alpha)Q(s,a) + \alpha\left[r + \gamma \max_a Q(s',a)\right]$$

- Equivalent to Q-learning algorithm we have discussed couple weeks a go

## Independent Q learning by multiple agents

(a) $V_i(s') = f_i\big(Q_1(s', a_1), \dots, Q_i(s', a_i), \dots, Q_n(s', a_n)\big) = \max_{a_i} Q_i(s', a_i)$

(b) $Q_i(s, a_i) = (1-\alpha)Q_i(s, a_i) + \alpha[r_i + \gamma V_i(s')]$

$$= (1-\alpha)Q_i(s, a_i) + \alpha\left[r_i + \gamma \max_{a_i} Q_i(s', a_i)\right]$$

- There are $n$ agents whose Q-table is being independently updated regardless of the actions taken by other users
  - $Q_i(s', a_i) \sim Q_i(s', a_1, \dots, a_n)$
- Still the transition of joint state $s$ depends on the all the actions taken by all agents, i.e., $p(s'|s, a_1, \dots a_i, \dots, a_n)$
  - Independent Q-learning thus ignore the effects of other agents' actions on state transition
    - ➢ treats other agents as a part of stochastic environment
    - ➢ Due to incomplete information on others' action, the agent cannot accurately learn the dynamic of the system

## Nash-Q learning

**Definition (Optimal Q-function)**

Optimal Q function is defined as

$$Q^*(s,a) = r_i(s,a,s') + \gamma \sum_{s' \in S} p(s'|s,a) V_i^*(s')$$

- $V_i^*(s') = \max_a Q^*(s',a)$
- With **optimum** policy $\pi^*(s') = \operatorname{argmax}_a Q^*(s',a)$

**Definition (Nash Q-function)**

Nash-Q function is defined as

$$Q_i^*(s,a_1,\dots,a_n) = r_i(s,a_1,\dots,a_n,s') + \gamma \sum_{s' \in S} p(s'|s,a_1,\dots,a_n) \underbrace{V_i(s',\pi_1^*,\dots,\pi_n^*)}_{Nash\ Q_i(s')}$$

- $V_i(s',\pi_1^*,\dots,\pi_n^*) = Q_i^*\big(s,\pi_1^*(s'),\dots,\pi_n^*(s')\big) =$ Nash $Q_i(s')$
- with Nash **equilibrium** strategy $(\pi_1^*,\dots,\pi_i^*,\dots,\pi_n^*)$ satisfying for all $s'$ and $i = 1,\dots,n$

$$V_i(s',\pi_1^*,\dots,\pi_i^*,\dots,\pi_n^*) \geq V_i(s',\pi_1^*,\dots,\pi_i,\dots,\pi_n^*) \text{ for all } \pi_i \in \Pi_i$$

## Nash-Q learning

- Q-learning directly find optimal Q-function (Q table) instead of optimum finding policy $\pi^*$

- Single agent Q-learning:
  - Iteratively find optimal Q values $Q^*(s,a)$ (table)

$$
\begin{aligned}
Q(s,a) &= (1-\alpha)Q(s,a) + \alpha[r(s,a) + \gamma V(s')] \\
&= (1-\alpha)Q(s,a) + \alpha\left[r(s,a) + \gamma \max_a Q(s',a)\right]
\end{aligned}
$$

- Nah Q-learning:
  - Iteratively find Nash-Q values Nash $Q_i^*(s,a_1,\dots,a_n)$ (table for each agent)

$$
\begin{aligned}
Q_i(s,\vec{a}) &= (1-\alpha)Q_i(s,\vec{a}) + \alpha[r_i + \gamma V_i(s')] \\
&= (1-\alpha)Q_i(s,\vec{a}) + \alpha[r_i + \gamma \text{ Nash } Q_i(s')]
\end{aligned}
$$

$Q_i(s',\vec{a})$ : Nash-Q values (state-action values)

Nash $Q_i(s')$: **Nash equilibrium value** of Nash-Q values

- the learning agent updates its Nash Q-value depending on the joint strategy of all the players and not only its own expected payoff.

## The Nash Q-Learning algorithm

MultiQ(StochastiGame, $f, \gamma, \alpha, T$)

| | |
|---|---|
| Inputs | equilibrium selection function $f$ |
| | discounting factor $\gamma$ |
| | learning rate $\alpha$ |
| | total training time $T$ |
| Outputs | state − value functions $V_i^*$ |
| | action − value functions $Q_i^*$ |
| Initialize | $s, a_1, \ldots, a_n$ and $Q_1, \ldots, Q_n$ |

for $t = 1{:}T$

1. select actions $a_1, \ldots, a_n$ in state $s$
2. observe rewards $r_1, \ldots, r_n$ and next state $s'$
3. for $i = 1$ to $n$ (for each agent)
   (a) $V_i(s') = f_i\big(Q_1(s', \vec{a}), \ldots, Q_n(s', \vec{a})\big) = \text{Nash}Q_i(s')$
   (b) $Q_i(s, \vec{a}) = (1 - \alpha_i)Q_i(s, \vec{a}) + \alpha_i[r_i + \gamma V_i(s')]$
4. agent choose actions action $a_1', \ldots, a_n'$
5. $s = s', a_1 = a_1', \ldots, a_n = a_n'$
6. adjust learning rate $\alpha = (\alpha_1, \ldots, \alpha_n)$

## Nash-Q learning algorithm

For agent $i$

$$\text{(a) } V_i(s') = f_i\Big(\underbrace{Q_1(s',\vec{a}),\dots,Q_i(s',\vec{a}),\dots,Q_n(s',\vec{a})}_{\text{Nash Q values for agents } i\,=\,1:n}\Big) = \underbrace{\text{Nash } Q_i(s')}_{\text{Nash equilibrium value}}$$

$$\text{(b) } Q_i(s,\vec{a}) = (1-\alpha)Q_i(s,\vec{a}) + \alpha[r_i + \gamma V_i(s')]$$
$$= (1-\alpha)Q_i(s,\vec{a}) + \alpha[r_i + \gamma \text{ Nash } Q_i(s')]$$

- It uses the principle of the Nash equilibrium where "each player effectively holds a correct expectation about the other players' behaviors, and acts rationally with respect to this expectation
  - "rationally" means that the agent will have a strategy that is a best response for the other players' strategies.
- Nash Q-Learning is more complex than multiagent Q-learning because each player needs to keep track of the other players actions and rewards.
  - Each agent needs to track Nash Q values $Q_1(s',\vec{a}),\dots,Q_n(s',\vec{a})$ for agents $i = 1:n$ to compute the Nash equilibrium value Nash $Q_i(s')$

## How to compute Nash $Q_i(s')$ ?

1. At state $s'$, agent $i$ have the $n$ Nash Q-values being tracked

$$\{Q_1(s',\vec{a}),\dots,Q_i(s',\vec{a}),\dots,Q_n(s',\vec{a})\}$$

   - In each state, the agent has to keep track of every other agent's actions and rewards.

2. Find the Nash equilibrium for the stage game $\{Q_1(s',\vec{a}),\dots,Q_i(s',\vec{a}),\dots,Q_n(s',\vec{a})\}$

   - For example, in two player general sum game, the agent $i$ will build the matrix game for state $s'$ with the reward matrix of both players as shown

| | $a_2^1$ | $a_2^2$ |
|---|---|---|
| $a_1^1$ | $Q_1(s',a_1^1,a_2^1), Q_2(s',a_1^1,a_2^1)$ | $Q_1(s',a_1^1,a_2^2), Q_2(s',a_1^1,a_2^2)$ |
| $a_1^2$ | $\boxed{Q_1(s',a_1^2,a_2^1), Q_2(s',a_1^2,a_2^1)}$ | $Q_1(s',a_1^2,a_2^2), Q_2(s',a_1^2,a_2^2)$ |

3. Compute the Nash equilibrium $\vec{a}_{NE}$ for the stage game (i.e., greedy optimization in single agent Q learning) and compute the Nash equilibrium value Nash $Q_i(s')$ for player $i$ at state $s'$

$$\text{Nash } Q_i(s') = Q_i(s',\vec{a}_{NE})$$

## How to compute Nash $Q_i(s')$ ?

4. Update Nash Q-values using the computed Nash equilibrium values

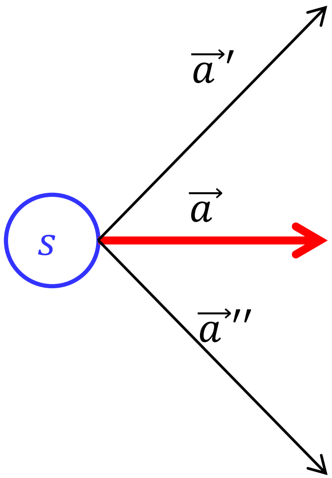
| | $a_2^1$ | $a_2^2$ |
|---|---|---|
| $a_1^1$ | $Q_1(s', a_1^1, a_2^1), Q_2(s', a_1^1, a_2^1)$ | $Q_1(s', a_1^1, a_2^2), Q_2(s', a_1^1, a_2^2)$ |
| $a_1^2$ | $Q_1(s', a_1^2, a_2^1), Q_2(s', a_1^2, a_2^1)$ | $Q_1(s', a_1^2, a_2^2), Q_2(s', a_1^2, a_2^2)$ |

$$Q_1(s, a_1, a_2) = (1-\alpha)Q_1(s, a_1, a_2) + \alpha[r_1 + \gamma \text{ Nash } Q_1(s')]$$

$$Q_2(s, a_1, a_2) = (1-\alpha)Q_2(s, a_1, a_2) + \alpha[r_2 + \gamma \text{ Nash } Q_2(s')]$$

## Nash-Q learning : Procedure

$S_t$ $A_t$ $R_{t+1}$ $S_{t+1}$ $NashA_{t+1}$ $S_{t+2}$

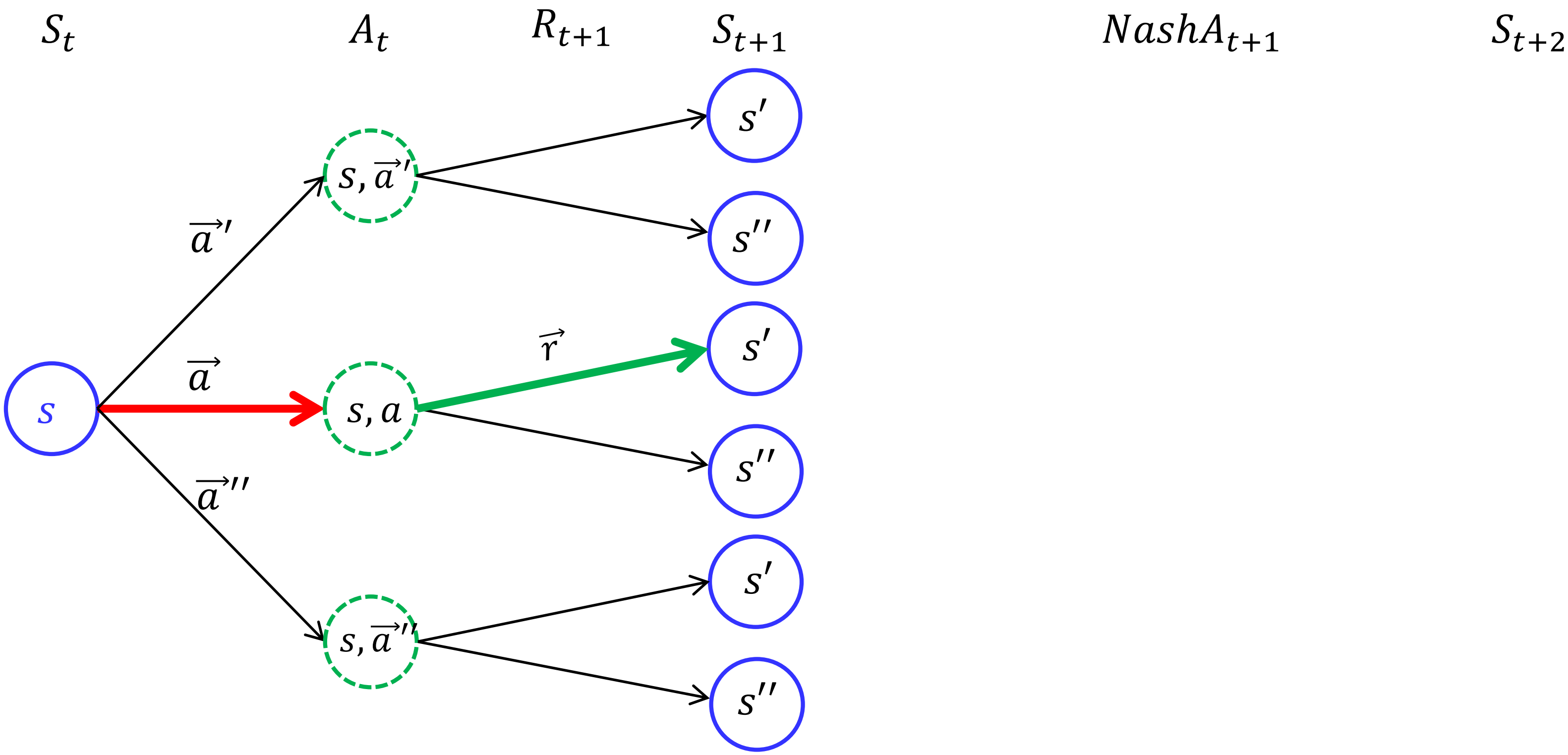



- Choose action $\vec{a} = (a_1, \dots, a_n)$ from $s$ using current Nash Q values $(Q_1(s,\vec{a}), \dots, Q_n(s,\vec{a}))$

$$\vec{a} = \begin{cases} \vec{a}_{\mathrm{NE}} \text{ for } (Q_1(s,\vec{a}), \dots, Q_n(s,\vec{a})) & \text{with prob } 1-\epsilon \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

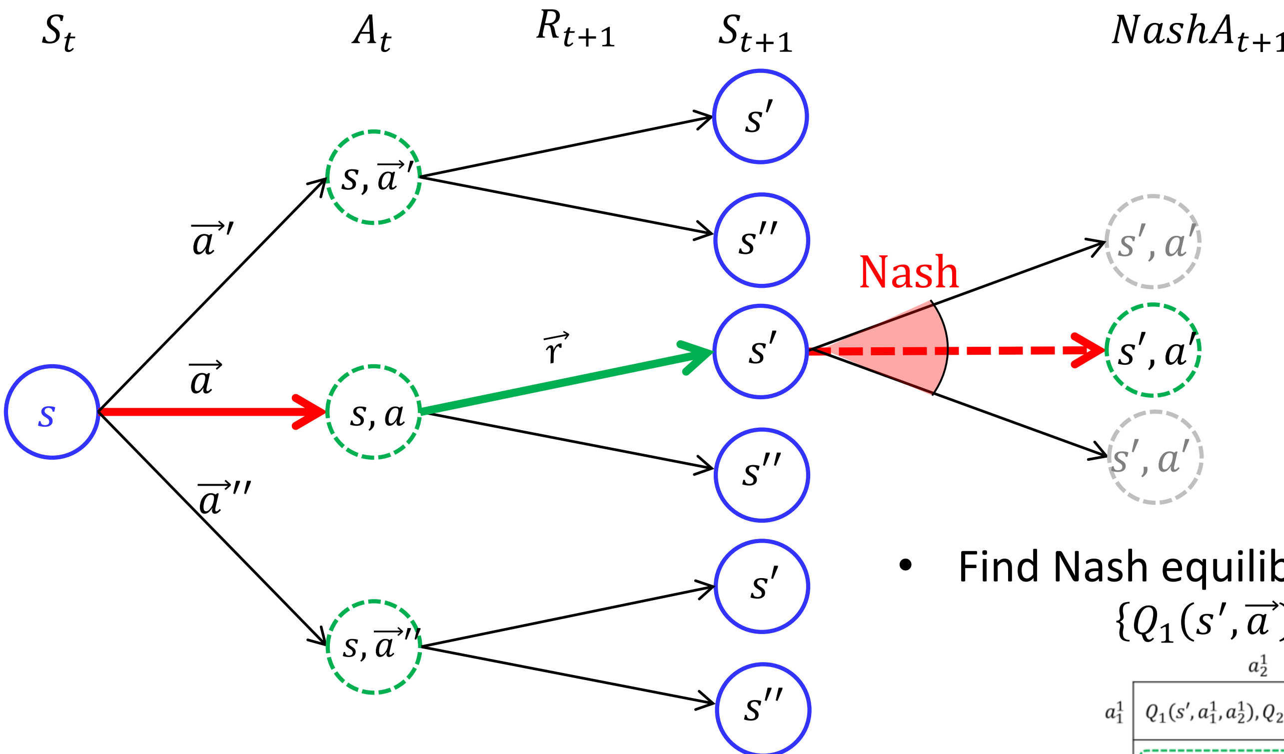
Any exploration policy can be used

## Nash-Q learning : Procedure




- Take action $\vec{a} = (a_1, \dots, a_n)$ given $s$ and observe reward $\vec{r} = (r_1, \dots, r_n)$ and the next state $s'$

## Nash-Q learning : Procedure




- Find Nash equilibrium for the stage game $\{Q_1(s',\vec{a}),\ldots,Q_n(s',\vec{a})\}$

| | $a_2^1$ | $a_2^2$ |
|---|---|---|
| $a_1^1$ | $Q_1(s',a_1^1,a_2^1),Q_2(s',a_1^1,a_2^1)$ | $Q_1(s',a_1^1,a_2^2),Q_2(s',a_1^1,a_2^2)$ |
| $a_1^2$ | $Q_1(s',a_1^2,a_2^1),Q_2(s',a_1^2,a_2^1)$ | $Q_1(s',a_1^2,a_2^2),Q_2(s',a_1^2,a_2^2)$ |

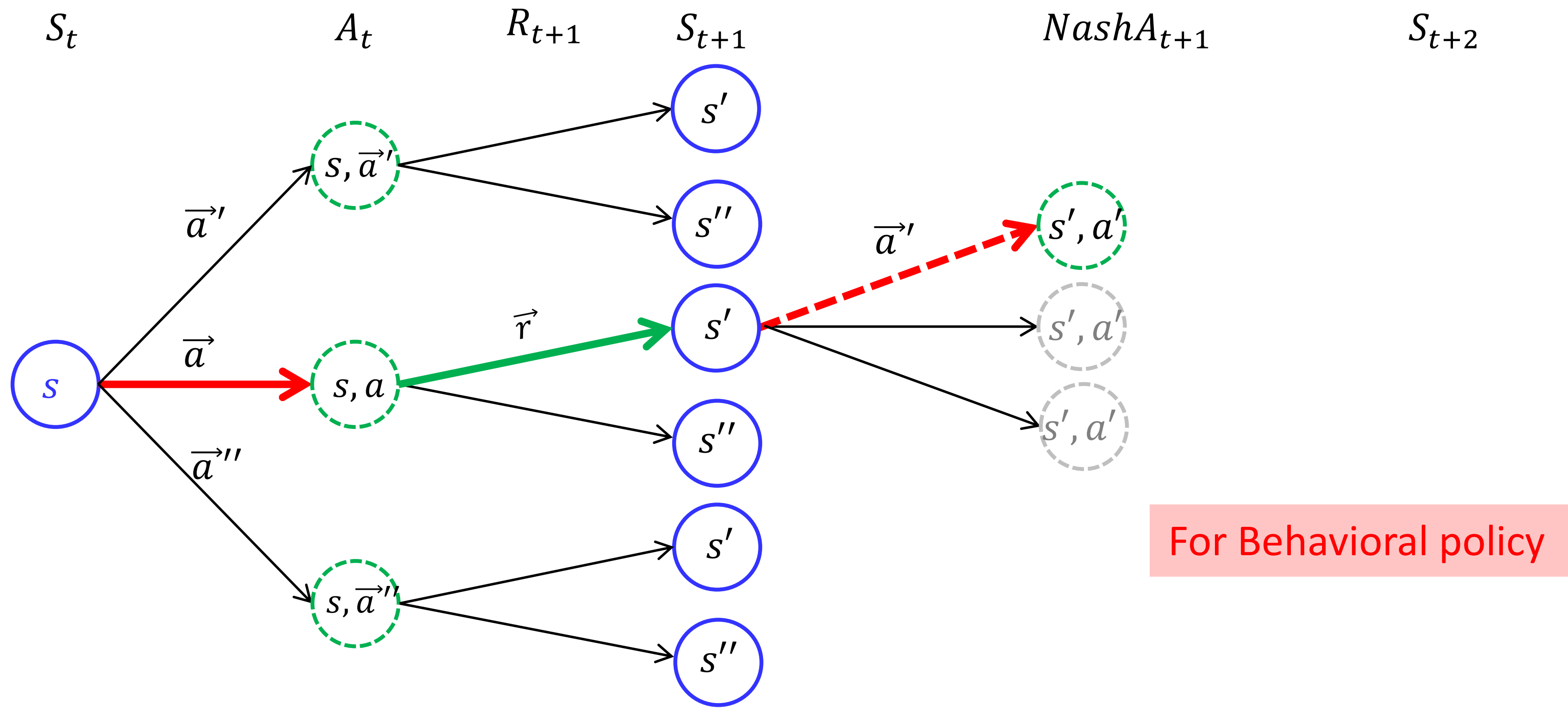
- The Nash equilibrium values are
  Nash $Q_1(s')$ , … , Nash $Q_n(s')$

- Update Nash-Q values, $Q_1(s,\vec{a}),\ldots,Q_n(s,\vec{a})$, using the Nash equilibrium values

  For $i = 1{:}n$

$$Q_i(s,\vec{a}) \leftarrow (1-\alpha)Q_i(s,\vec{a}) + \alpha[r_i + \gamma\,\text{Nash}\,Q_i(s')]$$

# Nash-Q learning : Procedure

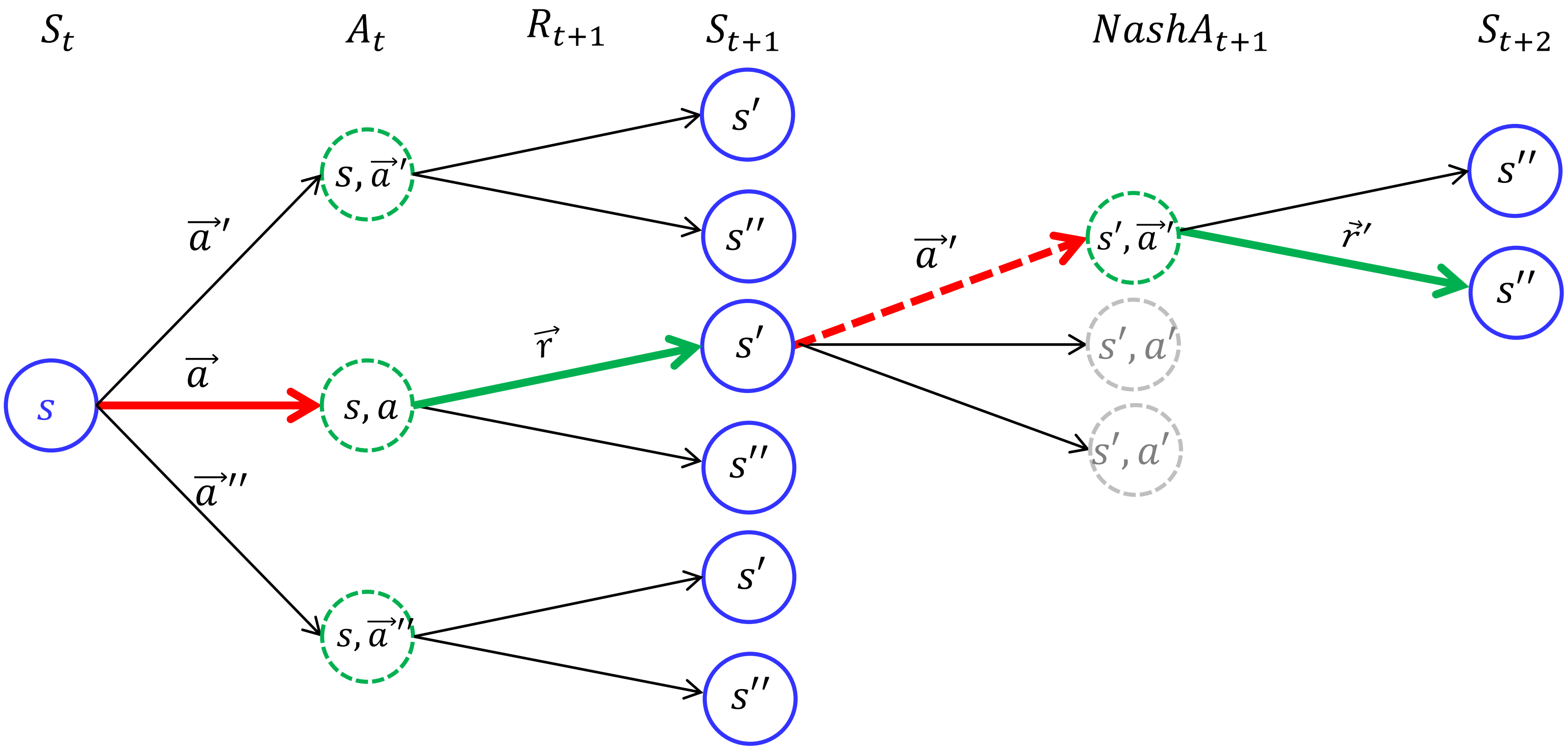



- Choose action $\vec{a} = (a_1, \dots, a_n)$ from $s$ using current Nash Q values $(Q_1(s',\vec{a}), \dots, Q_n(s',\vec{a}))$

$$\vec{a} = \begin{cases} \vec{a}_{\mathrm{NE}} \text{ for } (Q_1(s',\vec{a}), \dots, Q_n(s',\vec{a})) & \text{with prob } 1-\epsilon \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

Any exploration policy can be used

## Nash-Q learning : Procedure




- Take action $\vec{a}' = (a_1', \ldots, a_n')$ given $s'$ and observe $\vec{r}' = (r_1', \ldots, r_n')$ and $s''$

## Nash-Q learning: Convergence

The convergence of this Nash Q is based on three important assumptions:

- **Assumption 1**: Every state $s \in S$ and action $a_k \in A_k$ for $k = 1, \ldots, n$, are visited infinitely often.

- **Assumption 2**: The learning rate $\alpha_t$ satisfies the following conditions for all $s, t, a_1, \ldots, a_n$:
  - $0 \leq \alpha_t(s, a_1, \ldots, a_n) < 1, \sum_t^{\infty} \alpha_t(s, a_1, \ldots, a_n) = \infty, \sum_t^{\infty} [\alpha_t(s, a_1, \ldots, a_n)]^2 < \infty$
  - $\alpha_t(s, a_1, \ldots, a_n) = 0$ if $(s, a_1, \ldots, a_n) \neq (s_t, a_1, \ldots, a_n)$, meaning that agent will only update the Q-values for the present state and actions

- **Assumption 3**: One of the following conditions holds during learning:
  - Condition 1: Every stage game $(Q_1^t(s), \ldots, Q_n^t(s))$, for all $t$ and $s$, has a global optimal point, and agents' payoffs in this equilibrium are used to update their Q-functions
  - Condition 1: Every stage game $(Q_1^t(s), \ldots, Q_n^t(s))$, for all $t$ and $s$, has a saddle point, and agents' payoffs in this equilibrium are used to update their Q-functions

## Minmax-Q learning

- The Minimax-Q algorithm was developed by Littman in 1994 when he adapted the value iteration method of Q-Learning from a single player to a two player zero sum game

- This is for a fully competitive game where players have opposite goals and reward functions ($R1 = -R2$).
  - ➢ Each agent tries to maximize its reward function while minimizing the opponent's.

# Minmax-Q learning

## Multi Agent Q-learning Template

MultiQ(StochastiGame, $f, \gamma, \alpha, T$)

| | |
|---|---|
| Inputs | equilibrium selection function $f$ |
| | discounting factor $\gamma$ |
| | learning rate $\alpha$ |
| | total training time $T$ |
| Outputs | state – value functions $V_i^*$ |
| | action – value functions $Q_i^*$ |
| Initialize | $s, a_1, \ldots, a_n$ and $Q_1, \ldots, Q_n$ |

for $t = 1{:}T$

1. select actions $\vec{a} = (a_1, \ldots, a_n)$in state $s$
2. observe rewards $r_1, \ldots, r_n$ and next state $s'$
3. for $i = 1$ to $n$ (for each agent)
   (a) $V_i(s') = f_i(Q_1(s', \vec{a}), \ldots, Q_n(s', \vec{a}))$
   (b) $Q_i(s, \vec{a}) = (1 - \alpha_i)Q_i(s, \vec{a}) + \alpha_i[r_i + \gamma V_i(s')]$
4. agent choose actions action $a_1', \ldots, a_n'$
5. $s = s', a_1 = a_1', \ldots, a_n = a_n'$
6. adjust learning rate $\alpha = (\alpha_1, \ldots, \alpha_n)$

## Minmax-Q learning

For agent $i = 1:2$

(a) $V_i(s') = f_i(Q_1(s', a_i, a_{-i}), Q_2(s', a_i, a_{-i}),) = \max_{\pi_i(s',\cdot)} \min_{a_{-i} \in A_{-i}} \sum_{a_i \in A_i} Q_i(s', a_i, a_{-i}) \pi_i(s', a_i)$

$= \text{Maxmin}\, Q_i(s')$

(b) $Q_i(s, a_i, a_{-i}) = (1-\alpha) Q_i(s, a_i, a_{-i}) + \alpha[r_i + \gamma V_i(s')]$

$= (1-\alpha) Q_i(s, a_i, a_{-i}) + \alpha[r_i + \gamma\ \text{Maxmin}\, Q_i(s')]$

Action selection strategy:

$$\pi_i(s',\cdot) = \operatorname*{argmax}_{\pi_i(s',\cdot)} \min_{a_{-i} \in A_{-i}} \sum_{a_i \in A_i} Q_i(s', a_i, a_{-i}) \pi_i(s, a_i)$$

- Note that when computing the maxmin value, each agent can consider only its own action value function $Q_i(s', a_i, a_{-i})$
- But, still each agent need to track the action taken by the other agent for updating

## Minmax-Q learning

For agent 1

(a) $V_1(s') = f_1(Q_1(s', a_1, a_2), Q_2(s', a_1, a_2),) = \max_{\pi_1(s',\cdot)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q_1(s', a_1, a_2)\pi_1(s', a_1)$

$= \text{Maxmin } Q_1(s')$

(b) $Q_1(s, a) = (1-\alpha)Q_1(s, a) + \alpha[r_1 + \gamma V_1(s')]$

$= (1-\alpha)Q_1(s, a) + \alpha[r_1 + \gamma \text{ Maxmin } Q_1(s')]$

For agent 2

(a) $V_2(s') = f_1(Q_1(s', a_1, a_2), Q_2(s', a_1, a_2),) = \max_{\pi_2(s',\cdot)} \min_{a_1 \in A_1} \sum_{a_2 \in A_2} Q_2(s', a_1, a_2)\pi_2(s', a_2)$

$= \text{Maxmin } Q_2(s')$

(b) $Q_2(s, a) = (1-\alpha)Q_2(s, a) + \alpha[r_1 + \gamma V_2(s')]$

$= (1-\alpha)Q_2(s, a) + \alpha[r_1 + \gamma \text{ Maxmin } Q_2(s')]$

## Minmax-Q learning

- Because the property of a zero sum game, $Q_2(s', a_1, a_2) = -Q_1(s', a_1, a_2)$

$$\max_{\pi_2(s',\cdot)} \min_{a_1 \in A_1} \sum_{a_2 \in A_2} Q_2(s', a_1, a_2)\pi_2(s', a_2) = \max_{\pi_2(s',\cdot)} \min_{a_1 \in A_1} \sum_{a_2 \in A_2} -Q_1(s', a_1, a_2)\pi_2(s', a_2)$$

$$= \min_{\pi_2(s',\cdot)} \max_{a_1 \in A_1} \sum_{a_2 \in A_2} Q_1(s', a_1, a_2)\pi_2(s', a_2) = \text{minmax}\, Q_1(s')$$

- Therefore, player 2's Q-function can be updated using $Q_1(s, a)$

$$\text{(b) } Q_2(s, a) = (1-\alpha)Q_2(s, a) + \alpha[r_1 + \gamma V_2(s')]$$

$$= (1-\alpha)Q_2(s, a) + \alpha[r_1 + \gamma \text{ Maxmin}\, Q_2(s')]$$

$$\text{(b) } -Q_1(s, a) = (1-\alpha)\{-Q_1(s, a)\} + \alpha[r_1 + \gamma V_2(s')]$$

$$= (1-\alpha)\{-Q_1(s, a)\} + \alpha[r_1 + \gamma \text{ minmax}\, Q_1(s')]$$

- This result concludes that in Minmax-Q learning, we can only keep updating Q-function for player 1, $Q_1(s, a) \rightarrow Q(s, a)$

## Minmax-Q learning

Updating rule for agent 1

$$\text{(a) } V(s') = f_1\big(Q(s',a_1,a_2)\big) = \max_{\pi_1(s',\cdot)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q(s',a_1,a_2)\pi_1(s,a_1) = \text{Maxmin}\, Q(s')$$

$$\text{(b) } Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r_1 + \gamma V(s')]$$

$$= (1-\alpha)Q_1(s,a) + \alpha[r_1 + \gamma \,\text{Maxmin}\, Q(s')]$$

Action selection rule for agent 1

$$\pi_1(s',\cdot) = \operatorname*{argmax}_{\pi_1(s',\cdot)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q(s',a_1,a_2)\pi_1(s',a_1)$$

Updating rule for agent 2

Agent 2's $Q$ function $Q_2(s,a) = -Q(s,a)$

Action selection rule for agent 2

$$\pi_2(s',\cdot) = \operatorname*{argmax}_{\pi_2(s',\cdot)} \min_{a_1 \in A_1} \sum_{a_2 \in A_2} -Q(s',a_1,a_2)\pi_2(s,a_2)$$

$$= \operatorname*{argmin}_{\pi_2(s',\cdot)} \max_{a_1 \in A_1} \sum_{a_2 \in A_2} Q(s',a_1,a_2)\pi_2(s,a_2)$$

# Minmax-Q learning Algorithm

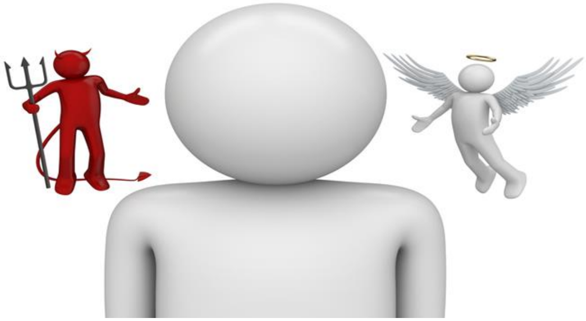
MultiQ(StochastiGame, $f, \gamma, \alpha, T$)

| | |
|---|---|
| Inputs | equilibrium selection function $f$ |
| | discounting factor $\gamma$ |
| | learning rate $\alpha$ |
| | total training time $T$ |
| Outputs | state − value functions $V_i^*$ |
| | action − value functions $Q_i^*$ |
| Initialize | $s, a_1, \ldots, a_n$ and $Q_1, \ldots, Q_n$ |

for $t = 1{:}T$

1. select actions $a_1, \ldots, a_n$ in state $s$
2. observe rewards $r_1, \ldots, r_n$ and next state $s'$
3. for $i = 1$ to $n$ (for each agent)
   (a) $V_i(s') = f_i\big(Q_1(s', a), \ldots, Q_n(s', a)\big) = \text{Minmax}Q_i(s')$
   (b) $Q_i(s, a) = (1 - \alpha_i)Q_i(s, a) + \alpha_i[r_i + \gamma V_i(s')]$
4. agent choose actions action $a_1', \ldots, a_n'$
5. $s = s', a_1 = a_1', \ldots, a_n = a_n'$
6. adjust learning rate $\alpha = (\alpha_1, \ldots, \alpha_n)$

## Friend-or-For Q learning




- This algorithm was developed by Littman (1998) and tries to fix some of the convergence problems of Nash-Q Learning

- The main concern lies within assumption 3, where every stage game needs to have either a global optimal point or a saddle point.
  - These restrictions cannot be guaranteed during learning.

- To alleviate this restriction, this new algorithm is built to always converge by changing the update rules depending on the opponent.
  - The learning agent has to classify the other agent as "**friend**" or "**foe**".
  - Player $i$'s friends are assumed to work together to maximize player $i$'s value
  - Player $i$'s foes are assumed to work together to minimize player $i$'s value

- Thus, $n$-player general-sum stochastic game can be treated as a two-player zero-sum game with an extended action set.

## Friend-or-For Q learning

### Multi Agent Q-learning Template

MultiQ(StochastiGame, $f, \gamma, \alpha, T$)

Inputs equilibrium selection function $f$

discounting factor $\gamma$

learning rate $\alpha$

total training time $T$

Outputs state – value functions $V_i^*$

action – value functions $Q_i^*$

Initialize $s, a_1, \ldots, a_n$ and $Q_1, \ldots, Q_n$

for $t = 1{:}T$

1. select actions $\vec{a} = (a_1, \ldots, a_n)$in state $s$
2. observe rewards $r_1, \ldots, r_n$ and next state $s'$
3. for $i = 1$ to $n$ (for each agent)

   (a) $V_i(s') = f_i(Q_1(s', \vec{a}), \ldots, Q_n(s', \vec{a}))$

   (b) $Q_i(s, \vec{a}) = (1 - \alpha_i)Q_i(s, \vec{a}) + \alpha_i[r_i + \gamma V_i(s')]$
4. agent choose actions action $a_1', \ldots, a_n'$
5. $s = s', a_1 = a_1', \ldots, a_n = a_n'$
6. adjust learning rate $\alpha = (\alpha_1, \ldots, \alpha_n)$

## Friend-or-For Q learning

For agent $i$

$$(a)\ V_i(s') = f_i\big(Q_1(s',\vec{a},\vec{o}),\dots,Q_i(s',\vec{a},\vec{o}),\dots,Q_n(s',\vec{a},\vec{o})\big)$$

$$= \max_{\pi_1(s',\cdot),\dots,\pi_{n_1}(s',\cdot)} \ \min_{o_1,\dots,o_{n_2}\in O_1\times\cdots\times O_{n_2}} \sum_{a_i\in A_i} Q_i(s',\vec{a},\vec{o})\pi_1(s,a_1)\cdots\pi_{n_1}(s,a_{n_1})$$

$\vec{a} = (a_1,\dots,a_{n_1})$: actions for the friends agents

$\vec{o} = (o_1,\dots,o_{n_2})$: actions for the foe agents

$$(b)\ Q_i(s,\vec{a},\vec{o}) = (1-\alpha)Q_i(s,\vec{a},\vec{o}) + \alpha[r_i + \gamma V_i(s')]$$

$$= (1-\alpha)Q_i(s,\vec{a},\vec{o}) + \alpha[r_i + \gamma\ FoF\ Q_i(s')]$$

# Friend-or-For Q learning

**For two player case (described in terms of player 1)**

(a) $V_1(s') = f_1\big(Q_1(s', a_1, a_2), Q_2(s', a_1, a_2)\big)$

$$= \begin{cases} \max\limits_{a_1 \in A_1, a_2 \in A_2} Q_1(s', a_1, a_2) & \textbf{If other player is fried:} \\ \max\limits_{\pi_1(s', \cdot)} \min\limits_{a_2 \in A_2} \sum\limits_{a_i \in A_i} Q_1(s', a_1, a_2) \pi_1(s, a_1) & \textbf{If other player is foe:} \end{cases}$$

(b) $Q_1(s, a_1, a_2) = (1-\alpha) Q_1(s, a_1, a_2) + \alpha[r_i + \gamma V_1(s')]$

$$= (1-\alpha) Q_1(s, a_1, a_2) + \alpha[r_i + \gamma FoF\, Q_1(s')]$$

## Friend-or-For Q learning

FoFQ(StochastiGame, $f, \gamma, \alpha, T$)

Inputs equilibrium selection function $f$ = Friend or Foe

discounting factor $\gamma$

learning rate $\alpha$

total training time $T$

Outputs state − value functions $V_i^*$

action − value functions $Q_i^*$

Initialize $s, a_1, \dots, a_n$ and $Q_1, \dots, Q_n$

for $t = 1{:}T$

1. select actions $\vec{a} = (a_1, \dots, a_n)$in state $s$
2. observe rewards $r_1, \dots, r_n$ and next state $s'$
3. for $i = 1$ to $n$ (for each agent)

   (a) $V_i(s') = FoF(Q_1(s', \vec{a}), \dots, Q_n(s', \vec{a}))$

   (b) $Q_i(s, \vec{a}) = (1 - \alpha_i)Q_i(s, \vec{a}) + \alpha_i[r_i + \gamma V_i(s')]$
4. agent choose actions action $a_1', \dots, a_n'$
5. $s = s', a_1 = a_1', \dots, a_n = a_n'$
6. adjust learning rate $\alpha = (\alpha_1, \dots, \alpha_n)$

## Correlated-Q learning

- Correlated equilibrium

| | Go | Wait |
|---|---|---|
| Go | -100, -100 | 10, 0 |
| Wait | 0, 10 | -10, -10 |

Traffic game

- What is the natural solution here?
  - A traffic light: a fair randomizing device that tells one of the agents to go and the other to wait.

- Benefits:
  - the negative payoff outcomes are completely avoided
  - fairness is achieved
  - the sum of social welfare exceeds that of mixed Nash equilibrium

# Correlated-Q learning

## Definition

A joint probability distribution $\pi \in \Delta(A)$ is a correlated equilibrium of a finite game if and only if

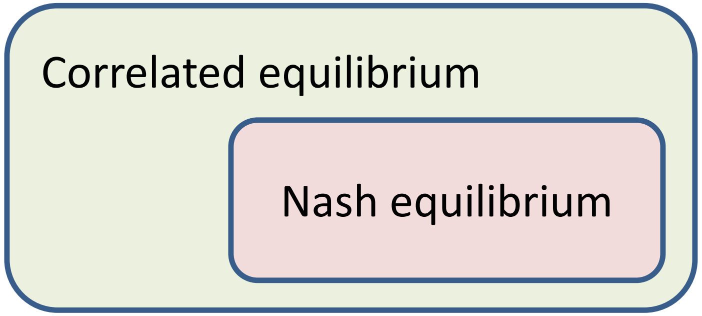
$$\sum_{a_{-i} \in A_{-i}} \pi(a) u_i(a_i, a_{-i}) \geq \sum_{a_{-i} \in A_{-i}} \pi(a) u_i(a_i', a_{-i})$$

For all players $i$, all $s_i \in S_i$, $t_i \in S_i$ such that $a_i' \in A_i$

## Theorem (Correlated equilibrium)

For every Nash equilibrium $\sigma^*$ there exists a corresponding correlated equilibrium $\sigma$

- Correlated equilibrium is a strictly weaker notion than Nash



## Computing correlated equilibria : Example

| | L | R |
|---|---|---|
| T | $6, 6$ | $2, 8$ |
| B | $8, 2$ | $0, 0$ |

- Each correlated equilibrium corresponds to a probability distribution $(a, b, c, d)$ over the possible pairs of actions, $\{(T, L), (T, R), (B, L), (B, R)\}$.
- The conditions needed to be correlated equilibrium, in addition to $(a, b, c, d)$ being a probability distribution, are

$$(T \to B) \quad 6a + 2b \geq 8a + 0b$$

$$(B \to T) \quad 8c + 0d \geq 6c + 2d$$

$$(L \to R) \quad 6a + 2c \geq 8a + 0c$$

$$(R \to L) \quad 8b + 0d \geq 6b + 2d$$

where, for example, the equation for $(T \to B)$ insures that the first player would not receive a higher expected payoff by using $B$ whenever told to play $T$.

- The equations reduce to $(a, b, c, d)$ is a probability vector such that $a \leq b, a \leq c. d \leq b,$ and $d \leq c$.

## Multi Agent Q-learning Template

MultiQ(StochastiGame, $f, \gamma, \alpha, T$)

| | |
|---|---|
| Inputs | equilibrium selection function $f$ |
| | discounting factor $\gamma$ |
| | learning rate $\alpha$ |
| | total training time $T$ |
| Outputs | state − value functions $V_i^*$ |
| | action − value functions $Q_i^*$ |
| Initialize | $s, a_1, \dots, a_n$ and $Q_1, \dots, Q_n$ |

for $t = 1{:}T$

1. select actions $\vec{a} = (a_1, \dots, a_n)$in state $s$
2. observe rewards $r_1, \dots, r_n$ and next state $s'$
3. for $i = 1$ to $n$ (for each agent)
   (a) $V_i(s') = f_i(Q_1(s', \vec{a}), \dots, Q_n(s', \vec{a}))$
   (b) $Q_i(s, \vec{a}) = (1 - \alpha_i)Q_i(s, \vec{a}) + \alpha_i[r_i + \gamma V_i(s')]$
4. agent choose actions action $a_1', \dots, a_n'$
5. $s = s', a_1 = a_1', \dots, a_n = a_n'$
6. adjust learning rate $\alpha = (\alpha_1, \dots, \alpha_n)$

## Correlated-Q learning

For agent $i$

$$\text{(a) } V_i(s') = f_i\Big(\underbrace{Q_1(s',\vec{a}), \dots, Q_i(s',\vec{a}), \dots, Q_n(s',\vec{a})}_{\text{Q values for agents } i = 1:n}\Big) = \text{CE } Q_i(s')$$

Correlated equilibrium value

$$\text{(b) } Q_i(s,\vec{a}) = (1-\alpha)Q_i(s,\vec{a}) + \alpha[r_i + \gamma V_i(s')]$$

$$= (1-\alpha)Q_i(s,\vec{a}) + \alpha[r_i + \gamma \text{ CE } Q_i(s')]$$

## Variants of Correlated-Q learning

How to compute $\text{CE}\, Q_i(s')$ ?

1. First compute the Correlated equilibrium $\pi(s',\vec{a})$ by solving the following constrain satisfaction problem

$$\sum_{\vec{a}\in A|a_i\in\vec{a}} \pi(s,\vec{a})Q_i(s',\vec{a}) \geq \sum_{\vec{a}\in A|a_i\in\vec{a}} \pi(s,\vec{a})Q_i(s',a_i',a_{-i}), \forall i \in N, \forall a_i, a_i' \in A_i \quad (1)$$

$$\pi(s',\vec{a}) > 0,\ \forall \vec{a} \in A \quad (2)$$

$$\sum_{\vec{a}\in A} \pi(s',\vec{a}) = 1 \quad (3)$$

- Variables: $\pi(s',\vec{a})$, constants: $\{Q_i(s',\vec{a}), \ldots, Q_i(s',\vec{a}), \ldots, Q_n(s',\vec{a})\}$

2. With the correlated equilibrium strategy $\pi(s,\vec{a})$, compute the correlation equilibrium value $\text{CE}\, Q_i(s')$ for player $i$ at state $s$ as

$$\text{CE}\, Q_i(s') = \sum_{\vec{a}\in A} \pi(s',\vec{a})Q_i(s',\vec{a})$$

## Variants of Correlated-Q learning

- The difficulty in learning equilibria in Markov games stems from the equilibrium selection problem:
  - ➢ How can multiple agents select among multiple equilibria?
- We introduce four variants of correlated-Q learning, which determine a unique Eq.
  - ➢ Resolves the equilibrium selection problem with its respective choice of objective function

## Variants of Correlated-Q learning

- Utilitarian equilibrium: an equilibrium which maximizes the sum of the expected payoffs of the players:

$$\sigma \in \underset{\sigma \in \mathrm{CE}}{\operatorname{argmax}} \sum_{i \in N} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})$$

- Egalitarian equilibrium : an equilibrium which maximizes the minimum expected payoff of a player

$$\sigma \in \underset{\sigma \in \mathrm{CE}}{\operatorname{argmax}} \min_{i \in N} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})$$

- Republican equilibrium : an equilibrium which maximizes the maximum expected payoff of a player

$$\sigma \in \underset{\sigma \in \mathrm{CE}}{\operatorname{argmax}} \max_{i \in N} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})$$

- Libertarian $i$ equilibrium: an equilibrium which maximizes the maximum of each individual player $i$'s rewards: $\sigma = \prod_i \sigma^i$, where

$$\sigma_i \in \underset{\sigma \in \mathrm{CE}}{\operatorname{argmax}} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})$$

# Correlated-Q learning

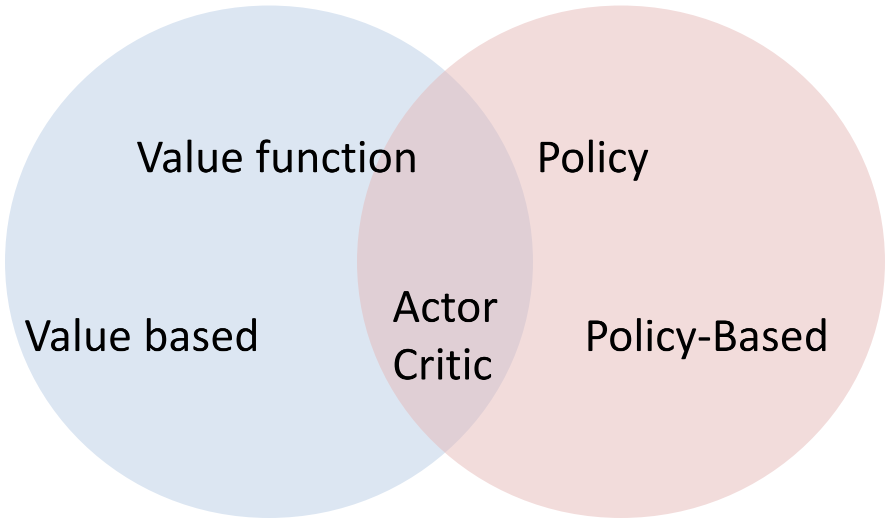
FoFQ(StochastiGame, $f, \gamma, \alpha, T$)

| | |
|---|---|
| Inputs | equilibrium selection function $f$ = Correlated eq. |
| | discounting factor $\gamma$ |
| | learning rate $\alpha$ |
| | total training time $T$ |
| Outputs | state − value functions $V_i^*$ |
| | action − value functions $Q_i^*$ |
| Initialize | $s, a_1, \ldots, a_n$ and $Q_1, \ldots, Q_n$ |

for $t = 1{:}T$

1. select actions $\vec{a} = (a_1, \ldots, a_n)$in state $s$
2. observe rewards $r_1, \ldots, r_n$ and next state $s'$
3. for $i = 1$ to $n$ (for each agent)
   (a) $V_i(s') = \mathrm{CE}(Q_1(s', \vec{a}), \ldots, Q_n(s', \vec{a}))$
   (b) $Q_i(s, \vec{a}) = (1 - \alpha_i) Q_i(s, \vec{a}) + \alpha_i [r_i + \gamma V_i(s')]$
4. agent choose actions action $a_1', \ldots, a_n'$
5. $s = s', a_1 = a_1', \ldots, a_n = a_n'$
6. adjust learning rate $\alpha = (\alpha_1, \ldots, \alpha_n)$

## Types of reinforcement learning



## Policy Objective Function

- Stochastic policy can be parameterized as

$$\pi_\theta(s,a) = P(a|s,\theta)$$

- We can measure the quality of the policy $\pi_\theta$ using the policy objective functions $J(\theta)$:
  - In episodic environments we can use the start value:

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

  - In continuing environments we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

  - Average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_s \pi_\theta(s,a) R(s,a)$$

✓ $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for $\pi_\theta$

## Policy Gradient

- The policy gradient $\nabla_\theta J(\theta)$ is given as

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- The parameters for the policy $\pi_\theta(s, a) = P(a|s, \theta)$ can be updated using gradient ascent

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

where $\alpha$ is a step-size parameter

## Policy Gradient Theorem (Stochastic policy)

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim d(s)$
  - Terminating after one time-step with reward $r = R(s, a)$

- Use likelihood ratios to compute the policy gradient

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r] = \sum_s d(s) \sum_s \pi_\theta(s, a) R(s, a)$$

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_s d(s) \sum_s \nabla \pi_\theta(s, a) R(s, a) \\ &= \sum_s d(s) \sum_s \pi_\theta(s, a) \nabla \log \pi_\theta(s, a)\, R(s, a) \\ &= \mathbb{E}_{\pi_\theta}[\nabla \log \pi_\theta(s, a)\, R(s, a)] \end{aligned}$$

$$\boxed{\nabla \pi_\theta(s, a) = \pi_\theta(s, a) \frac{\nabla \pi_\theta(s, a)}{\pi_\theta(s, a)} = \pi_\theta(s, a) \nabla \log \pi_\theta(s, a)}$$

## Policy Gradient Theorem (Stochastic policy)

**Theorem**

For any differentiable policy $\pi_\theta(s, a)$,

For any of the policy objective functions $J = J_1, J_{avR}, or \frac{1}{1-\gamma} J_{avV}$

The (stochastic) policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a)\, Q^{\pi_\theta}(s, a)]$$

- Expectation over (state and action)
- Policy gradient is (score function)×(action-value function)

# Monte-Carlo Policy Gradient (Reinforce)

- Using policy gradient theorem
- Using return $v_t$ as an unbiased sample of $Q^{\pi_\theta}(s, a)$, Update parameters by stochastic gradient ascent:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla \log \pi_\theta(s,a)\, Q^{\pi_\theta}(s,a)\right]$$
$$\sim \nabla \log \pi_\theta(s,a)\, v_t$$

$$\Delta\theta_t = \alpha \nabla \log \pi_\theta(s,a)\, v_t$$

**function REINFORCE**
    Initialise $\theta$ arbitrarily
    **for** each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
        **for** $t = 1$ to $T - 1$ **do**
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        **end for**
    **end for**
    **return** $\theta$
**end function**

## Reducing Variance Using a Critic

- Monte-Carlo policy gradient has high variance, especially when the episode is long
- Use a critic to estimate the action-value function (Bootstrap)

$$Q^W(s,a) \approx Q^{\pi_\theta}(s,a)$$

- Actor-critic algorithms follow an approximate policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla \log \pi_\theta(s,a)\, Q^{\pi_\theta}(s,a)\right]$$

$$\sim \nabla \log \pi_\theta(s,a)\, v_t \qquad \text{(Sample trajectory)}$$

$$\sim \nabla \log \pi_\theta(s,a)\, Q^W(s,a) \qquad \text{(Bootstrap)}$$

- Actor-critic algorithms maintain two sets of parameters
  - **Critic**: Update action-value function parameters
  - **Actor**: Update policy parameters $\theta$, in direction suggested by critic

# Actor-Critic Algorithm

**function** QAC
  Initialise $s$, $\theta$
  Sample $a \sim \pi_\theta$
  **for** each step **do**
    Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
    Sample action $a' \sim \pi_\theta(s', a')$
    $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
    $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
    $w \leftarrow w + \beta \delta \phi(s, a)$
    $a \leftarrow a', s \leftarrow s'$
  **end for**
**end function**

- Actor: Updates $\theta$ by policy gradient
- Critic: Updates w by linear TD(0)

## Recall Policy Hill Climbing (PHC) for Repeated Game

- PHC algorithm has been discussed as a way to solve a repeated matrix game

**Algorithm Policy hill − climbing (PHC) algorithm for agent *i***

**Initialize**

learning rate $\alpha \in (0,1], \delta \in (0,1]$

discunt factor $\gamma \in (0,1)$

exploration rate $\epsilon$

$Q_i(a_i) \leftarrow 0$ and $\pi_i(a_i) \leftarrow \frac{1}{|A_i|}\ \forall a_i \in A_i$

**Repeat**

(a) select an action $a_i$ according to the straegy $\pi(a_i)$ with some exploration rate $\epsilon$

(b) observe the immediate reward $r_i$

(c) update $Q$ values:

$$Q_i(a_i) = (1-\alpha)Q_i(a_i) + \alpha\left(r_i + \gamma \max_{a_i'} Q_i(a_i')\right)$$

(d) Update the strategy $\pi_i(a_i)$ and constrain it to a legal probability distribution

$$\pi_i(a_i) = \pi_i(a_i) + \begin{cases} \delta & if\ a_i = \max_{a_i'} Q_i(a_i') \\ -\frac{\delta}{|A_i|-1} & otherwise \end{cases}$$

## Policy Hill Climbing (PHC) for Stochastic Game

- We will expand the PHC algorithm so that it can be used for a general sum stochastic game

**Algorithm** **Policy hill – climbing (PHC) algorithm for agent *i***

**Initialize**

learning rate $\alpha \in (0,1], \delta \in (0,1]$

discunt factor $\gamma \in (0,1)$

exploration rate $\epsilon$

$Q_i(s, a_i) \leftarrow 0$ and $\pi_i(s, a_i) \leftarrow \frac{1}{|A_i|}\ \forall a_i \in A_i$

**Repeat**

(a) select an action $a_i$ according to the straegy $\pi(s, a_i)$ with some exploration rate $\epsilon$

(b) observe the immediate reward $r_i$

(c) update $Q$ values:

$$Q_i(s, a_i) = (1-\alpha)Q_i(s, a_i) + \alpha\left(r_i + \gamma \max_{a_i'} Q_i(s, a_i')\right)$$

(d) Update the strategy $\pi_i(s, a_i)$ and constrain it to a legal probability distribution

$$\pi_i(s, a_i) = \pi_i(s, a_i) + \begin{cases} \delta & if\ a_i = \max_{a_i'} Q_i(s, a_i') \\ -\frac{\delta}{|A_i| - 1} & otherwise \end{cases}$$

## The WoLF-Policy Hill Climbing (PHC) for Stochastic Game

- The WoLF-PHC algorithm is an extension of the PHC algorithm
  - WoLF(win-or-learn-fast) allows variable learning rate→ faster convergence

(c) update $Q$ values:

$$Q_i(s,a_i) = (1-\alpha)Q_i(s,a_i) + \alpha\left(r_i + \gamma \max_{a_i'} Q_i(s,a_i')\right)$$

update estimate of average policy $\bar{\pi}(s,a')$ :

$$C(s) \leftarrow C(s) + 1$$

$$\forall a' \in A_i, \qquad \bar{\pi}_i(s,a') \leftarrow \pi_i(s,a') + \frac{1}{C(s)}\left(\pi_i(s,a') - \bar{\pi}_i(s,a')\right)$$

(d) Update the strategy $\pi_i(s,a_i)$ and constrain it to a legal probability distribution

$$\pi_i(s,a_i) = \pi_i(s,a_i) + \begin{cases} \delta & \text{if } a_i = \max_{a_i'} Q_i(s,a_i') \\ -\dfrac{\delta}{|A_i|-1} & \text{otherwise} \end{cases}$$

WoLF

where

$$\delta = \begin{cases} \delta_w & \text{if } \sum_{a_i} \pi_i(s,a_i)Q_i(s,a_i) > \sum_{a_i} \bar{\pi}_i(s,a_i)Q_i(s,a_i) \\ \delta_l & \text{otherwise} \end{cases}$$

## The WoLF-Policy Hill Climbing (PHC) for Stochastic Game

- This algorithm has two different learning rates
  - ✓ When the algorithm is wining
  - ✓ When the algorithm is losing
- The losing learning rate $\delta_l$ is larger than wining learning rate $\delta_w$
  - When an agent is losing, it learns faster than when it is winning
  - This causes the agent to adapt quickly to the changes in the strategies of the other agents when it is doing more poorly than expected
  - Learns cautiously when it is doing better than expected
  - Also gives the other agents the time to adapt to the agent's strategy changes
- The different between the average strategy and the current strategy is used as a criterion to decide when the algorithm wins or loses
- The WoLF-PHC algorithm exhibits the property of convergence as it makes the agent converge to one of its Nash equilibria (no proof, but empirical results)
- The algorithm is also a rational learning algorithm as it makes the agent converge to its optimal strategy when its opponent plays a stationary strategy

## Comparison of MARL algorithms

| Algorithm | Applicability | Rationality | Convergence | Required info |
| --- | --- | --- | --- | --- |
| **Minmax-Q** | Zero-sum SGs | NO | YES | Other agent's action, rewards |
| **Nash-Q** | general sum SGs | NO | YES | Other agent's action, rewards |
| **Friend-or-foe Q** | general sum SGs | NO | YES | Other agent's action, rewards |
| **Correlated-Q** | general sum SGs | NO | YES | Other agent's action, rewards |
| **WoLF-PHC** | General sum SGs | YES | NO | Own action, reward |

- The WoLF-PHC algorithm does not need to observe the other player's strategies and actions
- The WoLF-PHC does not require to solve Linear programming nor quadratic programming