Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL)

Mario Martin

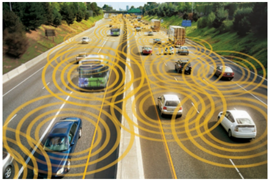
CS-UPC

April 9, 2021

Motivation and problems

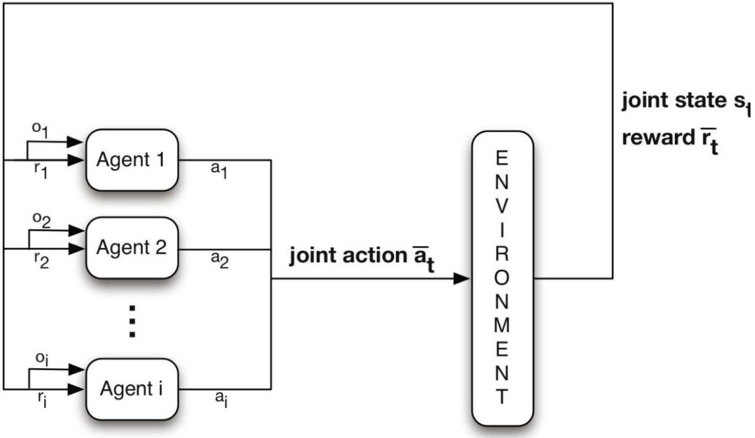
Multi-agent RL motivation

- All cases we have seen assume the agent is the only one that executes actions in the environment, but not always the case.
- Some examples:



- More examples: Games in general, finances, negotiation, home assistance, multi-robot rescue, wireless networks, etc.
- In cases where there are also other agents interacting with the environment, can we learn? Is the problem different?

Multi-agent RL setting



Multi-agent RL differences with single RL agent

- You take actions but other agents also take actions that change the state
- Naive idea: Let each agent learn its own policy assuming that the other agents are part of the environment (no social awareness).
- Surprisingly this works in some cases, but not in the most interesting cases.
- Why? *Non-stationarity*: $T(s, a, s')$ is not constant because it also depend on actions performed by other agents (no MDP)!
- You need to know the actions the other agents will do in order to return to the markovian property.

Multi-agent RL problems (1)

- *Agent returns are correlated and cannot be maximized independently.* So we need actions executed by the other agents to compute Q values and also for choosing the action (policy computation)

$$Q_i : S \; x \; A^n \longrightarrow \mathcal{R}_i$$

- Usually, agent does not know the actions other agents will take. In this case, several possibilities.
  - **Predict/Infer** actions of other agents (in this case you need to know the perception of other agents)

Multi-agent RL problems (1)

- *Agent returns are correlated and cannot be maximized independently.* So we need actions executed by the other agents to compute Q values and also for choosing the action (policy computation)

$$Q_i : S \; x \; A^n \longrightarrow \mathcal{R}_i$$

- Usually, agent does not know the actions other agents will take. In this case, several possibilities.
  - **Predict/Infer** actions of other agents (in this case you need to know the perception of other agents)
  - **Sharing** of: perceptions / policies / actions / rewards obtained / ER

Multi-agent RL problems (1)

- *Agent returns are correlated and cannot be maximized independently.* So we need actions executed by the other agents to compute Q values and also for choosing the action (policy computation)

$$Q_i : S \text{ x } A^n \longrightarrow \mathcal{R}_i$$

- Usually, agent does not know the actions other agents will take. In this case, several possibilities.
  - **Predict/Infer** actions of other agents (in this case you need to know the perception of other agents)
  - **Sharing** of: perceptions / policies / actions / rewards obtained / ER
  - **Communication** between agents (orders, perceptions, own action executed)

Multi-agent RL problems (1)

- *Agent returns are correlated and cannot be maximized independently.* So we need actions executed by the other agents to compute Q values and also for choosing the action (policy computation)

$$Q_i : S \ x \ A^n \longrightarrow \mathcal{R}_i$$

- Usually, agent does not know the actions other agents will take. In this case, several possibilities.
  - **Predict/Infer** actions of other agents (in this case you need to know the perception of other agents)
  - **Sharing** of: perceptions / policies / actions / rewards obtained / ER
  - **Communication** between agents (orders, perceptions, own action executed)

- *Curse of dimensionality*. Prediction should be of actions of **all** agents (and each agent several actions). This scales exponentially.
  - Fortunately in same cases no needed (factorization of reward function, graph approaches, etc.)

Multi-agent RL problems (2)

- Moreover, while learning... Other agents may learn too!!
  - We have to adapt continually!
  - "Moving target problem" central issue in multiagent learning

Multi-agent RL problems (2)

- Moreover, while learning... Other agents may learn too!!
  - We have to adapt continually!
  - "Moving target problem" central issue in multiagent learning
- Ideally agents should learn in a **decentralized** way (every agent working on its own).
  - A popular solution is to *train* agents in a *centralized* way and later *apply* the policies learned in a *decentralized* way.
  - This solve both problems: the prediction and the moving target problem at the same time

Multi-agent RL problems (3)

- **Exploration** is key in RL, but in MARL this can **destabilize the learning** and confuse other agents. They expect you to do something but you are exploring!
- Possible **miss-match** between individual rewards and collective goodness (f.i. the tragedy of commons).

MARL Mathematical formulation

Mathematical formulation: Game theory

- Usually MARL problems are formalized in the Game Theory framework.
- Game Theory is well established and allows to understand theoretically the MARL problem
- It is used as a reference specially in few agents cases.
- Kinds of games from simple to complex:
  1. Normal-form game: one-shot games
  2. Repeated game: game repeated several times (so we have history)
  3. Stochastic game: generalization to MDP where state changes

Subsection 1

Normal-form games

Normal-form games

Normal-form games consists of:

- Finite set of agents $i \in \mathcal{N} = \{1, \cdots, n\}$
- Each agent $i \in N$ has a set of actions $A_i \in \{a_1, a_2, \cdots\}$
- Set of joint actions $A = a_1 \times a_2 \times \cdots \times a_n$
- Rewards function $r_i : A \to \mathbb{R}$, where $A = A_1 \times \cdots \times A_n$

Each agent i selects policy $\pi_i : A_i \to [0, 1]$, takes action $a_i \in A_i$ with probability $\pi_i(a_i)$, and receives reward $r_i(a_1, \cdots, a_n)$. Joint action is a.

Given policy profile $(\pi_1, \cdots, \pi_n)$, *expected* reward to i is:

$$R_i(\pi_1, \cdots, \pi_n) = \sum_{a \in A} \pi_1(a_1) * \pi_2(a_2) * \cdots \pi_n(a_n) * r_i(a)$$

Agents selects policy to maximise their expected reward.

Mathematical formulation: Game theory

- Normal-form games are summarized by Payoff tables.
- Example of a payoff table game for 2 players with two actions (A, B) playable:

		Player 2	
		A	B
Player 1	A	(x, y)	(x, y)
	B	(x, y)	(x, y)

- In red, actions playable by Player 1 and rewards for each joint action. In blue the same for Player 2.

Examples

- Rock-Paper-Scissors:

		Player 2		
		R	P	S
Player 1	R	$(0,0)$	$(-1,1)$	$(1,-1)$
	P	$(1,-1)$	$(0,0)$	$(-1,1)$
	S	$(-1,1)$	$(1,-1)$	$(0,0)$

- Prisoner's dilemma:

		Player 2	
		C	D
Player 1	C	$(-1,-1)$	$(-5,0)$
	D	$(0,-5)$	$(-3,-3)$

Examples

- Chicken's game:

$$\begin{array}{c|c|c} & \multicolumn{2}{c}{\text{Player 2}} \\ \text{Player 1} & S & T \\ \hline S & (0,0) & (7,2) \\ T & (2,7) & (6,6) \end{array}$$

- Coordination game:

$$\begin{array}{c|c|c} & \multicolumn{2}{c}{\text{Player 2}} \\ \text{Player 1} & A & B \\ \hline A & (0,0) & (10,10) \\ B & (10,10) & (0,0) \end{array}$$

Mathematical formulation: Game theory

Classification of games:

- **Cooperative**: Agents cooperate to achieve a goal
  - Particular case: Shared team reward
- **Competitive**: Agents compete against each other
  - Particular case: Zero-sum games
  - Individual opposing rewards
- **Neither**: Agents maximize their utility which may require cooperating and/or competing
  - General-sum games

Learning goals for a pay-off matrix

- Learning is to improve performance via experience
  - But what is goal (end-result) of learning process?
  - How to measure success of learning?
- Many learning goals proposed:
  - Minimax/Nash/correlated equilibrium
  - Pareto-optimality
  - Social welfare & fairness
  - No-regret
  - ...

Learning goals for a pay-off matrix

- Learning goal will depend on the kind of pay-off matrix:
- For instance, in a competitive two-player zero-sum game where $u_i = -u_j$[1]
  - e.g. Rock-Paper-Scissors, Chess
- Utility that can be guaranteed against **worst-case** opponent
- Policy profile (π_i, π_j) is **maximin/minimax** profile if:

$$U_i(\pi_i, \pi_j) = \max_{\pi_i'} \min_{\pi_j'} U_i\left(\pi_i', \pi_j'\right) = \min_{\pi_j'} \max_{\pi_j'} U_i\left(\pi_i', \pi_j'\right) = -U_j(\pi_i, \pi_j)$$

[1] I change notation sometimes. Here u utility can be read also as reward r.

Learning goals for a pay-off matrix

- **Nash equilibrium**: When no unilaterally change in action help to improve reward for any agent
  - Every finite game has a *mixed*[2] (probabilistic) strategy (policy) Nash equilibrium
  - Achievable with BestResponse (BR): the strategy with highest payoff for a player, *given* knowledge of the other players' strategies
- Has become standard solution in game theory
- Generalization of minimax: In two-player zero-sum game, minimax is same as NE
- Solutions to Chicken's game, Coordination game, Prisoner's dilemma and Rock-Paper-Scissors.

[2]Versus *pure* (deterministic).

Examples

- Chicken's game:

$$\begin{array}{c|c|c} & S & T \\ \hline S & (0,0) & (7,2) \\ \hline T & (2,7) & (6,6) \end{array}$$

Player 1 (rows), Player 2 (columns)

- Coordination game:

$$\begin{array}{c|c|c} & A & B \\ \hline A & (0,0) & (10,10) \\ \hline B & (10,10) & (0,0) \end{array}$$

Player 1 (rows), Player 2 (columns)

Examples

- Prisoner's dilemma:

		Player 2	
		C	D
Player 1	C	$(-1,-1)$	$(-5,0)$
	D	$(0,-5)$	$(-3,-3)$

- Rock-Paper-Scissors:

		Player 2		
		R	P	S
	R	$(0,0)$	$(-1,1)$	$(1,-1)$
Player 1	P	$(1,-1)$	$(0,0)$	$(-1,1)$
	S	$(-1,1)$	$(1,-1)$	$(0,0)$

Learning goals for a pay-off matrix

- Has become standard solution in game theory...

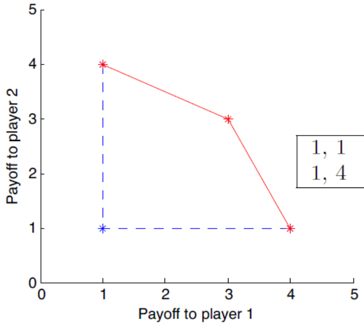
Learning goals for a pay-off matrix

- Has become standard solution in game theory... But some problems!
  - *Non-uniqueness*: Often multiple NE exist, how should agents choose same one?
  - *Incompleteness*: NE does not specify behaviours for off-equilibrium paths
  - *Sub-optimality*: NE not generally same as utility maximisation
  - *Rationality*: NE assumes all agents are rational (= perfect utility maximisers)

Learning goals for a pay-off matrix

- **Pareto Optimum**: Can't improve one agent without making other agent worse off
- Policy profile $\pi = (\pi_1, \ldots, \pi_n)$ is Pareto-optimal if there is no other profile π' such that

$$\forall i : U_i\left(\pi'\right) \geq U_i(\pi) \text{ and } \exists_i : U_i\left(\pi'\right) > U_i(\pi)$$



1, 1	4, 1
1, 4	3, 3

Pareto-front is set of all Pareto-optimal utilities (red line)

Learning goals for a pay-off matrix

- Pareto-optimality says nothing about social welfare and fairness
- **Welfare** and **fairness** of profile $\pi = (\pi_1, \ldots, \pi_n)$ often defined as

$$\text{Welfare } (\pi) = \sum_i U_i(\pi)$$

$$\text{Fairness } (\pi) = \prod_i U_i(\pi)$$

π welfare/fairness-optimal if maximum Welfare (π)/ Fairness (π)

- Any welfare/fairness-optimal π is also Pareto-optimal.

Learning goals for a pay-off matrix

- **No-Regret** policies.
- Given history $H^t = \left(a^0, a^1, \ldots, a^{t-1}\right)$, agent i's regret for not having taken action a_i is

$$R_i\left(a_i \mid H^t\right) = \sum_{\tau=0}^{t-1} u_i\left(a_i, a_{-i}^{\tau}\right) - u_i\left(a_i^{\tau}, a_{-i}^{\tau}\right)$$

Policy π_i achieves no-regret if

$$\forall a_i : \lim_{t \to \infty} \frac{1}{t} R_i\left(a_i \mid H^t\right) \leq 0$$

(Other variants exist)

Learning goals for a pay-off matrix

- Like Nash equilibrium, no-regret widely used in multiagent learning
- But, like NE, definition of regret has conceptual issues - Regret definition assumes other agents don't change actions

$$R_i\left(a_i \mid H^t\right)=\sum_{\tau=0}^{t-1} u_i\left(a_i, a_{-i}^\tau\right)-u_i\left(a_i^\tau, a_{-i}^\tau\right)$$

$\Rightarrow$ But: entire history may change if different actions taken!

- Thus, minimising regret not generally same as maximising utility

Learning goals for a pay-off matrix

Many algorithms designed to achieve some version of targeted optimality and safety:

- If other agent's policy π_j non learning fixed policy, agent i's learning should converge to best-response

$$U_i\left(\pi_i, \pi_j\right) \approx \max_{\pi_i'} U_i\left(\pi_i', \pi_j\right)$$

- If not in class, learning should at least achieve **safety** (maximin) utility

$$U_i\left(\pi_i, \pi_j\right) \approx \max_{\pi_i'} \min_{\pi_j'} U_i\left(\pi_i', \pi_j'\right)$$

Subsection 2

Repeated game

Normal-form game and repeated games

- In normal form, the information is available to the agents, so it is more a *decision* problem than a *learning* problem
- Normal-form game consists in a single interaction. No experience!
- Experience comes from repeated interactions

Repeated Games

Repeated game:

- Repeat same normal-form game: at each time t, each agent chooses action a_i^t and gets utility $u_i\left(a_1^t, \ldots, a_n^t\right)$
- Policy $\pi_i : \mathbb{H} \times A_i \rightarrow [0,1]$ assigns action probabilities based on history of interaction (experience)

$$\mathbb{H} = \cup_{t \in \mathbb{N}^0} \mathbb{H}^t, \quad \mathbb{H}^t = \left\{ H^t = \left(a^0, a^1, \ldots, a^{t-1}\right) \mid a^\tau \in A \right\}$$

Repeated Games

What is *expected* utility to agent i for policy profile $(\pi_1, \ldots, \pi_n)$?

- Repeating game $t \in \mathbb{N}$ times:

$$U_i(\pi_1, \ldots, \pi_n) = \sum_{H^t \in \mathbb{H}^t} P\left(H^t \mid \pi_1, \ldots, \pi_n\right) \sum_{\tau=0}^{t-1} r_i(a^\tau)$$

$$P\left(H^t \mid \pi_1, \ldots, \pi_n\right) = \prod_{\tau=0}^{t-1} \prod_{j \in N} \pi_j\left(H^\tau, a_j^\tau\right)$$

Repeated Games

What is expected utility to i for policy profile $(\pi_1, \ldots, \pi_n)$?

- Repeating game ∞ times:

$$U_i(\pi_1, \ldots, \pi_n) = \lim_{t \to \infty} \sum_{H^t} P\left(H^t \mid \pi_1, \ldots, \pi_n\right) \sum_{\tau} \gamma^{\tau} u_i(a^{\tau})$$

Discount factor $0 \leq \gamma < 1$ makes expectation finite Interpretation: low γ is "myopic", high γ is "farsighted" (Or: probability that game will end at each time is $1 - \gamma$)

Repeated Game: Rock-Paper-Scissors

- Example: Repeated Rock-Paper-Scissors

		Player 2		
		R	P	S
Player 1	R	$(0,0)$	$(-1,1)$	$(1,-1)$
	P	$(1,-1)$	$(0,0)$	$(-1,1)$
	S	$(-1,1)$	$(1,-1)$	$(0,0)$

- Compute empirical frequency of opponent actions over past 5 moves

$$P\left(a_j\right) = \frac{1}{5} \sum_{\tau=t-5}^{t-1} \left[a_j^{\tau} = a_j\right]_1$$

and take best-response action $\max_{a_i} \sum_{a_j} P\left(a_j\right) u_i\left(a_i, a_j\right)$

Algorithms

- Minimax-Q (Littman, 1994)
- Nash-Q (Hu and Wellman, 2003)
- JAL (Claus and Boutilier, 1998)
- CJAL (Banerjee and Sen, 2007)
- Regret Matching (Hart and Mas-Colell, 2001, 2000)

Minimax-Q (Littman, 1994)

- Designed for competitive games (or irrational with conservative costs): Assumes other agent will take worst action for me
- Q-values are over joint actions: $Q(s, a, o)$ where:
  - s is state
  - a is your action
  - o action of the opponent
- Instead of playing action with highest $Q(s, a, o)$, play *MaxMin*

$$
\begin{aligned}
Q(s, a, o) =& (1-\alpha) Q(s, a, o) + \alpha \left(r + \gamma V\left(s'\right)\right) \\
V(s) =& \max_{\pi_s} \min_{o} \sum_{a} Q(s, a, o) \pi_s(a)
\end{aligned}
$$

[Exploration and other details]

- In RL algorithms, exploration is necessary in order to improve and get out of local minima
- To study convergence of algorithms, usually exploration is reduced with experience
- Popular method in the list of algorithms is Boltzmann exploration with temperature τ decreasing with time

$$\pi(s, a) = \frac{e^{Q(s,a)/\tau}}{\sum_{\tilde{a}} e^{Q(s,\tilde{a})/\tau}}$$

- When ties, actions are selected randomly

Nash-Q (Littman, 1994)

- Designed for general cases.
- Instead of playing action with highest $Q(s,a,o)$, sample action from mixed exploration policy with policy derived from Nash equilibrium extracted from $Q(s,a)$
- From data collected from actions executed, update $Q(s,a)$

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

- Where $\texttt{Nash}([Q(s,a)])$ consists in solving the Nash equilibrium for Pay-off matrix $Q(s,a)$.
- That means that at each iteration we have to solve a Nash equilibrium problem

Algorithms: JAL and CJAL

- Joint Action Learning (JAL) (Claus and Boutilier, 1998) and Conditional Joint Action Learning (CJAL) (Banerjee and Sen, 2007) learn Q-values for joint actions $a \in A$:

$$Q^{t+1}\left(a^t\right) = (1-\alpha)Q^t\left(a^t\right) + \alpha r_i^t$$

r_i^t is reward received after joint action a^t
$\alpha \in [0, 1]$ is learning rate

Algorithms: JAL and CJAL

- Joint Action Learning (JAL) (Claus and Boutilier, 1998) and Conditional Joint Action Learning (CJAL) (Banerjee and Sen, 2007) learn Q-values for joint actions $a \in A$:

$$Q^{t+1}\left(a^{t}\right)=(1-\alpha) Q^{t}\left(a^{t}\right)+\alpha r_{i}^{t}$$

  r_i^t is reward received after joint action a^t
  $\alpha \in [0,1]$ is learning rate

- Use **opponent modeling** to compute *expected utilities* of actions:

$$\text{JAL: } E\left(a_{i}\right)=\sum_{a_{j}} P\left(a_{j}\right) Q^{t+1}\left(a_{i}, a_{j}\right)$$

$$\text{CJAL: } E\left(a_{i}\right)=\sum_{a_{j}} P\left(a_{j} \mid a_{i}\right) Q^{t+1}\left(a_{i}, a_{j}\right)$$

Opponent modelling

Opponent models estimated from history H^t :

- JAL:

$$P\left(a_j\right) = \frac{1}{t+1} \sum_{\tau=0}^{t} \left[a_j^\tau = a_j\right]_1$$

- CJAL:

$$P\left(a_j \mid a_i\right) = \frac{\sum_{\tau=0}^{t} \left[a_j^\tau = a_j, a_i^\tau = a_i\right]_1}{\sum_{\tau=0}^{t} \left[a_j^\tau = a_j\right]_1}$$

- Given expected utilities $E(a_i)$, use some action exploration scheme: (e.g. ϵ-greedy)

Opponent modelling

Opponent models estimated from history H^t :

- JAL:

$$P\left(a_j\right)=\frac{1}{t+1}\sum_{\tau=0}^{t}\left[a_j^\tau=a_j\right]_1$$

- CJAL:

$$P\left(a_j \mid a_i\right)=\frac{\sum_{\tau=0}^{t}\left[a_j^\tau=a_j, a_i^\tau=a_i\right]_1}{\sum_{\tau=0}^{t}\left[a_j^\tau=a_j\right]_1}$$

- Given expected utilities $E(a_i)$, use some action exploration scheme: (e.g. ϵ-greedy)
- Many other forms of opponent modelling exist
- JAL and CJAL can converge to Nash equilibrium in self-play

Algorithms: Regret Matching (RegMat) (Hart and Mas-Colell, 2000)

- Computes conditional regret for not choosing a_i' whenever a_i was chosen:

$$R\left(a_i, a_i'\right) = \frac{1}{t+1} \sum_{\tau : a_i^\tau = a_i} u_i\left(a_i', a_j^\tau\right) - u_i\left(a^\tau\right)$$

- Used to modify policy:

$$\hat{\pi}_i^{t+1}(a_i) = \begin{cases} \frac{1}{\mu} \max\left[R\left(a_i^\tau, a_i\right), 0\right] & a_i \neq a_i^t \\ 1 - \sum_{a_i' \neq a_i^\tau} \hat{\pi}_i^{t+1}\left(a_i'\right) & a_i = a_i^t \end{cases}$$

  where $\mu > 0$ is "inertia" parameter

- Converges to correlated equilibrium in self-play.

Some cooperative problems

- Independent learners Convergence to optimal joint action for simple cases:

		Player 2	
		C	*D*
Player 1	*C*	5	3
	D	2	0

- Climbing game: Independent learners stuck in (c,c), JAL gets to (b,b)

		Player 2		
		a	*b*	*c*
	a	11	−30	0
Player 1	*b*	−30	7	6
	c	0	0	5

Some cooperative problems

- Optimistic version of Q-learning (Lauer & Riedmiller, 00), that never reduces Q-values due to penalties converges quickly to optimal (a,a).

Some cooperative problems

- Optimistic version of Q-learning (Lauer & Riedmiller, 00), that never reduces Q-values due to penalties converges quickly to optimal (a,a).
- However, it does not solve the Stochastic Climbing game:

		Player 2		
		a	*b*	*c*
	a	$12/10$	$0/-60$	$5/-5$
Player 1	*b*	$0/-60$	$14/0$	$8/4$
	c	$5/-5$	$5/-5$	$7/3$

Some cooperative problems

- Penalty game

$$
\begin{array}{cc|c|c|c|}
 & & \multicolumn{3}{c}{\text{Player 2}} \\
 & & a & b & c \\
\hline
 & a & 10 & 0 & k \\
\hline
\text{Player 1} & b & 0 & 2 & 0 \\
\hline
 & c & k & 0 & 10 \\
\hline
\end{array}
$$

Subsection 3

Stochastic game

Stochastic Game (Markov Decision Games)

- In stochastic games we introduce the state in the game.
- So, in practice, now we have one pay-off table for state and we add transition dynamics (from state using joint action to state)
- In addition we distinguish in general from state and observation (Partial Observability)

Let's focus on cooperation problems

Most general definition of a problem is partially observable *stochastic game* (POSG) that consists in a tuple:

$$< I, S, \{A_i\}, P, \{R_i\}, \{\Omega_i\}, O >$$

- I, a finite set of agents
- S, a finite set of states with designated initial state distribution b_0
- A_i, each agent's finite set of actions
- P, the state transition model: $P(s' \mid s, \vec{a})$
- $\{R_i\}$ the reward model for each agent: $R_i(s, \vec{a})$
- Ω_i, each agent's finite set of observations
- O, the observation model: $P(\vec{O} \mid s, \vec{a})$

Stochastic Game (Markov Decision Games)

- Algorithms we have seen work well when few agents and also when value functions are used to store the Q-values
- How do we extend these methods to more complex scenarios like these?
- We will need function approximation, we will need to solve the Partial Observability
- May be the problem is too complex

Latest MARL research

Let's focus on cooperation problems

- We have seen three kinds of problems:
  - Cooperative
  - Competitive
  - Mixed
- We will focus now on cooperation!

Let’s focus on cooperation problems

- We have seen three kinds of problems:
  - Cooperative
  - Competitive
  - Mixed
- We will focus now on cooperation!
- Cooperation when all agents have to cooperate for the same goal
- Reward is shared to all agents
- We have seen them before (f.i. Coordination problem)

Let's focus on cooperation problems

- We have seen three kinds of problems:
  - Cooperative
  - Competitive
  - Mixed
- We will focus now on cooperation!
- Cooperation when all agents have to cooperate for the same goal
- Reward is shared to all agents
- We have seen them before (f.i. Coordination problem)
- Why we focus on these problems? Because they are important and because they are simpler!

Decentralized POMDP Stochastic games

A Dec-POMDP can be defined with the tuple:

$$\mathrm{M} = < I, S, \{A_i\}, P, R, \{\Omega_i\}, O >$$

- I, a finite set of agents
- S, a finite set of states with designated initial state distribution b_0
- A_i, each agent's finite set of actions
- P, the state transition model: $P(s' \mid s, \vec{a})$
- R, **the reward model:** $R(s, \vec{a})$
- Ω_i, each agent's finite set of observations
- O, the observation model: $P(\vec{O} \mid s, \vec{a})$

Independent Q-learners: IQL (Tan, 93)

- Naive idea: Try to maximize reward of each agent independently.
- Maximizing individual rewards means maximize cooperation, isn't it?

Independent Q-learners: IQL (Tan, 93)

- Naive idea: Try to maximize reward of each agent independently.
- Maximizing individual rewards means maximize cooperation, isn't it?
- Wrong. Only true when you know other agent's actions (coordination problem)

$$
\begin{array}{ccc|c|c|}
 & & \multicolumn{2}{c}{\text{Player 2}} \\
 & & \multicolumn{1}{c}{A} & \multicolumn{1}{c}{B} \\
\cline{3-4}
\text{Player 1} & A & 5 & -5 \\
\cline{3-4}
 & B & -5 & 5 \\
\cline{3-4}
\end{array}
$$

Decentralized learning learning

- We can model opponents but they also learn (change). So, experiences collected in ER become obsolete!
- In (Foerster et al. 18) authors use IS to maintain the ER.
- In (Bansal et al. 18) authors get rid of ER and use on-policy algorithms (PPO).
- Another way to explore is communication (at different levels).
- In general results are not so good as using other approaches.

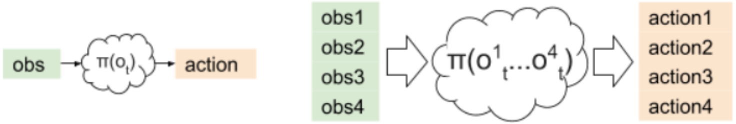
Centralized Learning

- We know in MARL a given agent faces a **non-stationary problem** (no longer Markovian) because other agents do changes in state (and we don't know how).
- In addition we have the *moving target* problem (other agents also learn!)
- So basic RL algorithms applied in a naive way will not have guarantees to work

Centralized Learning

- We know in MARL a given agent faces a **non-stationary problem** (no longer Markovian) because other agents do changes in state (and we don't know how).
- In addition we have the *moving target* problem (other agents also learn!)
- So basic RL algorithms applied in a naive way will not have guarantees to work
- Naive idea: Learn a centralized policy that control all agents
- Policy gets as inputs the observations of all the agents
- Non-stationarity and moving target problems solved!

Centralized Learning



- Non-stationarity and moving target problems solved!

Centralized Learning - Decentralized actuation

- ... But not realistic because we need a lot of communication channels (each agent to the "big brain")
- In addition, we have an exponential grown in actions to control: $|A|^{nagents}$... and in variance of the gradient!
- Centralized goes against the idea of multi-agent (only one agent!)
- ... But Decentralized idea neither worked. Any solution?

Centralized Learning - Decentralized actuation

- ... But not realistic because we need a lot of communication channels (each agent to the "big brain")
- In addition, we have an exponential grown in actions to control: $|A|^{nagents}$... and in variance of the gradient!
- Centralized goes against the idea of multi-agent (only one agent!)
- ... But Decentralized idea neither worked. Any solution?

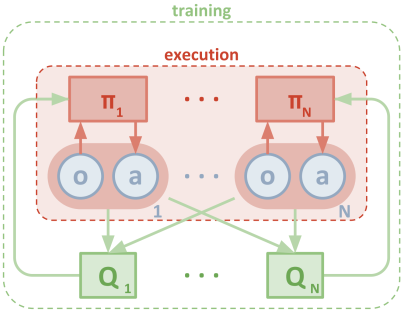
- Yes! A compromise between Centralized and Decentralized
- Learning will be done in a centralized way so we have the information needed to learn
- but considering than in execution time, each agent will have to act independently of the others

Subsection 1

Actor Critic approaches

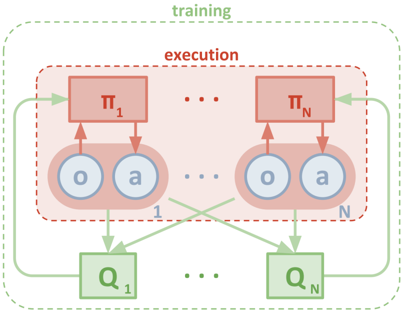
MADDPG (Lowe et al 2017)

- Based on Actor-Critic architecture: Each agent has an actor and a critic
- Extension of DDPG for MultiAgent framework

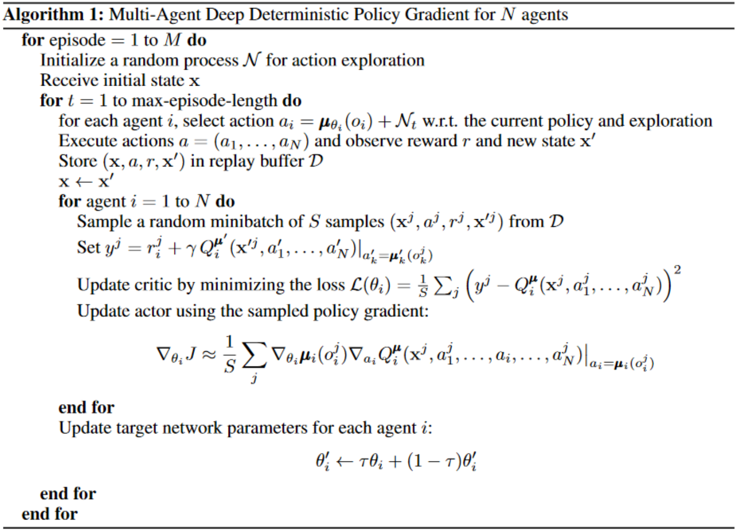


MADDPG (Lowe et al 2017)

- During training, each critic has information about the actions taken by all agents and their perceptions
- Actions are generated by own policy of each actor according to its perceptions
- Each agent's critic (with full information) is used to train its associated actor (with respect its own observation)



MADDPG (Lowe et al 2017)



Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

```
for episode = 1 to $M$ do
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length do
    for each agent $i$, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward $r$ and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to $N$ do
      Sample a random minibatch of $S$ samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

        $$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\boldsymbol{\mu}_i(o_i^j)\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent $i$:

        $$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for
```

MADDPG (Lowe et al 2017)

- Training is done in simulator or lab with all info available for critics
- Deployment of the agents is done in the execution step where information of perceptions of other agents is no longer necessary
- Once the agents have been deployed, no more learning occurs
- Critics are no longer necessary and agents work in a **decentralized** way.

MADDPG (Lowe et al 2017)

- Training is done in simulator or lab with all info available for critics
- Deployment of the agents is done in the execution step where information of perceptions of other agents is no longer necessary
- Once the agents have been deployed, no more learning occurs
- Critics are no longer necessary and agents work in a **decentralized** way.

- You may wonder why you need several agents instead of only one?

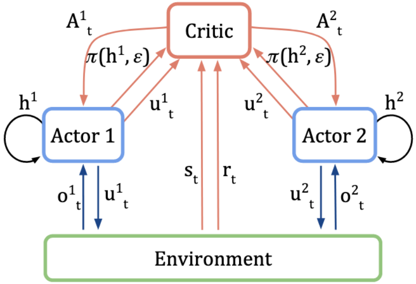
MADDPG (Lowe et al 2017)

- Training is done in simulator or lab with all info available for critics
- Deployment of the agents is done in the execution step where information of perceptions of other agents is no longer necessary
- Once the agents have been deployed, no more learning occurs
- Critics are no longer necessary and agents work in a **decentralized** way.

- You may wonder why you need several agents instead of only one?
- If agents are homogeneous you can work with only one critic, but in some cases agents of the team are not all equal
- In this cases having different critics help because the critic is specialized on the specific capabilities of the agent

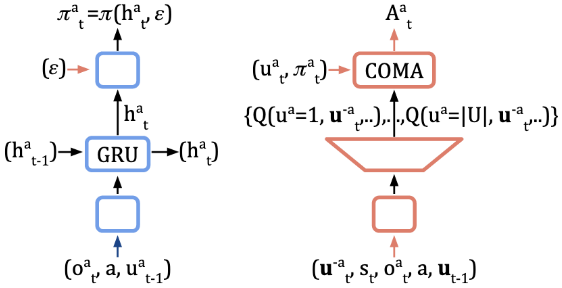
COMA (Foerster et al., 17)

- The *Counterfactual Multi-Agent* (COMA) architecture is based on Actor Critic.
- Only one Critic and *n* Actors.
- Centralized learning. Critic is removed after training.



COMA (Foerster et al., 17)

- Actor is standard probabilistic policy trained with recurrent NN



$\pi^a_t = \pi(h^a_t, \varepsilon)$
(ε)
h^a_t
(h^a_{t-1})
GRU
(h^a_t)
(o^a_t, a, u^a_{t-1})
A^a_t
(u^a_t, π^a_t)
COMA
$\{Q(u^a=1, \mathbf{u}^{-a}_t, ..), .., Q(u^a=|U|, \mathbf{u}^{-a}_t, ..)\}$
$(\mathbf{u}^{-a}_t, s_t, o^a_t, a, \mathbf{u}_{t-1})$

COMA (Foerster et al., 17)

- Critic computes Q-values on on-policy and using TD(λ)
- Given an agent, Critic computes Q-value for all possible joint actions where the actions of other actions are fixed.
- Actors are trained with Advantage Actor Critic (so it is on-policy!) BUT with a *counterfactual baseline*:

$$A^a(s, \mathrm{u}) = Q(s, \mathrm{u}) - \sum_{u'^a} \pi^a\left(u'^a \mid \tau^a\right) Q\left(s, \left(\mathrm{u}^{-a}, u'^a\right)\right)$$

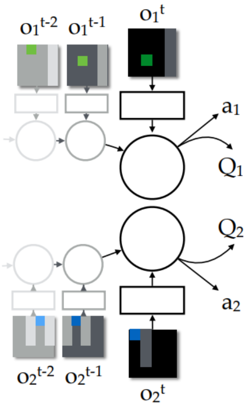
- In short, compares taken action with expected value under the current policy
- Intuitively, by using this baseline, the agent knows how much reward this action contributes relative to all other actions it could've taken.
- In doing so, it can better distinguish which actions will better contribute to the overall reward across all agents.

Subsection 2

Value based approaches

VDN (Sunehag et al. 17)

- Value Decomposition Network (VDN) Clever idea for Value based RL methods
- Problem with IQL: Each agent own reward, no communication with other agents
- (POMDP problem, so they use a recurrent network for Q-values)
- We have only long term reward for the joint action



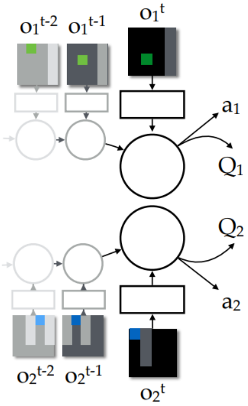
VDN (Sunehag et al. 17)

- Idea is that each agent contributes to total long-term reward, so we can decompose the credit to give to each agent as a sum:

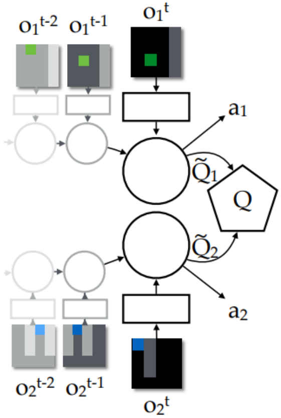
$$Q_{tot}\left(\left(h^1, h^2, \dots, h^d\right), \left(a^1, a^2, \dots, a^d\right)\right) \approx \sum_{i=1}^{d} \tilde{Q}_i\left(h^i, a^i\right)$$

- This decomposition is **learnt**! We only have Q_{tot}
- Tricky point is that $\tilde{Q}_i$ **are not true value functions** because they do not predict reward. They are used as tools and learnt decomposing Q_{tot}

VDN (Sunehag et al. 17)



o_1^{t-2} o_1^{t-1} o_1^t
a_1
Q_1
Q_2
a_2
o_2^{t-2} o_2^{t-1} o_2^t



o_1^{t-2} o_1^{t-1} o_1^t
a_1
$\tilde{Q}_1$
Q
$\tilde{Q}_2$
a_2
o_2^{t-2} o_2^{t-1} o_2^t

VDN (Sunehag et al. 17)

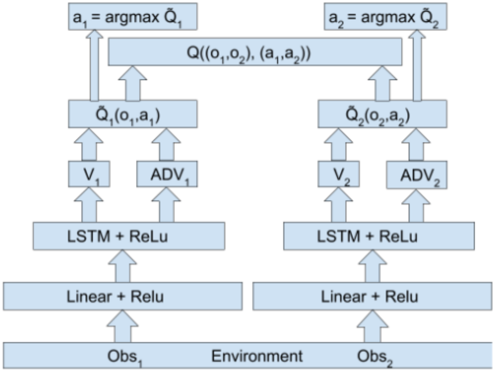
- VDN decomposition:

$$Q_{tot}\left(\left(h^1, h^2, \ldots, h^d\right), \left(a^1, a^2, \ldots, a^d\right)\right) \approx \sum_{i=1}^{d} \tilde{Q}_i\left(h^i, a^i\right)$$

- During training we use Q_{tot} values to backpropagate and learn credit to decisions taken by actions of agents that were selected using greedy criteria in their $\tilde{Q}_i$
- Coherence between $\tilde{Q}_i$ and Q_{tot} is maintained because greedifying Q_{tot} to obtain the joint actions and greedifying each $\tilde{Q}_i$ we obtain the same result
- After the system is trained, we go to a decentralized scheme when deploying agents (learning is stopped).

VDN (Sunehag et al. 17)

- VDN architecture:



a_1 = argmax $\tilde{Q}_1$
a_2 = argmax $\tilde{Q}_2$
Q((o_1,o_2), (a_1,a_2))
$\tilde{Q}_1$(o_1,a_1)
$\tilde{Q}_2$(o_2,a_2)
V_1
ADV_1
V_2
ADV_2
LSTM + ReLu
LSTM + ReLu
Linear + Relu
Linear + Relu
Obs_1
Environment
Obs_2

Figure 15: Value-Decomposition Individual Architecture

Q-Mix (Rashid et al. 18)

- Another cool idea and probably state-of-the-art algorithm
- Extend the VDN idea
- Key point in VDM is that:

$$\underset{\mathrm{u}}{\operatorname{argmax}} Q_{tot}(\boldsymbol{\tau}, \mathrm{u}) = \left(\begin{array}{c} \operatorname{argmax}_{u^1} Q_1\left(\tau^1, u^1\right) \\ \vdots \\ \operatorname{argmax}_{u^n} Q_n\left(\tau^n, u^n\right) \end{array} \right)$$

- which is trivial in the case of the VDM sum
- May be there could be more interesting functions that only the sum?
- (Remember Q_i values are not true value functions, they are only tools)

Q-Mix (Rashid et al. 18)

- Answer is yes.
- The condition to satisfy the argmax condition coherence between Q_{tot} and Q_i is monotonicity:

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \forall a \in A$$

- so they enforce this constraint by forcing the composition (mixing) function to learn a possible non-linearly but monotonic function
- This is done by ensuring positive weights in the mixing network that are learnt by a hyper-network (details in paper)

Q-Mix (Rashid et al. 18)

- QMIX architecture:

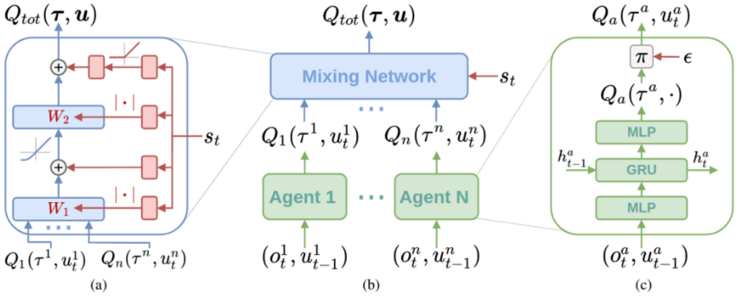


Figure 2. (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. (c) Agent network structure. Best viewed in colour.

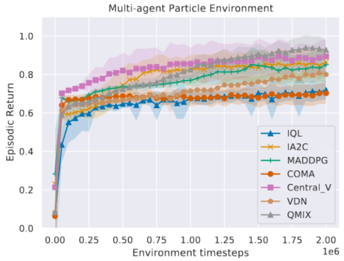
Q-Mix (Rashid et al. 18)

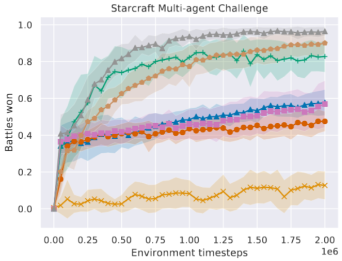
- Results of QMIX in StarCraft minigames here

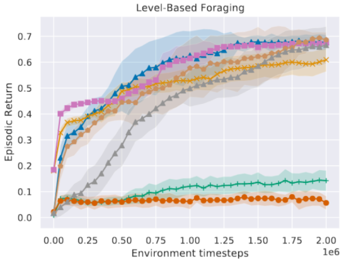
Comparison of methods

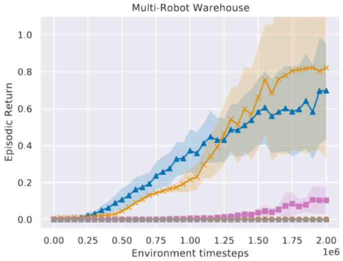
- Problems with QMix
  - Better approximation than VDN to values but QMIX neither can solve tasks that require significant coordination within a given time-step (coordination is non-monotonic!)
  - Poor exploration
  - QMIX ranks low in training stability compared to on-policy algorithms
- Based on QMIX and VDN, other approaches try to improve their results (WQMIX, MAVEN, etc)
- However, a recent unpublished papers (Papoudakis et alt. 20) and (Hu et alt 21) shows that with QMIX, in cooperative tasks, stat-of-the-art results are obtained in most cases.

Comparison of methods









Mixed cases

Mixed cases

- We have studied basically cooperative problems
- More interesting problems have mixed cooperation and competition
- Special case are the *Social Dilemmas*: Situations where any individual may profit from selfishness unless too many individuals choose the selfish option, in which case the whole group loses.
- Appear everywhere: Pollution, the tragedy of the commons, public goods, resource depletion, etc.

Some links for labs

Some links for labs

- Some test environments for MARL:
  - Petting Zoo (highly recommended)
  - StarCraft Multi-Agent Challenge
  - Multi-Agent Particle Environment
  - Arena
  - Some Sequential Social Dilemma Games, more here and here
  - Multi-Agent-Learning-Environments
  - Pommerman
  - Flatland challenge
  - Laser Tag
  - MicroRTS and Gym
  - Drones? Yes drones!
- Implementations: of value methods for RIIT paper and other basic methods and from whirl Lab PyMARL

Conclusions

Conclusions

- One of the hottest topics of RL research at the moment.
- Very difficult problem (dimensionality, patial observability, exploration, decomposition of reward, moving target problem) ... and still open!
- Some results in Cooperative cases. They can be extended at some extent to Mixed cases like Sequential Social Dilemmas.

Conclusions

- One of the hottest topics of RL research at the moment.
- Very difficult problem (dimensionality, patial observability, exploration, decomposition of reward, moving target problem) ... and still open!
- Some results in Cooperative cases. They can be extended at some extent to Mixed cases like Sequential Social Dilemmas.

- Other kind of algorithms for competition
- Two-players games are an special case. See you in next class!

Conclusions

- One of the hottest topics of RL research at the moment.
- Very difficult problem (dimensionality, patial observability, exploration, decomposition of reward, moving target problem) ... and still open!
- Some results in Cooperative cases. They can be extended at some extent to Mixed cases like Sequential Social Dilemmas.

- Other kind of algorithms for competition
- Two-players games are an special case. See you in next class!
- Very Funny example: hide and seek

Other lines of research in MARL

- In (Hernandez-Leal , et al. 19) authors describe 4 kinds of recent research in MARL
  a. Analysis of emergent behaviors: evaluate single-agent DRL algorithms in multiagent scenarios(e.g., Atari games, social dilemmas, 3D competitive games)
  b. Learning communication: agents learn communication protocols to solve cooperative tasks.
  c. Learning cooperation: agents learn to cooperate using only actions and (local) observations.
  d. Agents modeling agents: agents reason about others to fulfill a task (e.g., best response learners)
- We have focused in c) and d) on this slides. A lot more work done in the area!

oxwhirl / smac Public

Notifications Fork 191 Star 725

Code

Issues 8

Pull requests 2

Actions

Projects

Security

Insights

master

samvelyan Merge pull request #97 from jjshoots/master

…

Aug 8, 2022

94

docs
Update docs, add description for map: 1c3s5z
Jul 3, 2020

smac
fix options
Jul 9, 2022

.gitignore
Minor updates regarding pygame rendering
Jul 19, 2021

.pre-commit-confi...
Added black and flake8 in precommit hook
Jul 19, 2021

LICENSE
Initial commit
Jan 8, 2019

README.md
Update README.md
Jul 29, 2021

pyproject.toml
Blacked everything
Jul 19, 2021

setup.py

Blacked everything
Jul 19, 2021

README.md

SMAC WhiRL

Blizzard StarCraft II StarCraft II

Machine Learning API DeepMind PySC2

PySC2

paper blogpost

PyMARL PyTorch

QMIX COMA

mikayel@samvelyan.com tabish.rashid@cs.ox.ac.uk

here

```
pip install -e smac/
```

NOTE: If you want to extend SMAC, please install the package as follows:

```
git clone https://github.com/oxwhirl/smac.git
cd smac
pip install -e ".[dev]"
pre-commit install
```

You may also need to upgrade pip: `pip install --upgrade pip` for the install to work.

Installing StarCraft II

SMAC is based on the full game of StarCraft II (versions >= 3.16.1). To install the game, follow the commands bellow.

Linux

Please use the Blizzard's repository to download the Linux version of StarCraft II. By default, the game is expected to be in `~/StarCraftII/` directory. This can be changed by setting the environment variable `SC2PATH`.

MacOS/Windows

Please install StarCraft II from Battle.net. The free Starter Edition also works. PySC2 will find the latest binary should you use the default install location. Otherwise, similar to the Linux version, you would need to set the `SC2PATH` environment variable with the correct location of the game.

SMAC maps

SMAC is composed of many combat scenarios with pre-configured maps. Before SMAC can be used, these maps need to be downloaded into the `Maps` directory of StarCraft II.

Download the SMAC Maps and extract them to your `$SC2PATH/Maps` directory. If you installed SMAC via git, simply copy the `SMAC_Maps` directory from `smac/env/starcraft2/maps/` into `$SC2PATH/Maps` directory.

List the maps

To see the list of SMAC maps, together with the number of ally and enemy units and episode limit, run:

```
python -m smac.bin.map_list
```

Creating new maps

Users can extend SMAC by adding new maps/scenarios. To this end, one needs to:

- Design a new map/scenario using StarCraft II Editor:

   - Please take a close look at the existing maps to understand the basics that we use (e.g. Triggers, Units, etc),
   - We make use of special RL units which never automatically start attacking the enemy. Here is the step-by-step guide on how to create new RL units based on existing SC2 units,
- Add the map information in smac_maps.py,
- The newly designed RL units have new ids which need to be handled in starcraft2.py. Specifically, for heterogenious maps containing more than one unit types, one needs to manually set the unit ids in the `_init_ally_unit_types()` function.

Testing SMAC

Please run the following command to make sure that `smac` and its maps are properly installed.

```
python -m smac.examples.random_agents
```

Saving and Watching StarCraft II Replays

Saving a replay

If you've using our PyMARL framework for multi-agent RL, here's what needs to be done:

1. Saving models: We run experiments on *Linux* servers with `save_model = True` (also `save_model_interval` is relevant) setting so that we have training checkpoints (parameters of neural networks) saved (click here for more details).
2. Loading models: Learnt models can be loaded using the `checkpoint_path` parameter. If you run PyMARL on *MacOS* (or *Windows*) while also setting `save_replay=True`, this will save a .SC2Replay file for `test_nepisode` episodes on the test mode (no exploration) in the Replay directory of StarCraft II. (click here for more details).

If you want to save replays without using PyMARL, simply call the `save_replay()` function of SMAC's StarCraft2Env in your training/testing code. This will save a replay of all epsidoes since the launch of the StarCraft II client.

The easiest way to save and later watch a replay on Linux is to use Wine.

Watching a replay

You can watch the saved replay directly within the StarCraft II client on MacOS/Windows by *clicking on the corresponding Replay file*.

You can also watch saved replays by running:

```
python -m pysc2.bin.play --norender --replay <path-to-replay>
```

This works for any replay as long as the map can be found by the game.

For more information, please refer to PySC2 documentation.

Documentation

For the detailed description of the environment, read the SMAC documentation.

The initial results of our experiments using SMAC can be found in the accompanying paper.

Citing SMAC

If you use SMAC in your research, please cite the SMAC paper.

M. Samvelyan, T. Rashid, C. Schroeder de Witt, G. Farquhar, N. Nardelli, T.G.J. Rudner, C.-M. Hung, P.H.S. Torr, J. Foerster, S. Whiteson. The StarCraft Multi-Agent Challenge, CoRR abs/1902.04043, 2019.

In BibTeX format:

```
@article{samvelyan19smac,
  title = {{The} {StarCraft} {Multi}-{Agent} {Challenge}},
  author = {Mikayel Samvelyan and Tabish Rashid and Christian Schroeder de Witt and
Gregory Farquhar and Nantas Nardelli and Tim G. J. Rudner and Chia-Man Hung and
Philiph H. S. Torr and Jakob Foerster and Shimon Whiteson},
  journal = {CoRR},
  volume = {abs/1902.04043},
  year = {2019},
}
```

Code Examples

Below is a small code example which illustrates how SMAC can be used. Here, individual agents execute random policies after receiving the observations and global state from the environment.

If you want to try the state-of-the-art algorithms (such as QMIX and COMA) on SMAC, make use of PyMARL - our framework for MARL research.

```
from smac.env import StarCraft2Env
import numpy as np


def main():
    env = StarCraft2Env(map_name="8m")
    env_info = env.get_env_info()

    n_actions = env_info["n_actions"]
    n_agents = env_info["n_agents"]

    n_episodes = 10

    for e in range(n_episodes):
        env.reset()
        terminated = False
```

RLlib PPO

IMPALA here

Rodrigo de Lazcano PettingZoo API

here

About

SMAC: The StarCraft Multi-Agent Challenge

benchmark machine-learning reinforcement-learning multiagent-systems starcraft-ii

Readme

MIT license

725 stars

19 watching

191 forks

Releases 2

Version 1 Latest
Dec 4, 2019

+ 1 release

Packages

No packages published

Contributors 13

+ 2 contributors

Languages

Python 100.0%