Policy Gradient Algorithms

Hong Xingxing

December 7, 2018

Outline

Asynchronous one-step Q-learning

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ, θ^-, and counter $T = 0$.
Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state s
repeat
  Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
  Receive new state s' and reward r
  $$y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$$
  Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
  $s = s'$
  $T \leftarrow T + 1$ and $t \leftarrow t + 1$
  if $T \mod I_{target} == 0$ **then**
    Update the target network $\theta^- \leftarrow \theta$
  end if
  if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
    Perform asynchronous update of θ using $d\theta$.
    Clear gradients $d\theta \leftarrow 0$.
  end if
until $T > T_{max}$

A3C

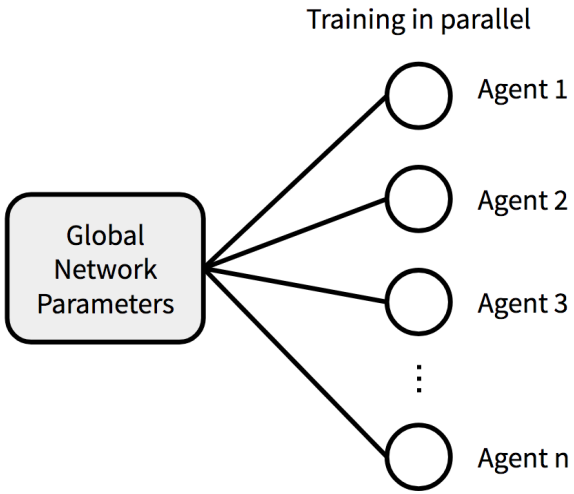
In A3C, the critics learn the value function while multiple actors are trained in parallel and get synced with global parameters from time to time. Hence, A3C is designed to work well for parallel training.

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

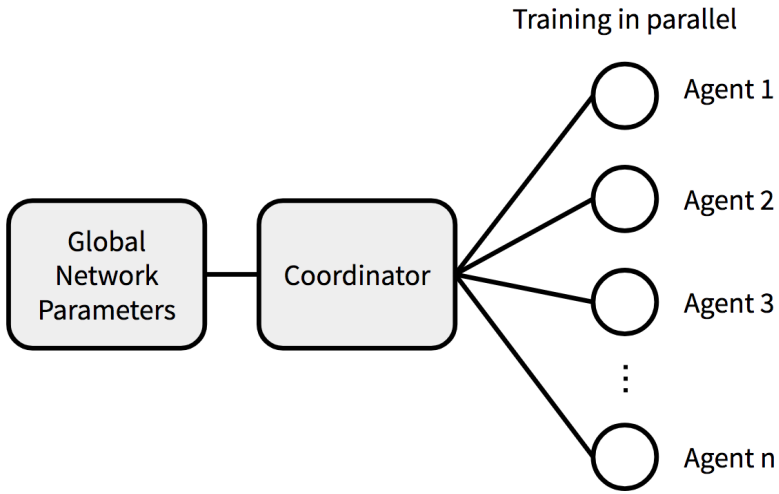
// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

A2C

In A2C waits for all the parallel actors to finish their work before updating the global parameters and then in the next iteration parallel actors starts from the same policy. The synchronized gradient update keeps the training more cohesive and potentially to make convergence faster.



A3C (Async)



A2C (Sync)

Multi-Agent Applications

Applications that involve interaction between multiple agents

- Multi-robot control
- Multiplayer games
- Hierarchical reinforcement learning can also be seen multi-agent system, with multiple levels of hierarchy being equivalent to multiple agents

MADDPG

Traditional RL algorithms such as Q-Learning and policy gradient are poorly suited to multi-agent environments.

- **Learning stability**: each agent's policy is changing as training processes. Environment becomes non-stationary from the perspective of any individual agent.
- **Experience replay**: prevents the straightforward use of past experience replay, which is crucial for stabilizing deep Q-learning
- **Differentiable model**: model-based policy optimization via back-propagation requires a (differential) model of the word dynamics

MADDPG

This paper propose a general-purpose multi-agent learning algorithms:

- Learned policies only use local information (their own observations) at executing time
- Does not assume a differentiable model of environment dynamics or any particular structures on the communication methods between agents
- Is applicable not only to cooperative interaction but to competitive or mixed interaction involving both physical and communication behavior

MADDPG

- **MADDPG frame**
  - A simple extension of actor-critic policy gradient methods where critic is augmented with extra information about the polices of other agents and actor only has access to local information
  - Centralized training with decentralized execution
- **Q-Learning**
  - Q function generally cannot contain different information at training and testing time

Problem Formulation

Markov Games: multi-agent extension of Markov decision processes (MDPs) called partially observable Markov games

- There are N agents in total with a set of state $\mathcal{S}$
- Each agent owns a set of possible action, $\mathcal{A}_1, \ldots, \mathcal{A}_N$
- Each agent owns a set of observation, $\mathcal{O}_1, \ldots, \mathcal{O}_N$
- Each agent i uses a stochastic policy to choose actions: $\pi_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \mapsto [0, 1]$
- The state transition function involves all states, action and observation spaces $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \mathcal{A}_N \mapsto \mathcal{S}$
- The initial states are determined by a distribution $\rho : \mathcal{S} \mapsto [0, 1]$
- Each agent i aims to maximize its own total expected return $R_i = \sum_{t=0}^{T} \gamma^t {r_i}^t$

DQN review

DQN learns the action-value function Q^* corresponding to the optimal policy by minimizing the loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}\left[(Q^*(s,a|\theta) - y)^2\right] \tag{1}$$

where

$$y = r + \gamma \max_{a'} \bar{Q}^*(s', a') \tag{2}$$

To help stabilize learning:

- $\bar{Q}$ is a target Q function whose parameters are periodically updated with the most recent θ
- Experience replay buffer $\mathcal{D}$ contains (s, a, r, s'). However, experience replay buffer cannot be used in such setting $P(s'|s, a, \pi_1, ..., \pi_N) \neq P(s'|s, a, \pi'_1, ..., \pi'_N)$ when any $\pi_i \neq \pi'_i$

Policy Gradient Algorithms review

Adjust the parameters θ of the policy in order to maximize the objective $J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[R]$ by taking steps in the direction of $\nabla_\theta J(\theta)$

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\right] \tag{3}$$

Some variants

- REINFORCE when estimate Q^π by a sample return $R^t = \sum_{i=t}^{T} \gamma^{i-t} r_i$
- ACTOR-CRITIC when estimate Q^π by an approximation of the true value-action function $Q^\pi(s,a)$ by temporal-difference learning and leads to a variety of actor-critic algorithms

Proposition

Proposition 1. *Consider N agents with binary actions: $P(a_i = 1) = \theta_i$, where $R(a_1, \ldots, a_N) = \mathbf{1}_{a_1=\cdots=a_N}$. We assume an uninformed scenario, in which agents are initialized to $\theta_i = 0.5\ \forall i$. Then, if we are estimating the gradient of the cost J with policy gradient, we have:*

$$P(\langle \hat{\nabla} J, \nabla J \rangle > 0) \propto (0.5)^N$$

where $\hat{\nabla} J$ is the policy gradient estimator from a single sample, and ∇J is the true gradient.

Proposition Proof

We can write $P(a_i) = \theta_i^{a_i}(1-\theta_i)^{1-a_i}$, and $\log P(a_i) = a_i \log \theta_i + (1-a_i)\log(1-\theta_i)$. The policy gradient estimator (from a single sample) is:

$$\begin{aligned}
\frac{\hat{\partial}}{\partial \theta_i} J &= R(a_1, ..., a_N) \frac{\partial}{\partial \theta_i} \log P(a_1, ..., a_N) \\
&= R(a_1, ..., a_N) \frac{\partial}{\partial \theta_i} \sum_i a_i \log \theta_i + (1-a_i)\log(1-\theta_i) \\
&= R(a_1, ..., a_N) \frac{\partial}{\partial \theta_i} (a_i \log \theta_i + (1-a_i)\log(1-\theta_i)) \\
&= R(a_1, ..., a_N) (\frac{a_i}{\theta_i} - \frac{1-a_i}{1-\theta_i})
\end{aligned}$$

Proposition Proof

For $\theta_i = 0.5$ we have:

$$\frac{\hat{\partial}}{\partial \theta_i} J = R(a_1, ..., a_N)(2a_i - 1) \tag{4}$$

And the expected reward can be calculated as:

$$\mathbb{E}(R) = \sum_{a_1, ... a_N} R(a_1, ..., a_N)(0.5)^N \tag{5}$$

Consider the case $R(a_1, ..., a_N) = 1_{a_1 = ... = a_N = 1}$. Then

$$\mathbb{E}(R) = (0.5)^N \tag{6}$$

and

$$\mathbb{E}(\frac{\hat{\partial}}{\partial \theta_i} J) = \frac{\partial}{\partial \theta_i} J = (0.5)^N \tag{7}$$

Proposition Proof

The variance of a single sample of the gradient is then:

$$\begin{aligned}\mathbb{V}(\frac{\hat{\partial}}{\partial\theta_i}J) &= \mathbb{E}(\frac{\hat{\partial}}{\partial\theta_i}J^2) - \mathbb{E}(\frac{\hat{\partial}}{\partial\theta_i}J)^2 \\ &= (0.5)^N - (0.5)^{2N}\end{aligned}$$

What is the probability of taking a step in the right direction? Look at $P(\langle\hat{\nabla}J, \nabla J\rangle > 0)$

$$\langle\hat{\nabla}J, \nabla J\rangle = \sum_i \frac{\hat{\partial}}{\partial\theta_i}J \times (0.5)^N = (0.5)^N \sum_i \frac{\hat{\partial}}{\partial\theta_i}J \tag{8}$$

so $P(\langle\hat{\nabla}J, \nabla J\rangle > 0) = (0.5)^N$. As the number of agents increase, the probability of taking a gradient step in the right direction decreases exponentially.

Deterministic Policy Gradient(DPG) Algorithms review

Deterministic Policies $\mu_\theta : \mathcal{S} \mapsto \mathcal{A}$. The gradient of the objective $J(\theta) = \mathbb{E}_{s\sim p^\mu}[R(s,a)]$ as

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_\theta \mu_\theta(a|s) \nabla Q^\mu(s,a)|_{a=\mu_\theta(s)}\right] \tag{9}$$

DDPG is a variant of DPG where the policy μ and critic Q^μ are approximated with deep neural networks.

MADDPG

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 $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)\big|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      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:

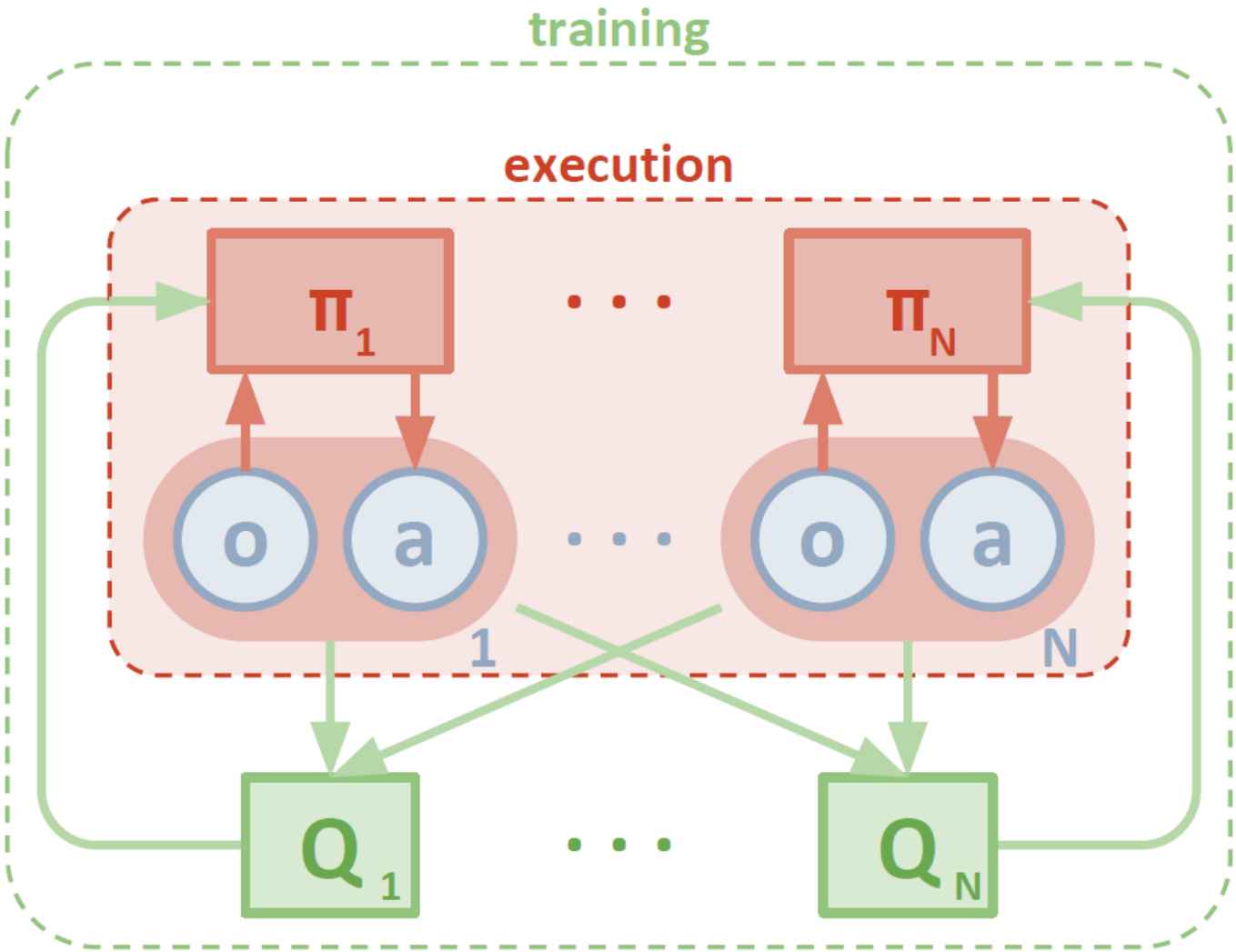
$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\boldsymbol{\mu}_i(o_i^j)\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

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

  end for
end for

Methods



Methods

- The learned policies can only use local information at execution time
- Do not assume a differentiable model of the environment dynamics
- Do not assume any particular structure on the communication method

Multi-Agent Stochastic Policy Gradient

Consider a game with N agents with policies parameterized by $\theta = \{\theta_1, ..., \theta_N\}$, and let $\pi = \{\pi_1, ..., \pi_N\}$ be the set of all agent policies.

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s \sim p^{\mu}, a_i \sim \pi_i} \left[\nabla_{\theta_i} \log \pi_i(a_i|o_i) Q_i^{\pi}(x, a_1, ..., a_N) \right] \tag{10}$$

- $Q_i^{\pi}(x, a_1, ..., a_N)$ is a centralized action-value function that takes as input the actions of all agents, $a_1, ..., a_N$, in addition to some state information x, and outputs the Q-value for agent i
- x could be $x = (o_1, ..., o_N)$

Multi-Agent Deterministic Policy Gradient

Consider N continuous policies μ_{θ_i} w.r.t parameters θ_i (abbreviated as μ_i), the gradient can be written as:

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{x,a\sim\mathcal{D}}\left[\nabla_{\theta_i}\mu_i(a_i|o_i)\nabla_{a_i} Q_i^{\mu}(x, a_1, ..., a_N)|_{a_i=\mu_i(o_i)}\right] \tag{11}$$

- Experience replay buffer $\mathcal{D}$ contains the tuples $(x, x', a_1, ..., a_N, r_1, ..., r_N)$
- The centralized value-action function Q_i^{μ} is updated as:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{x,a,r,x'}\left[(Q_i^{\mu}(x, a_1, ..., a_N) - y)^2\right] \tag{12}$$

$$y = r_i + \gamma Q_i^{\mu'}(x', a'_1, ..., a'_N)|_{a'_j=\mu'_j(o_j)} \tag{13}$$

- $\mu' = \{\mu_{\theta'_1}, ..., \mu_{\theta'_N}\}$ is the set of target policies with delayed parameters θ'_i

Multi-Agent Deterministic Policy Gradient

A primary motivation behind MADDPG is that, if we know the actions taken by all agents, the environment is stationary even as the policies change, since $P(s'|s, a_1, ..., a_N, \pi_1, ..., \pi_N) = P(s'|s, a_1, ..., a_N) = P(s'|s, a_1, ..., a_N, \pi'_1, ..., \pi'_N)$ for any $\pi_i \neq \pi'_i$

Inferring Policies of Other Agents

Each agent i can additionally maintain an approximation $\hat{\mu}_{\phi_i^j}$ (where ϕ are the parameters of the approximation) to the true policy of agent j, μ_j. This approximate policy is learned by maximizing the log probability of agent $j's$ actions, with an entropy regularizer:

$$\mathcal{L}(\phi_i^j) = -\mathbb{E}_{o_j, a_j}\Big[\log \hat{\mu}_i^j(a_j|o_j) + \lambda H(\hat{\mu}_i^j)\Big] \tag{14}$$

where H is the entropy of the policy distribution

$$\hat{y} = r_i + \gamma Q_i^{\mu'}(x', \hat{\mu}_i'^1(o_1), ..., \hat{\mu}_i'^N(o_N)) \tag{15}$$

Agents with Policy Ensembles

To mitigate the high variance triggered by the interaction between competing or collaborating agents in the environment, MADDPG proposed one more element-policy ensembles:

- Train K policies for one agent
- Pick a random policy for episode rollouts
- Take an ensemble of these K policies to do gradient update

$$J_e(\boldsymbol{\mu}_i) = \mathbb{E}_{k\sim\text{unif}(1,K),s\sim p^{\boldsymbol{\mu}},a\sim\boldsymbol{\mu}_i^{(k)}}\left[R_i(s,a)\right]$$

$$\nabla_{\theta_i^{(k)}} J_e(\boldsymbol{\mu}_i) = \frac{1}{K}\mathbb{E}_{\mathbf{x},a\sim\mathcal{D}_i^{(k)}}\left[\nabla_{\theta_i^{(k)}}\boldsymbol{\mu}_i^{(k)}(a_i|o_i)\nabla_{a_i}Q^{\boldsymbol{\mu}_i}\left(\mathbf{x},a_1,\ldots,a_N\right)\Big|_{a_i=\boldsymbol{\mu}_i^{(k)}(o_i)}\right].$$

Experiments

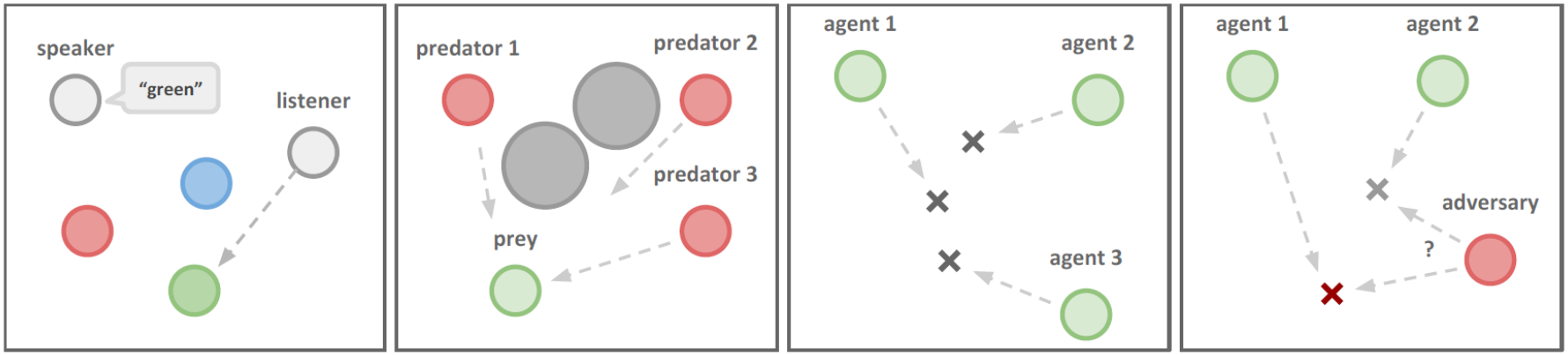


Figure 2: Illustrations of the experimental environment and some tasks we consider, including a) *Cooperative Communication* b) *Predator-Prey* c) *Cooperative Navigation* d) *Physical Deception*. See webpage for videos of all experimental results.

Experiments

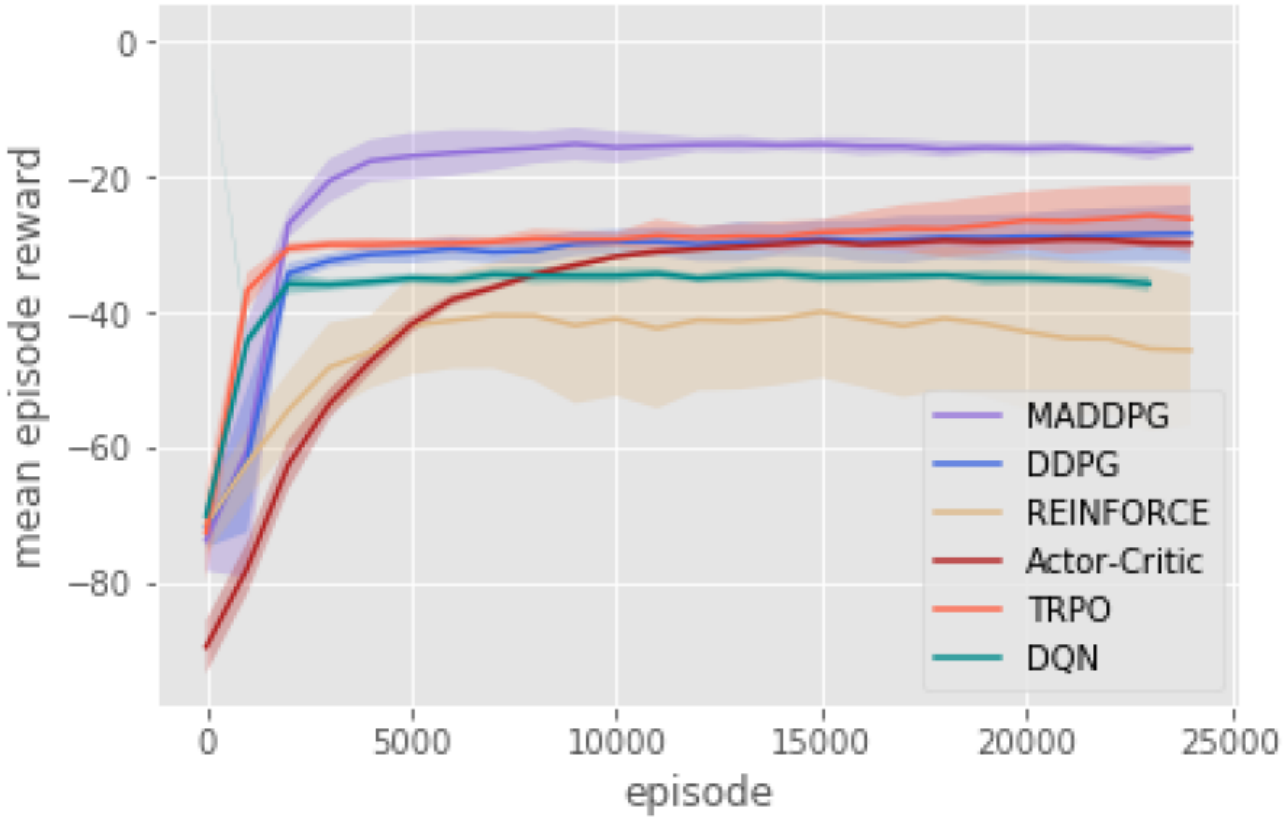
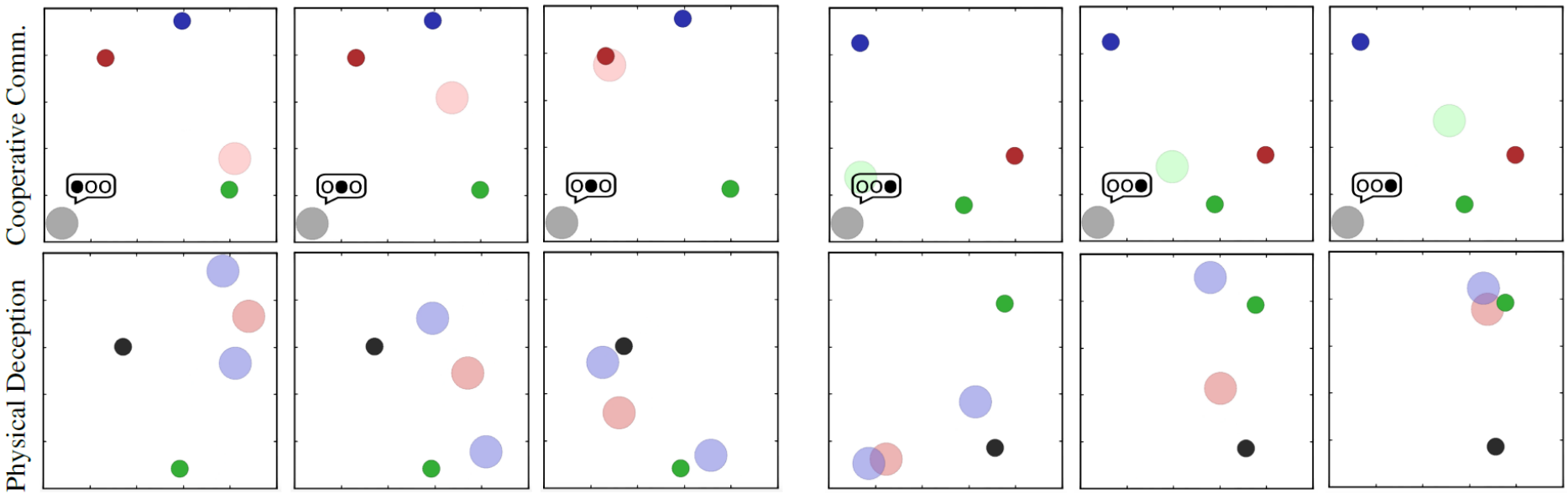


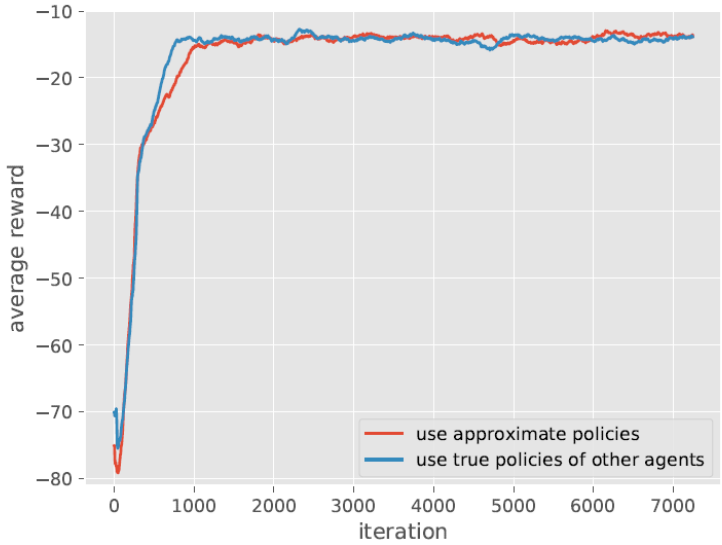
Figure 4: The reward of MADDPG against traditional RL approaches on cooperative communication after 25000 episodes.

Experiments



(a) MADDPG (b) DDPG

Experiments



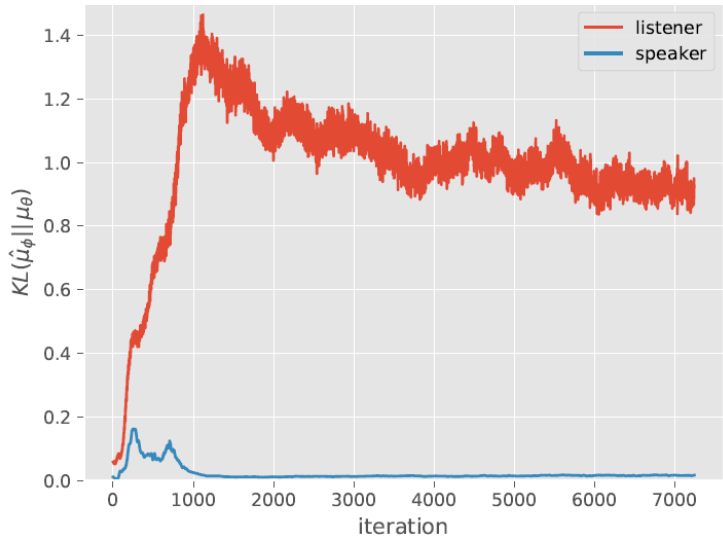


Figure 6: Effectiveness of learning by approximating policies of other agents in the cooperative communication scenario. *Left:* plot of the reward over number of iterations; MADDPG agents quickly learn to solve the task when approximating the policies of others. *Right:* KL divergence between the approximate policies and the true policies.

References

- https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html
- Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments
- Asynchronous Methods for Deep Reinforcement Learning
- https://sites.google.com/site/multiagentac/
- https://github.com/openai/multiagent-particle-envs