Robust Multi-agent Q-learning in Cooperative Games with Adversaries

Eleni Nisioti, Daan Bloembergen, Michael Kaisers
Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

Abstract

We present RoM-Q [1], a new Q-learning-like algorithm for finding policies robust to attacks in multi-agent systems (MAS). We consider a novel type of attack, where a team of adversaries, aware of the optimal multi-agent Q-value function, performs a worst-case selection of both the agents to attack and the actions to perform. Our motivation lies in real-world MAS where vulnerabilities of particular agents emerge due to their characteristics and robust policies need to be learned without requiring the simulation of attacks during training. In our simulations, where we train policies using RoM-Q, Q-learning and minimax-Q and derive corresponding adversarial attacks, we observe that policies learned using RoM-Q are more robust, as they accrue the highest rewards against all considered adversarial attacks.

Introduction

Many real-world problems involve multi-agent systems (MAS), where uncertainty is an emerging property arising due to interactions among agents, rather than from external model misspecification. Reinforcement learning (RL) algorithms are increasingly being used to optimize the operation of different types of MAS, with most recent prominent examples being complex multi-player computer games (OpenAI et al. 2019; Vinyals et al. 2019). In multi-agent RL (MARL), agents co-evolve and optimize policies based on their expectation of rewards experienced in their environment. As rewards depend on the interactions between agents, multi-agent policies are not inherently *robust*: a misalignment of the behavior of a small number of agents between the training and evaluation of a MAS can lead to arbitrarily bad performance.

Robustness is a long-standing pursuit in the control and reinforcement learning theory (Zhou, Doyle, and Glover 1996; Morimoto and Doya 2005). While single-agent approaches pursue robustness during learning or planning by considering stochastic perturbations in transition probabilities and rewards (Abbasi Yadkori et al. 2013; Mohammed et al. 2019) or time-variant Markov dynamics (Lecarpentier and Rachelson 2019), MARL studies robustness primarily in terms of performance in the presence of different types of agents. All robust approaches share, however, a common ground: the environment is governed by some sort of uncertainty. In MARL, a policy is considered robust when agents perform well in various multi-agent environments, not necessarily encountered during the training process.

Minimax decision rules are a common tool for designing robust policies in MARL (Littman 1994; Li et al. 2019). Devised in game theory to compute best-response policies in zero-sum games (von Neumann and Morgenstern 1947), minimax decision rules can be straightforwardly adopted to design agents acting in their best interests by best-responding to other agents that behave as zero-sum opponents (Littman 1994). As is customary, in this work we refer to agents behaving as zero-sum opponents as adversaries.

Our motivation primarily lies in *critical* multi-agent systems: e.g., the operation of communication networks and power grids is characterized by highly undesirable unsafe regions, as they are associated with loss of information due to over-flows or physical damage of components due to overloads. Policies for such systems are traditionally learned offline under the assumption that all agents aim at maximizing a common objective. This leads to policies that are not necessarily robust to misbehavior of even single agents and, therefore, may remain vulnerable to attacks during deployment. In this work, we focus on misbehavior due to a fixed number of adversaries that arrive at some random time step in the system, perform an adversarial selection of target agents and, then, directly manipulate their actions. We refer to this type of attack as a *multi-agent adversarial attack*. This is a novel type of attack that extends the classical notion of worst-case selection of actions in an adversarial attack to an adversarial choice among multiple agents. It is of particular interest in MAS where individual agents may be more vulnerable or vital to the operation of a network, as their misbehavior can cause cascading failures.

To learn robust multi-agent policies we design a temporal difference learning algorithm, where the value of the target policy is computed assuming that a multi-agent adversarial attack is taking place. We refer to this algorithm as robust multi-agent Q-learning (RoM-Q). Training using RoM-Q is performed in a centralized manner, where an agent observ-

[1] The implementation of RoM-Q and simulations conducted for this paper can be found at https://github.com/eleninisioti/robust-marl

ing the MAS states and actions learns a Q-value function in the joint state-action space. To evaluate the target policy, the agent considers all possible selections of a given number of adversaries and updates the Q-value function for the worst-case selection. Thus, the learning update requires the solution of different linear programs, each one following the form used by minimax-Q, whose number depends on the considered number of agents and adversaries. During training, each agent employs an ϵ-greedy policy, while during evaluation, the greedy policy is executed.

Simulations are performed on a stylized problem setting capturing the abstract essence of load balancing. Inspired by real-world systems such as computer networks and the smart grid, we model the MAS as a network of nodes, assigned with the execution of tasks, that aims at minimizing the cost of its operation and, most importantly, avoid an over-flow. We use the terms nodes and agents interchangeably.

Related work

When studying robustness in MARL, there are two essentially distinct ways to view adversaries. The *Bayesian approach* views adversaries as players in a game played between themselves and the cooperative agents comprising the system (Johanson, Zinkevich, and Bowling 2008). Under this direction, agents need to follow some form of opponent-aware reinforcement learning and attempt to learn policies that converge to the Nash equilibria of a general-sum or zero-sum game. The second approach computes *best-response policies* to attacks based on the agents' current estimation of the values of the optimal policy that can be found a priori. A robust operator "imagines" attacks during training, in order to come up with a policy robust to this type of attack without requiring the simulation of attacks during training (Klima et al. 2019). As the effect of an attack is calculated based on an agent's own value function, a model of the adversary is not required.

When studying robustnes in MARL, in addition to the policies followed by adversaries, we also need to define the type of attack. We consider that policies define the actions that an adversary performs when controlling an agent, while the type of attack includes all other parameters required to fully describe the attack, which, among others, may include the number of attackers, selection of agents and probability of occurrence. We can argue that safety is in the eye of the designer, with different approaches to robustness anticipating different types of attacks. In the spirit of the recent deep reinforcement learning bloom, adversaries often manipulate the observations of agents with the aim of fooling the function approximators used for decision making. This type of attack can be at most as effective as the direct manipulation of actions considered in our single-step multi-agent adversarial attack. The work in (Lin et al. 2020) introduces a novel attack in cooperative systems, where an attacker first uses reinforcement learning to find the actions that will have the worst long-term effect on the system reward and, then, manipulates the observations of an agent to lead the victim to taking the wrong action. In contrast, our attack is short-sighted, choosing the actions that will bring the lowest reward in the current state. This is more appropriate in critical systems, where adversarial attacks attempt to bring the largest damage in a short time span.

A variety of RL algorithms have already been combined with minimax updates. The stage was set by the seminal work of (Littman 1994) that introduced minimax-Q by blending the framework of MDPs with Markov games. Minimax policy gradients were introduced in (Li et al. 2019) to ensure robustness to various types of opponents in a multi-agent setting with a continuous action space. Robust temporal difference learning (Klima et al. 2019) considered security games where agents are attacked with a certain probability and modified classical temporal difference learning with minimax updates. Our work resembles this approach, as we also define a temporal difference algorithm for being robust to a certain type of attack. However, the type of attack that we consider here differs from the one in (Klima et al. 2019): adversaries in our formulation come in a certain number and find the most vulnerable agents to attack, instead of assuming that all agents are attacked with an equal probability.

Background

For our mathematical notation, we use $\mathcal{S}$ to represent a set, $\vec{s}$ for a vector, lower-case letters for random variables and upper-case letters for functions. We only use vector notation when referring to variables of a MAS[2]. We indicate that a variable refers to agent i with a subscript, and use superscripts to denote other indexes, such as the time step.

Temporal difference learning Single-agent reinforcement learning agents interact with an environment that can be modeled as a Markov Decision Process (MDP) (Sutton and Barto 1998). At a given point in time, an agent is in a state s, performs an action a and observes a reward r and the next state s'. The environment is characterized by the reward function $R(s, a)$ and the transition probability function $T(s, a, s')$. In temporal difference learning, an agent computes the optimal policy in the absence of knowledge of $R(s, a)$ and $T(s, a, s')$. To achieve this, the agent continuously refines its estimation of the Q-function, which captures the expected reward for any given state and action, by interacting with the environment. The update equation for Q-values is:

$$Q^{\pi}(s,a) = Q^{\pi}(s,a) + \alpha\big(r + \gamma V^{T}(s') - Q^{\pi}(s,a)\big) \quad (1)$$

where α is the learning rate, quantifying how quickly the agent forgets information about the past, and γ is the discount factor, quantifying how much the agent discounts future information. The quantity in the parenthesis is the temporal difference error and denotes the difference between the previous estimate of the policy and the improved estimate, after experience gathered in the current time step. $V^{T}(s')$ is termed the value of the target policy and differs among learning algorithms. In Q-learning the value of the target policy is equal to $\max_a Q(s', a)$.

[2]We ignore the fact that a single agent can have a multi-dimensional state or action space while the state of a MAS can be one-dimensional, as it is not relevant to our discussion.

Stochastic games Stochastic games offer a learning framework for a MAS. They can be seen as an augmentation of repeated games with states and one-step dynamics described by an MDP. At a given point in time, the MAS is in a state $\vec{s}$ that gives rise to a game played between N agents. To model a MAS as a zero-sum game, we can assume that agents are divided into D defenders and K adversaries. Then, $\pi_D(\vec{s})$ denotes the joint policy followed by defenders, which is a mapping from the joint state space $\mathcal{S}_D$ to the joint action space $\mathcal{A}_D$. Equivalently, $\pi_A(\vec{s})$ denotes the joint policy followed by adversaries.

The value of a state $\vec{s}$ from the perspective of a defender under the target policy can be computed as:

$$V^T(\vec{s}) = \max_{\pi_D(\vec{s})} \min_{\vec{a}_K} \sum_{\vec{a}_D} Q(\vec{s}, \vec{a}_K, \vec{a}_D)\pi_D(\vec{a}_D|\vec{s}) \quad (2)$$

Minimax-Q Minimax-Q is an extension of Q-learning to zero-sum stochastic games, where the policies of two players are proven to converge to the Nash Equilibrium of the game (Littman 1994). The target value in this case is as defined in Eq. (2). At each learning iteration, the player chooses an action to execute based on its policy $\pi(s)$ and the exploration scheme, observes the state reward, the transition of the game state and the action of the adversary a_K, and updates its Q-value function based on its current estimation of the value function. It then uses linear programming to update the policy and value function as follows:

$$\pi(s) = \arg\max_{\pi'_D(s)} \min_{a'_K} \sum_{a'_D} Q(s, a'_K, a'_D)\pi'_D(\vec{a}_D|s) \quad (3)$$

$$V^\pi(s) = \min_{a'_K} \sum_{a'_D} Q(s, a'_K, a'_D)\pi_D(\vec{a}_D|s) \quad (4)$$

Robust Multi-agent Q-learning

In this section we present RoM-Q, a MARL algorithm for learning policies robust to multi-agent adversarial attacks.

In our formulation, N agents are defenders, with each agent $i \in [1, \cdots, N]$ performing actions based on the policy defined over its own action space and the system state. We denote the set of possible partitions of agents into K adversaries and D defenders as $\mathcal{C}_K$, while c_K denotes such a partition. We employ temporal difference learning, as presented in (1), and define the value of the target policy as:

$$V^T(\vec{s}) = \min_{\vec{a}_j} \min_{j \in \mathcal{C}_K} \max_{\pi_{-j}(\vec{s})} \sum_{\vec{a}_{-j}} Q(\vec{s}, \vec{a}_{-j}, \vec{a}_j)\pi_{-j}(\vec{a}_D|\vec{s}) \quad (5)$$

where j refers to a set of agents and $-j$ refers to all agents not belonging in this set ($-j = \mathcal{N}/j$). The two minimizers model the adversarial selection of both the target agents ($j \in C_k$) and their actions (a_j), assuming that all other agents stick to their intended policy ($\max_{\pi_{-j}}$). Thus, our update is similar to the one performed by minimax-Q, with one notable difference: the type of an agent is not known a priori. Instead, agents are attacked adversarially, by picking the partition that will give the minimum Q-value. While minimax-Q requires solving one linear program per sampling step, our addition of minimizing over all possible partitions renders the calculation of $V^T(\vec{s})$ a mixed integer linear program. RoM-Q finds the optimal solution of Eq. (5) by solving a linear program for each possible partition c_K, the number of possible partitions being $(N!)/(K-N)!$. This exhaustive enumeration entails high complexity in general, but was not prohibitive for the simulations performed on the toy network examined in this paper. Future work may improve scaling by approximating solutions with sampling.

Algorithm 1: RoM-Q

Data: $\mathcal{N}, \mathcal{S}, \mathcal{A}, \alpha, \gamma, \epsilon, K, h$
Result: N policies $\pi_i(s)$

1 **for** $\vec{s} \in \mathcal{S}, \vec{a} \in \mathcal{A}, i \in \mathcal{N}$ **do**
2 Initialize $Q(\vec{s}, \vec{a}), V(\vec{s}), \pi_i(\vec{s})$
3 **end**
4 **while** *learning has not converged* **do**
5 with probability ϵ perform random action $\vec{a}$ otherwise, select actions $\vec{a} = \arg\max Q(\vec{s}, \vec{a})$
6 Observe reward r, next state $\vec{s}'$
7 **if** s' *is terminal* **then**
8 $target = r$
9 Sample new initial state s'
10 **else**
11 $target = r + \gamma V(\vec{s}')$
12 **end**
13 $Q(\vec{s}, \vec{a}) = (1-\alpha)Q(\vec{s}, \vec{a}) + \alpha(target)$
14 Find all possible subsets c_k of K adversaries out of N agents
15 **for** $c_k \in \mathcal{C}$ **do**
16 $\pi_{-c_k}(\vec{s}) = \arg\max_{\pi_{-c_k}} \min_{\vec{a}_{c_k}} \sum_{\vec{a}_{-c_k}} \pi_{-c_k}(\vec{s})Q(\vec{s}, \vec{a}_{c_k}, \vec{a}_{-c_k})$
17 $V_{-c_k}(\vec{s}) = \min_{\vec{a}_{c_k}} \sum_{\vec{a}_{-c_k}} \pi_{-c_k}[\vec{s}]Q(\vec{s}, \vec{a}_{c_k}, \vec{a}_{-c_k})$
18 **end**
19 Choose adversarial set $\bar{c_k} = \arg\min_{c_k}(V_{-c_k}(\vec{s}))$
20 **for** *node* $i \in \{\mathcal{N} - c_k\}$ **do**
21 $\pi_i(\vec{s}) = \pi_{-\bar{c}_k[i]}(\vec{s})$
22 **end**
23 $V(\vec{s}) = \min_{\vec{a}_{\bar{c}_k}} \sum_{\vec{a}_{-\bar{c}_k}} \pi_{-\bar{c}_k}(\vec{s})Q(\vec{s}, \vec{a}_{\bar{c}_k}, \vec{a}_{-\bar{c}_k})$
24 **end**

Algorithm 1 presents the pseudocode for RoM-Q, which requires as input the set of agents $\mathcal{N}$, the state and action space $(\mathcal{S}, \mathcal{A})$, the learning hyper-parameters $(\alpha, \gamma, \epsilon)$, the size of the adversarial attack K and the problem horizon (h). At each time step, the MAS performs a random or greedy action (line 5), observes the next state and reward (line 6) and updates its Q-value function and value function (lines 13-23). In lines (7-10) the MAS is reset due to reaching a terminal state. In our case, a state is terminal when one of the agents has over-flown or the MAS has survived for h consecutive time steps. While updating the Q-value function (line 13) requires a single update, updating the value function requires computing the value of the target policy for every possible partition of the MAS into defenders and adversaries (lines 15-18) and then updating based on the partition with the lowest value (19-23).

The multi-agent adversarial attack

In this section, we describe the type of attack that inspired the design of RoM-Q. In general, this attack can be used during evaluation against any policy and is not a component of the RoM-Q algorithm.

An attack is essentially a deterministic policy that maps states to agent and action selections. We denote this policy as $\sigma(\vec{s}) = (c_k, \vec{a}_{c_k})$, where c_k denotes the set of nodes selected for attack and $\vec{a}_{c_k}$ the actions they will perform. We consider that adversaries, which arrive during evaluation with a certain probability δ, select both nodes and actions adversarially. These two selection steps are not independent: the adversarial selection of nodes takes into account the effect of the performed actions, so that the attack brings about the largest decrease in the immediate reward in hindsight, i.e. after the remaining defenders perform actions according to the learned policy.

We assume that training has completed and the MAS has learned a Q-value function $Q(\vec{s}, \vec{a})$ (which is not necessarily the optimal one). To compute the optimal adversarial attack, $\sigma^*(\vec{s})$, we consider all possible partitions $\mathcal{C}$ of the N agents into K adversaries and $N - K$ defenders, denoted as $\mathcal{C}_K$, and, for each partition $c_K \in \mathcal{C}$ solve the following problem:

$$\min_{c_K} V_{-c_K}(\vec{s}) = \min_{\vec{a}_{c_K}} \max_{\vec{a}_{-c_K}} Q^*(\vec{s}, \vec{a}_{-c_k}, \vec{a}_{c_K}) \qquad (6)$$

This means that adversaries marginalize over the actions of defenders and, then, perform the actions that will incur the minimum expected reward, based on the optimal joint policy. We denote this quantity as $V_{c_k}(s)$, as it is a type of value function, indicating the expected reward when adversarial agents and defenders perform their actions and the optimal policy is followed thereafter. We present the pseudocode for finding the optimal adversarial policy $\sigma^*(\vec{s})$ for a given learned Q-value function $Q^*(\vec{s}, \vec{a})$ in Algorithm 2. We should emphasize that the adversarial policy is deterministic: as no learning takes place during deployment and policy π^* is deterministic, computing a probabilistic adversarial policy is not required. Thus, the value function computed by the adversarial attack (line 4 in Algorithm 2) differs from the value of the target policy in line 23 of Algorithm 1, which was computed assuming that defenders follow a probabilistic policy.

Experiments and results

In this section we evaluate the ability of RoM-Q to find multi-agent policies that are robust to multi-agent adversarial attacks. We experiment with a load balancing problem for a toy network characterized by a critical over-flow area. We compare the performance of policies learned by a multi-agent system using RoM-Q to the performance of policies learned by Q-learning and minimax-Q, where evaluation takes into account the ability of these methods to get optimal rewards in the absence of attacks, as well as their ability to perform well when attacks take place.

Load balancing To simplify the analysis and reach intuitive conclusions, we limit simulations in this paper to a toy network consisting of two inter-connected nodes, as presented in Fig. 1. Each node i is modeled by its capacity c_i, i.e. the maximum number of tasks it can hold, and the probability of arrival of a new task in a given time step, r_i^{arr}. The number of tasks currently stored in a node comprise its state and the system state consists of the states of all nodes. Nodes are capable of executing a task using the action a_i^{exec} and off-loading a task using the action a_i^{off}. Executing a task incurs a penalty p_i^{exec}, while off-loading a task from node i to node j incurs a penalty $p_{i,j}^{\text{off}}$. Finally, over-flows incur a disproportionately large penalty p_i^{over}. In addition to these costs received per time step, nodes also receive a reward u_i if the node has not over-flown in the current time step. This toy problem can be viewed as a multi-agent variant of the classical cliff walking problem, where the cliff region is determined by the capacities of the nodes.

Training consists of S_{train} learning samples. If one of the nodes over-flows, all nodes are reset to a zero state. At the end of the training process, we find an optimal policy $\pi_i^*(\vec{s})$ for each node i. Policies are defined over the joint state space and the action space of that node, which is the Cartesian product of its two actions, i.e. $\pi_i(s) : \mathcal{S} \rightarrow \mathcal{A}_i^e \times \mathcal{A}_i^o$. No attacks take place during training.

Evaluation requires the definition of the adversarial policy, which is used to perform attacks during S_{eval} evaluation steps. At each step, a greedy multi-agent policy $\pi^*(\vec{s})$ competes against a greedy adversarial policy $\sigma^*(\vec{s})$. Adversaries arrive randomly with a probability δ in a given evaluation step. In our simulations, we consider that $K = 1$ adversaries may arrive during evaluation.

Algorithm 2: Computing the optimal policy of a multi-agent adversarial attack.

Data: $Q^*(\vec{s}, \vec{a}), K$
Result: $\sigma^*(\vec{s}), \forall \vec{s} \in \mathcal{S}$
1 Find all possible subsets c_K of K adversaries out of N nodes
2 **for** $\vec{s} \in \mathcal{S}$ **do**
3 **for** $c_k \in \mathcal{C}$ **do**
4 $V_{c_k}(\vec{s}) = \min_{a_{c_k}} \max_{a_{-c_k}} Q^*(\vec{s}, \vec{a}_{c_k}, \vec{a}_{-c_k})$
5 **end**
6 Choose adversarial set $\bar{c}_k = \arg\min_{c_k}(V_{c_k}(\vec{s}))$
7 $\vec{a}_{\bar{c}_k} = \arg\min_{\vec{a}_{\bar{c}_k}} \max_{-a_{c_k}} Q^*(\vec{s}, \vec{a}_{\bar{c}_k}, \vec{a}_{-\bar{c}_k})$
8 $\sigma^*(\vec{s}) = [\bar{c}_k, \vec{a}_{\bar{c}_k}]$
9 **end**

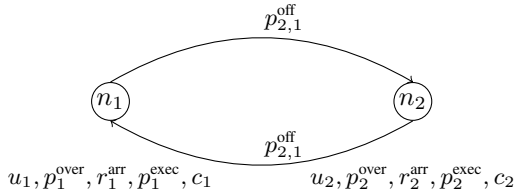

Figure 1: Schematic of a toy network consisting of two nodes.

Table 1: Learning hyper-parameters

Parameter	Value
training samples S_{train}	1000000
evaluation samples S_{eval}	20000
trials I	40
learning rate α	0.01
exploration rate ϵ	0.1
discount factor γ	0.9

Table 2: Network modeling parameters

Parameter	Value
reward u	[8, 8]
over-flow penalty p^{over}	[100,100]
capacity c	[3,3]
arrival rate r^{arr}	[0.5, 0.5]
execution penalty p^{exec}	[4,1]
off-loading penalty p^{off}	[2,2]
number of adversaries K	1

We generate a pool of adversarial policies for evaluation: $\{\sigma^*_{\text{Q-learning}}(s), \sigma^*_{\text{minimax-Q}}(s), \sigma^*_{\text{RoM-Q}}(s)\}$. These are the adversarial policies found using Algorithm 2 based on the optimal policies learned using Q-learning ($\pi^*_{\text{Q-learning}}$), minimax-Q ($\pi^*_{\text{minimax-Q}}$) and RoM-Q (($\pi^*_{\text{RoM-Q}}$)), respectively. In addition to evaluating the optimal policy learned by each method, we also save intermediate policies during training to examine how training converges to robust solutions. Finally, we compute confidence intervals at a 95% confidence level, presented in plots as shaded regions, by performing I independent training and evaluation trials, where the number of the trial is used as a seed. We present all hyper-parameters associated with learning in Table 1 and all parameters related to the modeling of the nodes and system in Table 2. By varying these modeling parameters we can examine a wide range of problem settings, where the properties of nodes give rise to different vulnerabilities and optimal policies. In our current simulations, we have opted for a problem setting where node 1 receives a higher penalty for executing than off-loading a packet and node 2 has a higher cost for off-loading than executing. To make comparisons easier, we choose the same arrival rate for the two nodes and set it to a value low enough to ensure that nodes do not need to constantly execute tasks to avoid an over-flow.

Learning optimal and adversarial policies

We now analyze the different learning methods in terms of their ability to find policies with optimal rewards in the absence of attacks. We also visualize these optimal policies, the state visits when following them during evaluation, and their optimal adversarial policy.

The heatmaps in the left column of Fig. 2 illustrate the number of visits per state using color, as well as the optimal policies of the two nodes, using arrows. As we only show visits after convergence to the optimal policy, some states are never visited. There are two arrows starting from each square of the state space: the green one visualizes the actions of n_1 and the orange one the actions of n_2. When

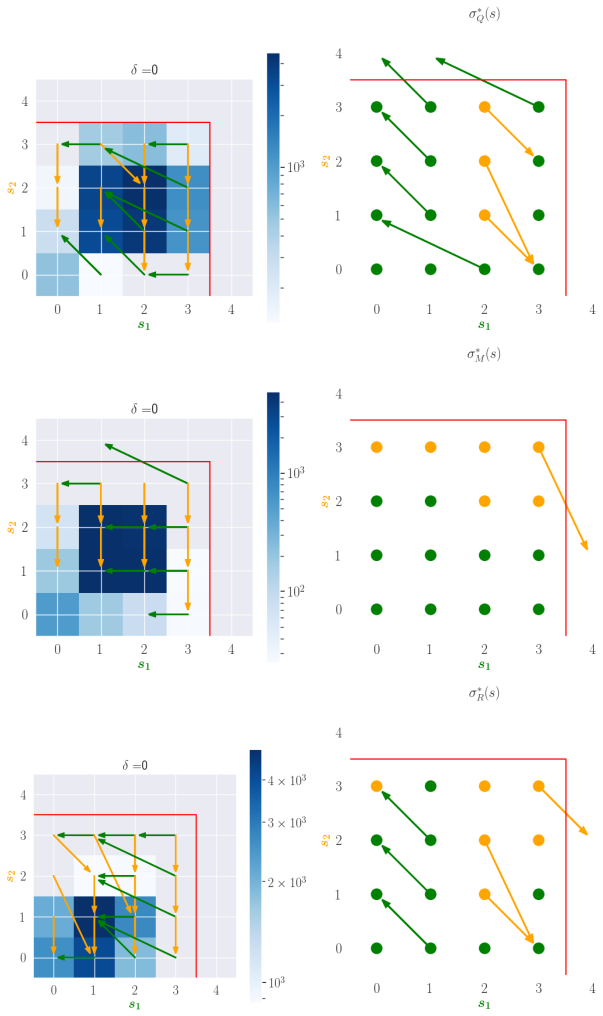

Figure 2: Heatmap of state visits and actions of the optimal policy (left) and the adversarial policies (right) for Q-learning, minimax-Q and RoM-Q (from top to bottom).

node i executes a task, i.e. $a^{\text{e}}_i = 1$, the arrow points to a decrease in the value of its state (leftwards for n_1 and downwards for n_2). Similarly, when node i off-loads a task, i.e. $a^{\text{off}}_i = 1$, the arrow points to an increase in the value of the other node's state (upwards for n_1 and rightwards for n_2) and a decrease in the value of its own state. For states that have no indicated action, the optimal action is $\vec{a} = [a^e_1 = 0, a^o_1 = 0, a^e_2 = 0, a^o_2 = 0]$. The right column presents the respective optimal adversarial policies $\sigma^*_{\text{Q-learning}}(s), \sigma^*_{\text{minimax-Q}}(s), \sigma^*_{\text{RoM-Q}}(s)$, where, in each state, the node chosen to be controlled by the adversary and the adversarial actions are shown in the appropriate color.

We observe that none of the three methods over-flows in the absence of attacks. Q-learning learns policies that keep the two nodes close to capacity, as nodes prefer to remain inactive when they have a small number of tasks, in order to avoid penalties associated with the off-loading or execution of tasks. The adversarial policy chooses to attack node

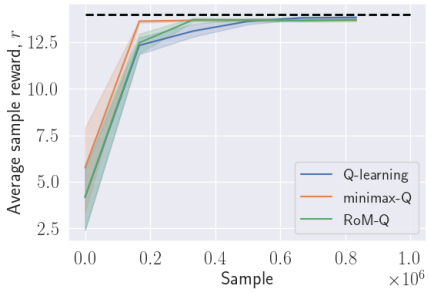

Figure 3: Evaluating Q-learning, minimax-Q and RoM-Q in terms of the improvement of the average system reward per sample during training. The optimal solution is indicated with a dashed black line.

1 when its load is low, while node 2 is attacked when node 1 has a high load. Thus, adversaries following $\sigma^*_{\text{Q-learning}}$ attack nodes in order to manipulate them into over-flowing the other node. We can also observe that adversaries sometimes choose to execute tasks: although this appears to come in contrast to the adversary's objective of over-flowing the system, it incurs an additional cost, and can therefore be worst-case for an one-step attacker. Minimax-Q learns a more conservative policy, restricting nodes to visiting states with low and intermediate node values. The optimal adversary for minimax-Q, $\sigma^*_{\text{minimax-Q}}$, has the objective of over-flowing nodes by keeping them idle. Finally, RoM-Q learns a policy where executions are frequent for both nodes. An important difference between the policy learned by minimax-Q and RoM-Q is that the former keeps the nodes idle when their state is equal to 1. As we will see later in this section, this makes minimax-Q less robust to attacks taking place during evaluation.

To get a better understanding of the learning process, we visualize the improvement in the performance of policies during training time. In Figure 3, we present the evolution of rewards, averaged over I trials, with training time measured in samples of experience. Performance is measured as the average system reward per evaluation sample. As the duration of an episode is 50 and, based on the values in Table 2, the optimal sum of rewards accrued during an episode is 700, the optimal value for the average reward is 14. In order to examine the ability of the policies to avoid over-flow during the course of an evaluation episode, we have measured the evolution of the duration of episodes and observed that all algorithms learn a policy that prevents over-flows after around $0.2 \cdot 10^6$ samples.

Behavior under attacks

In this section we visualize the behavior of the learned policies when attacks take place during evaluation. Note that the actions presented in this section are the ones under the optimal policy, and differ from the ones executed during evaluation, as, with probability $\delta = 1$, an adversary is controlling one of the two nodes. The adversarial policy used to produce all heatmaps in Figure 4 is the one depicted on the bottom right of Figure 2.

In Figure 4, we observe that Q-learning over-flows often when being under attack. This is because its optimal policy keeps both nodes close to capacity. Similarly, state visits under minimax-Q visit the over-flow area, but with smaller frequency. In contrast, RoM-Q remains in the safe region most of the time despite attacks taking place. We also observe that the policy learned using Q-learning experiences more over-flows for the second node, while over-flows for minimax-Q are distributed more evenly between the two nodes. This is due to the adversarial policy, which attempts to over-flow node 2 by manipulating node 1 when the load is low. By being more robust than Q-learning, minimax-Q avoids some over-flows occurring early in the episode, but ultimately fails significantly more often than RoM-Q.

Robustness analysis

We now evaluate the performance of policies learned by the different algorithms against different adversarial policies, sampled from the following pool: $\{\sigma^*_{\text{Q-learning}}(s), \sigma^*_{\text{minimax-Q}}(s), \sigma^*_{\text{RoM-Q}}(s)\}$, and for different probabilities of attack, δ.

We can derive various interesting observations by closely inspecting Fig. 5. First, the performance of Q-learning drops drastically for $\delta > 0.1$ for any type of attack and exhibits larger variation. Second, minimax-Q and RoM-Q exhibit better robustness to attacks than Q-learning. Third, in the absence of attacks during evaluation, minimax-Q and RoM-Q achieve slightly lower rewards than Q-learning, due to the fact that their optimal policies are more conservative and nodes execute tasks often in order to stay far from the over-flow area. Finally, and most importantly, the policies learned using RoM-Q achieve the highest reward against all types of attacks and are thus the most robust.

Discussion and conclusions

We presented a reinforcement learning algorithm for learning policies robust to adversarial attacks taking place during deployment. Attacks in our framework occur with unknown probability and consist of a pre-determined number of adversaries that choose both agents and their actions adversarially. We demonstrated through simulations on a toy network that taking into account the different vulnerabilities of agents comprising the MAS is important when adversaries choose their victims adversarially.

A limitation of RoM-Q is the complexity entailed in solving at each learning iteration a number of linear programs that scales with the number of possible combinations of K adversaries out of N agents. Although policies are learned off-line, and can thus profit from simulation environments rich in resources, it would be useful to find a variant of RoM-Q with reduced complexity. If one considers that the learning update in RoM-Q involves a mixed integer linear program, a promising direction would be to express it in a standard form that allows for approximate solutions and comes with optimality guarantees, such as a multiple knapsack problem (Martello and Toth 1990), or to use (anytime) sampling approximations. Overall, we believe that our new approach defines a robustness to multi-agent adversarial at-

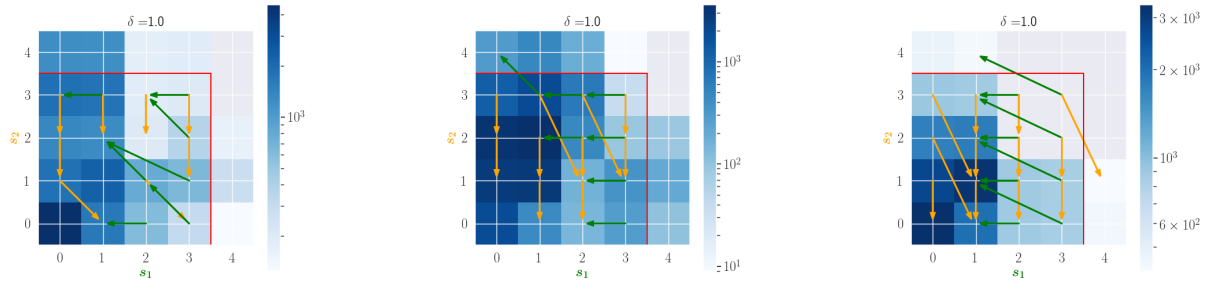

Figure 4: (From left to right): State visits and optimal actions for different probabilities of attack for Q-learning, minimax-Q and RoM-Q.

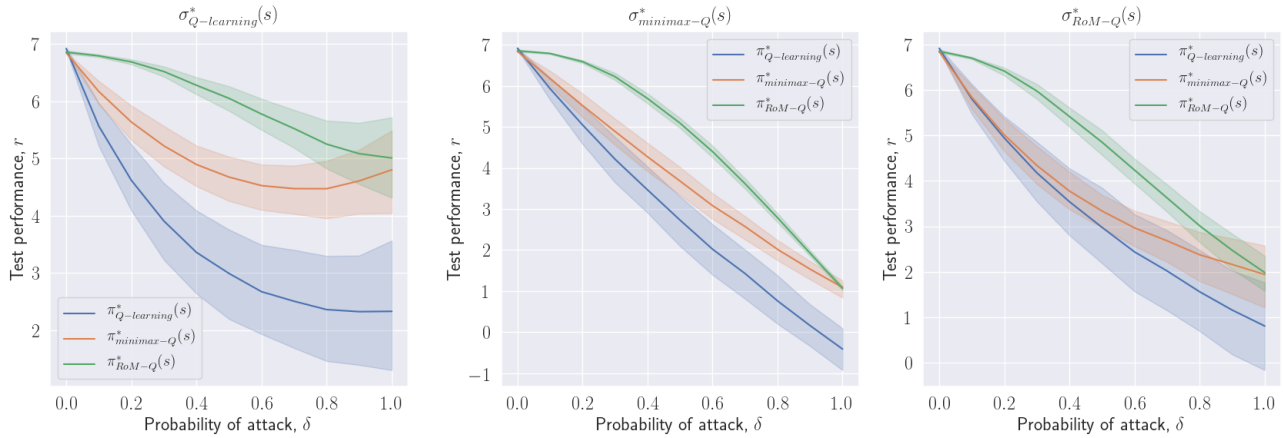

Figure 5: (From left to right): Performance measured as the total reward accrued in episode for different adversarial and optimal policies.

tacks that is both much needed and widely applicable for deploying multi-agent learning solutions to critical systems.

References

[Abbasi Yadkori et al. 2013] Abbasi Yadkori, Y.; Bartlett, P. L.; Kanade, V.; Seldin, Y.; and Szepesvari, C. 2013. Online learning in markov decision processes with adversarially chosen transition probability distributions. In *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc. 2508–2516.

[Johanson, Zinkevich, and Bowling 2008] Johanson, M.; Zinkevich, M.; and Bowling, M. 2008. Computing robust counter-strategies. In Platt, J. C.; Koller, D.; Singer, Y.; and Roweis, S. T., eds., *Advances in Neural Information Processing Systems 20*. Curran Associates, Inc. 721–728.

[Klima et al. 2019] Klima, R.; Bloembergen, D.; Kaisers, M.; and Tuyls, K. 2019. Robust temporal difference learning for critical domains. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 350–358.

[Lecarpentier and Rachelson 2019] Lecarpentier, E., and Rachelson, E. 2019. Non-stationary markov decision processes a worst-case approach using model-based reinforcement learning. In *NeurIPS*.

[Li et al. 2019] Li, S.; Wu, Y.; Cui, X.; Dong, H.; Fang, F.; and Russell, S. 2019. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. *Proceedings of the AAAI Conference on Artificial Intelligence* 33:4213–4220.

[Lin et al. 2020] Lin, J.; Dzeparoska, K.; Zhang, S. Q.; Leon-Garcia, A.; and Papernot, N. 2020. On the robustness of cooperative multi-agent reinforcement learning. *ArXiv* abs/2003.03722.

[Littman 1994] Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, 157–163. Morgan Kaufmann.

[Martello and Toth 1990] Martello, S., and Toth, P. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. USA: John Wiley & Sons, Inc.

[Mohammed et al. 2019] Mohammed, A.; Hang, R.; Haitham, B.-A.; Vladimir, M.; Rui, L.; Mingtian, Z.; and Jun, W. 2019. Wasserstein robust reinforcement learning. *ArXiv* abs/1907.13196.

[Morimoto and Doya 2005] Morimoto, J., and Doya, K. 2005. Robust reinforcement learning. *Neural Comput.* 17(2):335–359.

[OpenAI et al. 2019] OpenAI; :; Berner, C.; Brockman, G.;

Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; Józefowicz, R.; Gray, S.; Olsson, C.; Pachocki, J.; Petrov, M.; de Oliveira Pinto, H. P.; Raiman, J.; Salimans, T.; Schlatter, J.; Schneider, J.; Sidor, S.; Sutskever, I.; Tang, J.; Wolski, F.; and Zhang, S. 2019. Dota 2 with large scale deep reinforcement learning.

[Sutton and Barto 1998] Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edition.

[Vinyals et al. 2019] Vinyals, O.; Babuschkin, I.; Czarnecki, W.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.; Powell, R.; Ewalds, T.; Georgiev, P.; Oh, J.; Horgan, D.; Kroiss, M.; Danihelka, I.; Huang, A.; Sifre, L.; Cai, T.; Agapiou, J.; Jaderberg, M.; and Silver, D. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 575.

[von Neumann and Morgenstern 1947] von Neumann, J., and Morgenstern, O. 1947. *Theory of games and economic behavior*. Princeton University Press.

[Zhou, Doyle, and Glover 1996] Zhou, K.; Doyle, J. C.; and Glover, K. 1996. *Robust and Optimal Control*. USA: Prentice-Hall, Inc.