c-MBA: Adversarial Attack for Cooperative MARL Using Learned Dynamics Model

Nhan H. Pham
IBM Research, Thomas J. Watson Research Center
Yorktown Heights, NY, USA
{nhp}@ibm.com

Lam M. Nguyen
IBM Research, Thomas J. Watson Research Center
Yorktown Heights, NY, USA
{LamNguyen.MLTD}@ibm.com

Jie Chen
MIT-IBM Watson AI Lab, IBM Research
Cambridge, MA, USA
chenjie@us.ibm.com

Hoang Thanh Lam
IBM Research
Dublin, Ireland
t.l.hoang@ie.ibm.com

Subhro Das
MIT-IBM Watson AI Lab, IBM Research
Cambridge, MA, USA
subhro.das@ibm.com

Tsui-Wei Weng
Halıcıoğlu Data Science Institute, University of California San Diego
La Jolla, CA, USA
lweng@ucsd.edu

Abstract

In recent years, a proliferation of methods were developed for cooperative multi-agent reinforcement learning (c-MARL). However, the robustness of c-MARL agents against adversarial attacks has been rarely explored. In this paper, we propose to evaluate the robustness of c-MARL agents via a model-based approach, named **c-MBA**. Our proposed attack can craft much stronger adversarial state perturbations of c-MARL agents to lower total team rewards than existing model-free approaches. Our numerical experiments on two representative MARL benchmarks illustrate the advantage of our approach over other baselines: our model-based attack consistently outperforms other baselines in all tested environments.

1 Introduction

Deep neural networks are known to be vulnerable to adversarial examples, where a small and often imperceptible adversarial perturbation can easily fool the state-of-the-art deep neural network classifiers [22, 16, 4, 19]. Since then, a wide variety of deep learning tasks have been shown to also be vulnerable to adversarial attacks, ranging from various computer vision tasks to natural language processing tasks [7, 25, 8, 1].

While most of the existing DRL attack algorithms focus on the *single* DRL agent setting, in this work we propose to study the vulnerability of *multi-agent* DRL, which has been widely applied in many safety-critical real-world applications including swarm robotics [3], electricity distribution, and traffic control [18]. In particular, we focus on the collaborative multi-agent reinforcement learning (c-MARL) setting, where a group of agents is trained to generate joint actions to maximize the team

ML Safety Workshop, 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

reward. We note that c-MARL is a more challenging yet interesting setting than the *single* DRL agent setting, as now one also needs to consider the interactions between agents, which makes the problem becomes more complicated.

Our contribution can be summarized as follows:

- In this work, we propose the first *model-based* adversarial attack framework on c-MARL, where we name it **c-MBA** (**M**odel-**B**ased **A**ttack on **c**-MARL). We formulate the attack into a two-step process and solve for adversarial state perturbation efficiently by existing proximal gradient methods. We show that our model-based attack is stronger and more effective than all of existing model-*free* baselines.
- We show on both the multi-agent MuJoCo and multi-agent particle environments that our **c-MBA** consistently outperforms the SOTA baselines in all tested environments. We show that c-MBA can reduce the team reward up to $8-9\times$ when attacking the c-MARL agents. In addition, **c-MBA** with the proposed victim selection strategy matches or even outperforms other **c-MBA** variants in all environments with up to 80% of improvement on reducing team reward.

Paper outline. Section 2 discusses related works in adversarial attacks for DRL and present general background in c-MARL setting. We describe our proposed attack framework **c-MBA** in Section 3. Section 4 presents the evaluation of our approach on several standard c-MARL benchmarks. Finally, we summarize our results and future directions in Section 5.

2 Related work and background

Related work. Most of existing adversarial attacks on DRL agents are on *single* agent [6, 12, 9, 24] while there is only two other works [11, 5] that focus on the c-MARL setting. Whereas [5] considers a different problem than ours where they want to find an optimally "sparse" attack by finding an attack with minimal attack steps, [11] proposes a two-step attack procedure to generate state perturbation for c-MARL setting which is the most relevant to our work. However, there are two major differences between our work and [11]: (1) their attack is only evaluated under the StarCraft Multi-Agent Challenge (SMAC) environment [21] where the action spaces are discrete; (2) their approach is model-*free* as they do not involve learning the dynamics of the environment and instead propose to train an adversarial policy for a fixed agent to minimize the the total team rewards. The requirement on training an adversarial policy is impractical and expensive compared to learning the dynamics model. To the best of our knowledge, there has not been any work considering adversarial attacks on the c-MARL setting using model-based approach on continuous action spaces. In this paper, we perform adversarial attacks on agents trained using MADDPG [14] on two multi-agent benchmarks including multi-agent MuJoCo and multi-agent particle environments. Note that in the setting of adversarial attacks, once the agents are trained, policy parameters will be frozen and we do not require any retraining of the c-MARL agents during our attack.

Background in c-MARL. We consider multi-agent tasks with continuous action spaces modeled as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [17]. A Dec-POMDP has a finite set of agents $\mathcal{N} = \{1, \cdots, n\}$ associated with a set of states $\mathcal{S}$ describing global states, a set of continuous actions $\mathcal{A}_i$, and a set of individual state $\mathcal{S}_i$ for each agent $i \in \mathcal{N}$. Given the current state $s_t^i \in \mathcal{S}_i$, the action $a_t^i \in \mathcal{A}_i$ is selected by a parameterized policy $\pi^i : \mathcal{S}_i \to \mathcal{A}_i$. The next state for agent i is determined by the state transition function $\mathcal{P}_i : \mathcal{S} \times \mathcal{A}_i \to \mathcal{S}$, and agent i will receive a reward r_t^i calculated from a reward function $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_i \to \mathbb{R}$ and observe a new state $s_{t+1}^i \in \mathcal{S}_i$. In addition, Dec-POMDP is associated with an initial state distribution $\mathcal{P}_0$ and a discount factor γ. Training a c-MARL agent is to find a joint policy that maximize the total team rewards $\sum_{i,t} r_t^i$. Note that for the ease of exposition, we do not differentiate state and observation in this work and use them interchangeably throughout the paper.

3 c-MBA: Model-based attack for c-MARL

Our goal is to generate adversarial perturbations imposed to the victim agents' input (state) in order to deteriorate the total team reward. The added perturbations encourages the victim agents' state to be

close to a desired failure state corresponding to low reward. To avoid sampling from the environment, we use a pre-trained model that learns the dynamics of the environment to predict the next state from the perturbed state and current action, then find the suitable noise that minimizes the distance between the predicted next state and a predefined target state. In this section, we assume the target state is given and we consider learning this failure state in a data-driven approach in our future work. The overall attack can be formulated as an optimization problem as follows.

Formally, we consider a multi-agent setting with $|\mathcal{N}| = n$ agents, each agent $i \in \mathcal{N}$ receives state s_t^i locally and takes action a_t^i following the pre-trained c-MARL policy $\pi^i(s_t^i)$. Let $s_t = (s_t^1, \cdots, s_t^n) \in \mathcal{S}$ be the joint global state at time step t which is concatenated from local states s_t^i for each agent $i \in \mathcal{N}$. We also denote the joint action $a_t = (a_t^1, \cdots, a_t^n)$ concatenated from each agent's action a_t^i. Let $\mathcal{V}_t \subseteq \mathcal{N}$ be the set of victim agents at time step t, i.e. the set of agents that can be attacked. Let $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ be a parameterized function that approximates the dynamics of the environment, where $\mathcal{A}$ is the set of concatenated actions, one from each $\mathcal{A}_i$. Let s_{fail} be the targeted failure state which can lead to poor performance to the agent. We denote ε as an upper bound on budget constraint w.r.t some ℓ_p-norm $\|\cdot\|_p$. The state perturbation $\Delta s = (\Delta s^1, \cdots, \Delta s^n)$ (we suppress the dependence on t of Δs to avoid overloading the notation) to s_t is the solution to the following problem:

$$\begin{aligned} \min_{\Delta s = (\Delta s^1, \cdots, \Delta s^n)} \quad & d(\hat{s}_{t+1}, s_{fail}) \\ \text{s.t.} \quad & \hat{s}_{t+1} = f(s_t, a_t) \\ & a_t^i = \pi^i(s_t^i + \Delta s^i), \quad \forall i \in \mathcal{N} \\ & \Delta s^i = \mathbf{0}, \quad \forall\, i \notin \mathcal{V}_t \\ & \ell_{\mathcal{S}} \le s_t + \Delta s \le u_{\mathcal{S}} \\ & \left\|\Delta s^i\right\|_p \le \varepsilon, \quad \forall i \in \mathcal{V}_t \end{aligned} \tag{1}$$

where $\mathbf{0}$ is a zero vector, and the state vector follows a boxed constraint specified by $\ell_{\mathcal{S}}$ and $u_{\mathcal{S}}$.

Let us first provide some insights for the formulation (1). For each agent i, using the trained policy π^i, we can compute the corresponding action a_t^i given its (possibly perturbed) local state s_t^i or $s_t^i + \Delta s^i$. From the concatenated state-action pair (s_t, a_t), we can predict the next state $\hat{s}_{t+1}$ via the learned dynamics model f. Then by minimizing the distance between $\hat{s}_{t+1}$ and the targeted failure state s_{fail} subject to the budget constraint, we are forcing the victim agents to move closer to a damaging failure state in the next time step leading to low team reward.

Note that problem (1) can be reformulated as a constrained nonconvex problem which can be efficiently solved by first-order method to obtain a stationary point. We defer the details to Appendix A. Here, the convergence guarantee is that the perturbation found by solving the optimization problem (1) will make the next state (predicted by our dynamics model) closest to the failure state given the budget constraint. Finally, the full attack algorithm of c-MBA at timestep t can be summarized in Alg. 1.

Algorithm 1 c-MBA algorithm at timestep t

1: **Initialization:**
2: Given s_t, s_{fail}, π, f, $\mathcal{V}_t$; initialize $\Delta s = \varepsilon * sign(x)$ for $x \sim N(0, 1)$, attack budget ε, p; choose learning rate $\eta > 0$
3: **For** $k = 0, \cdots, K-1$ **do**
4: Compute $a_t = (a_t^1, \cdots, a_t^n)$ where $a_t^i = \pi^i(s_t^i + \Delta s^i)$ if $i \in \mathcal{V}_t$ and $a_t^i = \pi^i(s_t^i)$ otherwise.
5: Update Δs as $\Delta s_{k+1} = \text{proj}_{\mathcal{C}_{p,\varepsilon,t}}[\Delta s_k - \eta \nabla_{\Delta s} d(f(s_t, a_t), s_{fail})]$.
6: **End For**

One of the key enabler to solve (1) is the availability of the learned dynamics model f. If the dynamics is known, we can solve (1) easily with proximal gradient methods. However, in practice when we often do not have the full knowledge of the environment and in order to solve (1), we need to learn the dynamics model via some function approximator such as neural networks. We describe the process of training a dynamics model in Appendix B.

4 Experiments

We perform the attack on multi-agent MuJoCo (MA-MuJoCo) environments [2] including **Ant(4x2)**, **HalfCheetah(2x3)**, **HalfCheetah(6x1)**, and **Walker2d(2x3)**. The pair **name(config)** indicates the

name of MuJoCo environment along with the agent partition, where a configuration of 2x3 means there are in total 2 agents and each agent has 3 actions. We provide more details on the construction of these agents in Appendix C.

1. **Uniform**: the perturbation follows the Uniform distribution $U(-\varepsilon, \varepsilon)$.
2. **Gaussian**: the perturbation follows the Normal distribution $\mathcal{N}(0, \varepsilon)$.
3. **Lin et al. (2020) + iFGSM**: Since there is no other work performing adversarial attack for continuous action space in c-MARL, we adapt the approach in [11] to form another baseline.

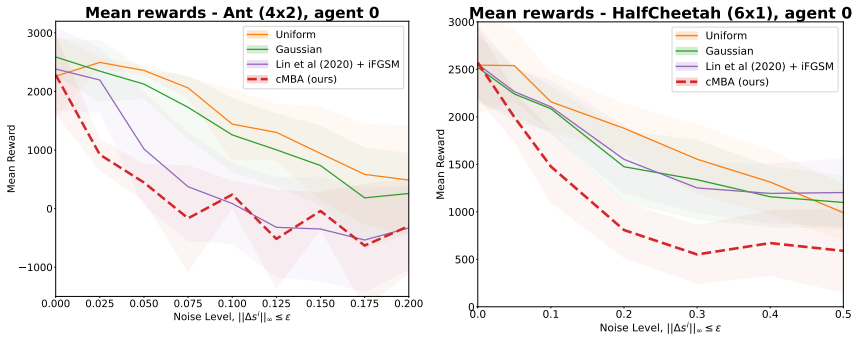

Figure 1: c-MBA vs baselines when attacking one agent in 2 MA-MuJoCo environments - **Exp. (I)**.

One key component to perform c-MBA attack is the definition of the failure state. We provide more details on the failure state corresponding to each environment in Appendix C. We evaluate c-MBA comprehensively using 3 experiments. Due to space constraint, we only present part of our results here and we defer the full results in Appendix D.

- **Experiment (I) – model-free baselines vs model-based attack c-MBA using ℓ_∞-constrained perturbation**: we compare c-MBA with other baselines when attacking individual agent under ℓ_∞ constraint.
- **Experiment (II) – attacking multiple agents using model-free baselines vs model-based attack c-MBA with ℓ_∞ perturbation**: we report results on attacking multiple agents simultaneously. This setting is not previously considered in [11].
- **Experiment (III) – model-free baselines vs model-based attack c-MBA on ℓ_1 perturbation**: we compare c-MBA with other baselines when attacking individual agent under ℓ_1 constraint. The results of this experiment are presented in Appendix D.

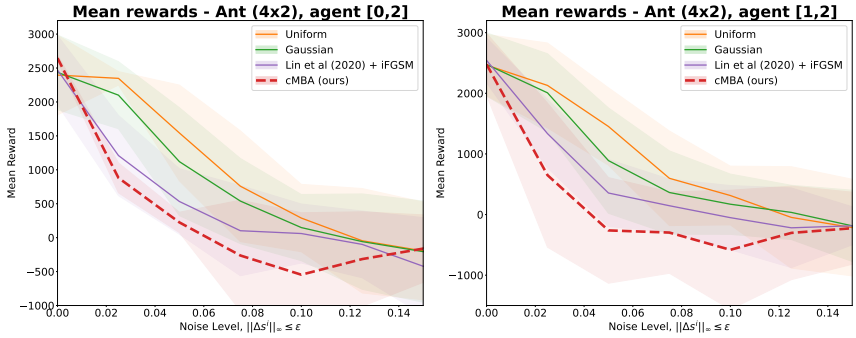

Figure 2: c-MBA vs baselines in Ant(4x2) when attacking two agents - **Exp. (II)**.

Experiment (I) – model-free baselines vs model-based attack c-MBA on ℓ_∞ perturbation. In this experiment, we run the 3 baseline attacks along with two variants of our model-based attack on the four MA-MuJoCo environments with one victim agent ($n_v = 1$). Fig. 1 illustrate the performance when we perform these attacks on each agent with different attack budget using ℓ_∞-norm where our model-based attack outperforms all the other baselines. In particular, our model-based attack yields much lower rewards under relatively low budget constraints (when $\varepsilon \in [0.05, 0.2]$) compared to other baselines.

Experiment (II) – attacking two agents using model-free baselines vs model-based attack c-MBA using ℓ_∞ constrained: We conduct experiments using model-free and model-based approaches to simultaneously attack two agents in Ant(4x2) environment. Fig. 2 illustrate the performance of various attacks. We also observe that our c-MBA attack outperforms other baselines.

5 Conclusions

In this paper, we propose a new attack algorithm named **c-MBA** for evaluating the robustness of c-MARL environment. Our c-MBA algorithm is the first model-based attack to craft adversarial observation perturbations and we have shown that c-MBA outperforms existing model-free baselines attack by a large margin undermulti-agent MuJoCo benchmarks.

References

[1] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*, 2018.

[2] C. S. de Witt. Multi-agent mujoco. `https://github.com/schroederdewitt/multiagent_mujoco`, 2020.

[3] Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. A taxonomy for swarm robots. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, volume 1, pages 441–447. IEEE, 1993.

[4] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[5] Yizheng Hu and Zhihua Zhang. Sparse adversarial attack in multi-agent reinforcement learning. *arXiv preprint arXiv:2205.09362*, 2022.

[6] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[7] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

[8] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8018–8025, 2020.

[9] Jernej Kos and Dawn Song. Delving into adversarial attacks on deep policies. *arXiv preprint arXiv:1705.06452*, 2017.

[10] Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al. Adversarial examples in the physical world, 2016.

[11] Jieyu Lin, Kristina Dzeparoska, Sai Qian Zhang, Alberto Leon-Garcia, and Nicolas Papernot. On the robustness of cooperative multi-agent reinforcement learning. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 62–68. IEEE, 2020.

[12] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. *arXiv preprint arXiv:1703.06748*, 2017.

[13] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[14] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

[15] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

[16] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.

[17] Frans A Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs*. Springer, 2016.

[18] Afshin OroojlooyJadid and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *arXiv preprint arXiv:1908.03963*, 2019.

[19] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[20] Bei Peng, Tabish Rashid, Christian A Schroeder de Witt, Pierre-Alexandre Kamienny, Philip HS Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients. *arXiv preprint arXiv:2003.06709*, 2020.

[21] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

[22] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[23] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[24] Tsui-Wei Weng, Krishnamurthy Dj Dvijotham, Jonathan Uesato, Kai Xiao, Sven Gowal, Robert Stanforth, and Pushmeet Kohli. Toward evaluating robustness of deep reinforcement learning with continuous control. In *International Conference on Learning Representations*, 2019.

[25] Wei Emma Zhang, Quan Z Sheng, Ahoud Alhazmi, and Chenliang Li. Adversarial attacks on deep-learning models in natural language processing: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(3):1–41, 2020.

Appendix

A Details on how to solve (1) efficiently

Problem (1) can be efficiently solved by proximal-gradient-based methods [15]. Firstly, by substituting a_t and $\hat{s}_{t+1}$ with their definitions, (1) is equivalent to

$$\begin{array}{ll} \min\limits_{x} & d(f(s_t, \pi(s_t + x)), s_{fail}) \\ \text{s.t.} & x \in \mathcal{C}_{p,\varepsilon,t} \end{array} \tag{2}$$

where $\mathcal{C}_{p,\varepsilon,t} := \{x = (x^1, \cdots, x^n) : \ell_{\mathcal{S}} - s_t \leq x \leq u_{\mathcal{S}} - s_t, \left\|x^i\right\|_p \leq \varepsilon \text{ for } i \in \mathcal{V}_t, \text{ and } x^i = \mathbf{0} \text{ for } i \notin \mathcal{V}_t\}$. When f and π are represented by neural network and if we choose the distance function as $d(a, b) = \|a - b\|^2$, (2) is a constrained nonconvex problem, then We can use the projected gradient descent (PGD) algorithm [15] to solve (2). The PGD iteration to update x_k at iteration k starting from x_0 can be described as

$$x_{k+1} = \text{proj}_{\mathcal{C}_{p,\varepsilon,t}} \left[x_k - \eta \nabla_x d(f(s_t, \pi^i(s_t + x)), s_{fail})\right]$$

where $\text{proj}_{\mathcal{C}_{p,\varepsilon,t}}(\cdot)$ is the projection to the convex set $\mathcal{C}_{p,\varepsilon,t}$ and η is the learning rate. The projection is simple to calculate since $\mathcal{C}_{p,\varepsilon,t}$ is the intersection of a ℓ_p-norm ball and boxed constraint.

B Details on Training Dynamics Model

The parameter w for the dynamics model f is the solution of the following optimization problem

$$\min_{\phi} \sum_{t \in \mathcal{D}} \|f(s_t, a_t; \phi) - s_{t+1}\|^2 \tag{3}$$

where $\mathcal{D}$ is a collection of state-action transitions $\{(s_t, a_t, s_{t+1})\}_{t \in \mathcal{D}}$ and s_{t+1} is the actual state that the environment transitions to after taking action a_t determined by a given policy. In particular, we separately collect transitions using the pre-trained policy π_{tr} and a random policy π_{rd} to obtain $\mathcal{D}_{train}$ and $\mathcal{D}_{random}$. The motivation of using the random policy to sample is to avoid overfitting the dynamics model to the trained policy. Then the dataset $\mathcal{D}$ is built as $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{random}$. Since (3) is a standard supervised learning problem, the dynamics model f can be solved by existing gradient-based methods. We describe the whole process of training the dynamics model in Algorithm 2 in Appendix B.

Algorithm 2 Training dynamics model

1: **Initialization:** Given pre-trained policy π_{tr} and a random policy π_{rd}; initialize dynamics model parameter ϕ_0.
2: Form $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{random}$ by collecting a set of transitions $\mathcal{D}_{train}$ and $\mathcal{D}_{random}$ using policy π_{tr} and π_{rd}, respectively.
3: **For** $k = 0, 1, \cdots$ **do**

$$\phi_{k+1} = \texttt{GradientBasedUpdate}(\mathcal{D}, \phi_k)$$

4: **End For**

For each environment, we collect 1 million transitions using the trained MARL policy π_{tr} and a random policy π_{rd}. We partition the collected data into train and test set with a ratio 90-10. The dynamics model is represented by a fully-connected neural network. The network contains 4 hidden layers with 1000 neurons at each layer and the activation function for each hidden layer is ReLU. We train the network for 100 epochs using AdamW [13] with early stopping where the learning rate is tuned in the set $\{0.001, 0.0005, 0.0001, 0.00005, 0.00001\}$ to obtain the model with the best test mean squared error. Please refer to our submitted code for further details.

C More Details on Experiment Setup in Section 4

Experiment setup. We use MADDPG [14] to train MARL agents for the four MA-MuJoCo environments as well as the multi-agent particle environment. Using the trained agents, we collect datasets containing one million transitions to train the dynamics model for each environment. The dynamics model is a fully connected neural network with three hidden layers of 1000 neurons. We use AdamW [13] as the optimizer and select the best learning rate from $\{1, 5\} \times \{10^{-5}, 10^{-4}, 10^{-3}\}$ (the best learning rate is the one achieving lowest prediction error on a test set of $80,000$ samples). For our model-based attack, we run PGD algorithm for $K = 30$ steps to solve (2). We perform each attack over 16 episodes then average the rewards. We also illustrate the standard deviation of rewards using the shaded area in the plots.

To obtain the **Lin et al. (2020) + iFGSM** baseline, we train an adversarial policy for one agent to minimize the total team reward while the remaining agents use the trained MARL policy. This adversarial policy is trained for 1 million timesteps. We then use this trained policy to generate a "target" action and use iterative FGSM method [10, 4] to generate the adversarial observation perturbation for the agents' input. Note that the adversarial policy is trained on the same agent that is being attacked.

Agent partitioning for MA-MuJoCo environments. Each of the original MuJoCo agent in the single-agent setting contains multiple joints and the way these joints are partitioned will lead to different multi-agent configurations. These configurations are described as follows:

- **Walker (2x3)** environment: this environment has 6 joints, 3 for each leg and the whole agent is divided into 2 group of joints $\{1, 2, 3\}$ and $\{4, 5, 6\}$ representing two legs [20, Fig. 4F].
- **HalfCheetah(2x3)** environment: there are two agents, each represents a front or rear leg with joints $\{1, 2, 3\}$ and $\{4, 5, 6\}$ [20, Fig. 4C].
- **HalfCheetah(6x1)** environment: each agent represents each of the total 6 joints [20, Fig. 4D].
- **Ant(4x2)** environment: each agent controls one leg with two joints out of 4 legs [20, Fig. 4J].

Specifying target observation for each environment. To perform our model based attack, we need to specify a target observation that potentially worsens the total reward. Currently, we do not have a general procedure to specify this target observation. We specify the target observations based on prior knowledge about the environments as follows. In multi-agent MuJoCo environments, each agent has access to its own observation of the agent consisting the position-related and velocity-related information. The position-related information includes part of x, y, z coordinates and the quarternion that represents the orientation of the agent. The velocity-related information contains global linear velocities and angular velocities for each joint in a MuJoCo agent. We refer the reader to [23] for more information about each MuJoCo environment. Now we describe the design of this target observation for each environment as follows:

- **Walker(2x3)** environment: Since the episode ends whenever the agent falls, i.e. the z coordinate falls below certain threshold. In this environment, the target observation has a value of 0 for the index that corresponds to the z coordinate of the MuJoCo agent (index 0).
- **HalfCheetah(2x3)** and **HalfCheetah(6x1)** environments: As the goal is to make agent moves as fast as possible, we set the value at index corresponding to the linear velocity to 0 (index 8).
- **Ant(4x2)** environment: As the agent can move freely in a 2D-plan, we set the index corresponding to the x, y linear velocities to 0 (indices 13 and 14).

D Additional Experiments

In this section, we present experimental results in addition to ones presented in Section 4.

Full results for Experiment (I) – model-free baselines vs model-based attack c-MBA on ℓ_∞ perturbation . We first show the full results on running c-MBA and three other baselines in 4

MA-MuJoCo environments under ℓ_∞-norm budget constraint in Fig. 3. c-MBA performs the best in all cases and significantly outperforms other baselines in majority of the cases.

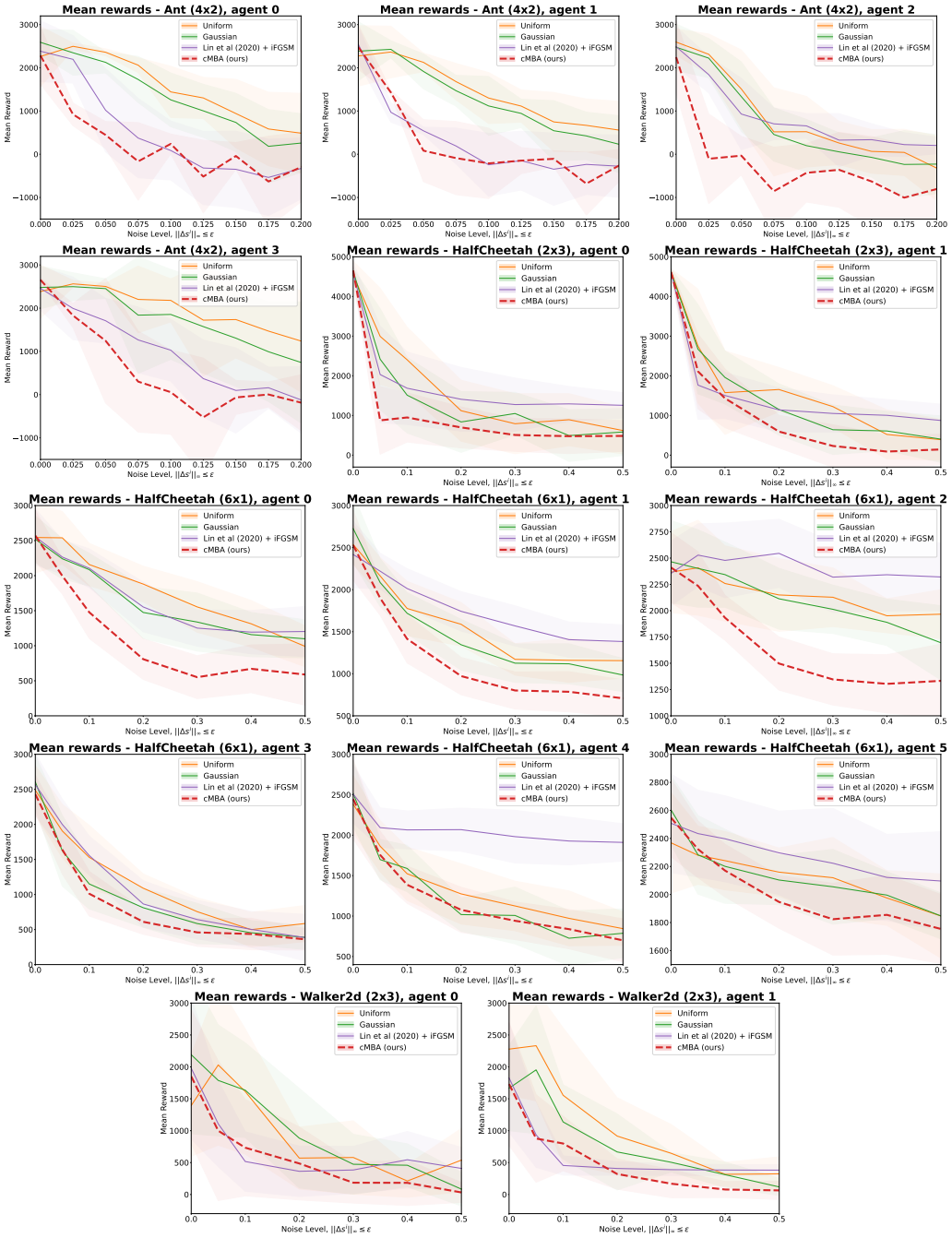

Figure 3: c-MBA vs baselines when attacking one agent in 4 MA-MuJoCo environments - **Exp. (I)**.

Full results of Experiment (II) – attacking two agents using model-free baselines vs model-based attack c-MBA using ℓ_∞ constrained: We conduct experiments using model-free and model-based approaches to simultaneously attack two agents in Ant(4x2) environment. Fig. 4 illustrate the performance of various attacks. We observe that our c-MBA attack outperforms other baselines in 5 out of 6 settings, and especially it's able to achieve low reward (close to or below 0) at lower attack budget levels. For example, at $\varepsilon = 0.025$, the team reward reduction of c-MBA is 68%, 522%, 713% *more* than the three model-free baselines.

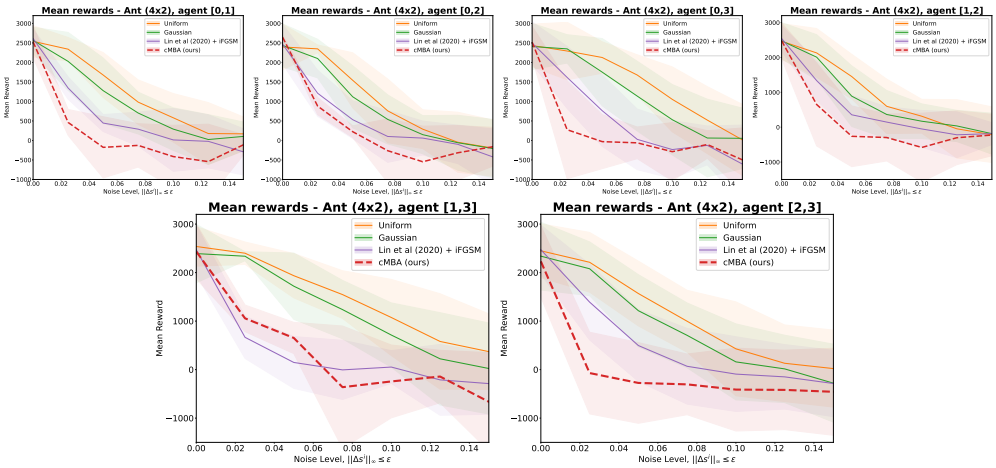

Figure 4: c-MBA vs baselines in Ant(4x2) when attacking two agents - **Exp. (II)**.

Experiment (III): model-free baselines vs model-based attack c-MBA on ℓ_1 perturbation. In addition to the ℓ_∞-norm budget constraint, we also evaluate adversarial attacks using the ℓ_1-norm constraint. Note that using ℓ_1-norm for budget constraint is more challenging as the attack needs to distribute the perturbation across all observations while in the ℓ_∞-norm the computation of perturbation for individual observation is independent. Fig. 5 illustrate the effect of different attacks on **HalfCheetah(6x1)** and **Walker2d(2x3)** environments, respectively. Our c-MBA is able to outperform other approaches in almost all settings. In **HalfCheetah(6x1)**, using $\varepsilon = 1.0$ under ℓ_1 budget constraint, the amount of total team reward reduced by c-MBA variants is up to 156%, 37%, and 42% *more* than Lin et al. (2020) + iFGSM, Gaussian, and Uniform baselines, respectively.

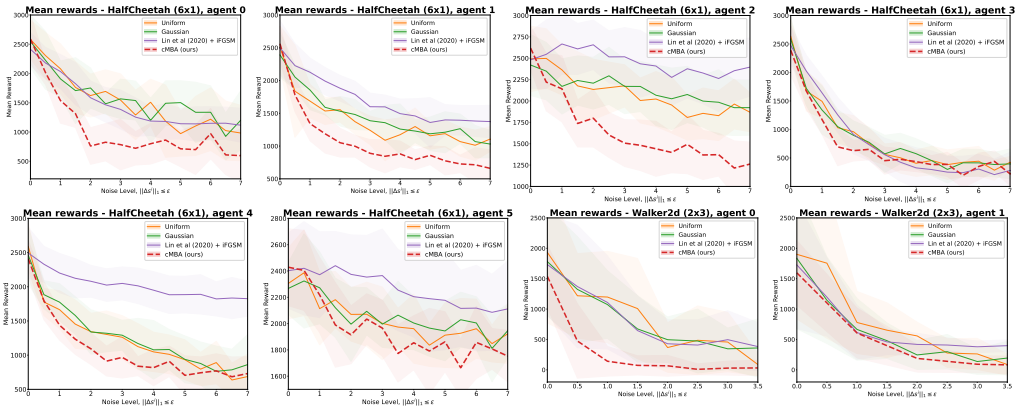

Figure 5: c-MBA vs baselines in **HalfCheetah(6x1)** and **Walker2d(2x3)** - **Exp. (III)**.

In summary, our c-MBA attack is able to shows its advantage in all tested multi-agent environment where it achieves lower reward with smaller budget level.