PIC: Permutation Invariant Critic for Multi-Agent Deep Reinforcement Learning

Iou-Jen Liu* **Raymond A. Yeh*** **Alexander G. Schwing**
University of Illinois at Urbana-Champaign
United States
{iliu3, yeh17, aschwing}@illinois.edu

Abstract: Sample efficiency and scalability to a large number of agents are two important goals for multi-agent reinforcement learning systems. Recent works got us closer to those goals, addressing non-stationarity of the environment from a single agent's perspective by utilizing a deep net critic which depends on all observations and actions. The critic input concatenates agent observations and actions in a user-specified order. However, since deep nets aren't permutation invariant, a permuted input changes the critic output despite the environment remaining identical. To avoid this inefficiency, we propose a 'permutation invariant critic' (PIC), which yields identical output irrespective of the agent permutation. This consistent representation enables our model to scale to 30 times more agents and to achieve improvements of test episode reward between 15% to 50% on the challenging multi-agent particle environment (MPE).

Keywords: Multi-agent Reinforcement Learning, Graph Neural Network, Permutation Invariance

1 Introduction

Single-agent deep reinforcement learning has achieved impressive performance in many domains, including playing Go [1, 2] and Atari games [3, 4]. However, many real world problems, such as traffic congestion reduction [5, 6], antenna tilt control [7], and dynamic resource allocation [8] are more naturally modeled as multi-agent systems. Unfortunately, directly deploying single-agent reinforcement learning to each agent in a multi-agent system does not result in satisfying performance [9, 10].

Particularly, in multi-agent reinforcement learning [8, 10–19], estimating the value function is challenging, because the environment is non-stationary from the perspective of an individual agent [10, 11]. To alleviate the issue, recently, multi-agent deep deterministic policy gradient (MADDPG) [10] proposed a centralized critic whose input is the concatenation of all agents' observations and actions. Similar to MADDPG, Foerster et al. [11, 12], Kim et al. [17], Jiang and Lu [14], Das et al. [15], Iqbal and Sha [13], Yang et al. [20] also deploy centralized critics to handle a non-stationary environment.

However, concatenating all agents' observations and actions assigns an implicit order, *i.e.*, the placement of an agent's observations and actions will make a difference in the predicted outcome. Consider the case of two *homogeneous* agents and let us denote the action and observation of the two agents with 'A' and 'B.' There exists two equally valid permutations (AB) and (BA) which represent the environment. Using a permuted input in classical deep nets will result in a different output. Consequently, referring to the same underlying state of the environment with two different vector representations makes learning of the critic sample-inefficient: the deep net needs to learn that both representations are identical. Due to an increase in the number of possible permutations, this representational inconsistency worsens as the number of agents grows.

To address this concern, we propose the 'permutation invariant critic' (PIC). Due to the permutation invariance property of PICs, the same environment state will result in the same critic output,

*Authors contributed equally.

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

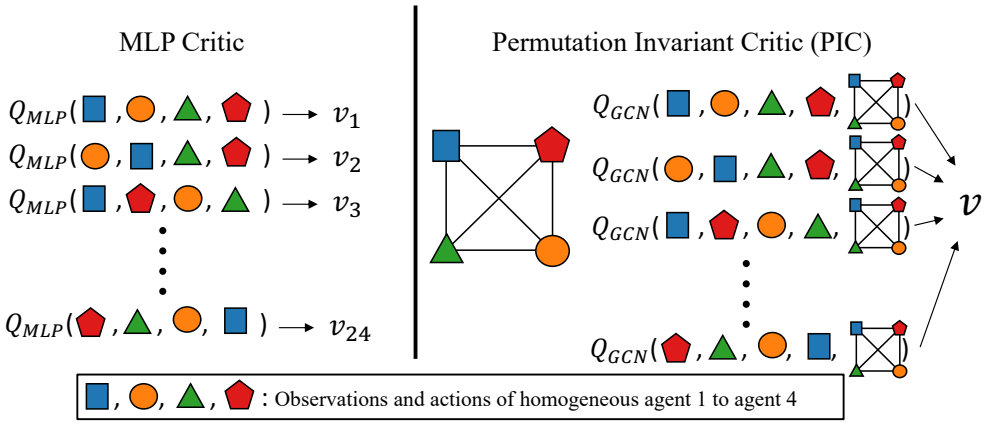

Figure 1: Consider an environment with four homogeneous cooperative agents. Different permutations of the agents' observations and actions refer to the same underlying environment state. However, MLP critics result in 4! different outputs for the same environment state. In contrast, permutation invariant critics yield the same output value for all 4! equivalent permutations.

irrespective of the agent ordering (as shown in Fig. 1). In addition, to tackle environments with homogeneous and heterogeneous agents (*e.g.*, agents which have different action space, observation space, or play different roles in a task), we augment PICs with attributes. This enables the proposed PIC to model the relation between heterogeneous agents.

For rigorous results we follow the strict evaluation protocol proposed by Henderson et al. [21] and Colas et al. [22] when performing experiments in multi-agent particle environments (MPEs) [10, 23]. We found that permutation invariant critics result in 15% to 50% higher average test episode rewards than the MADDPG baseline [10]. Furthermore, we scaled the MPE to 200 agents. Our permutation invariant critic successfully learns the desired policy in environments with a large number of agents, while the baseline MADDPG [10] fails to develop any useful strategies.

In summary, our main contributions are as follows: a) We develop a permutation invariant critic (PIC) for multi-agent reinforcement learning algorithms. Compared with classic MLP critics, the PIC achieves better sample efficiency and scalability. b) To deal with heterogeneous agents we study adding attributes. c) We speedup the multi-agent particle environment (MPE) [10, 23] by a factor of 30. This permits to scale the number of agents to 200, 30 times more than those used in the original MPE environment (6 agents). Code is available at `https://github.com/IouJenLiu/PIC`.

2 Related Work

We briefly review graph neural nets, which are the building block of PICs, and multi-agent deep reinforcement learning algorithms with centralized critics.

Graph Neural Networks. Graph neural networks are deep nets which operate on graph structured data [24]. Input to the network are hence a set of node vectors and connectivity information about the nodes. More notably, these graph networks are permutation equivariant, *i.e.*, the ordering of the nodes in a vector representation does not change the underlying graph [25]. Many variants of graph networks exists, for example, Graph Convolutional Nets (GCN) [26], the Message Passing Network [27], and others [25, 28]. The relation and difference between these approaches are reviewed in [29]. The effectiveness of graph nets has been shown on tasks such as link prediction [30, 31], node classification [32, 26], language and vision [33, 34], graph classification [35–38], *etc*. Graph nets also excel on point sets, *e.g.*, [25, 28]. Most relevant to our multi-agent reinforcement learning setting, graph networks have been shown to be effective in modeling and reasoning about physical systems [39] and multi-agent sports dynamics [40, 41, 35]. Different from these works, here, we study the effectiveness of graph nets for multi-agent reinforcement learning.

Multi-agent Reinforcement Learning. To deal with non-stationary environments form the perspective of a single agent, MADDPG [10] uses a centralized critic that operates on all agents' observations and actions. Similar to MADDPG, Foerster et al. [12] use a centralized critic. In addition, to handle the credit assignment problem [8, 42, 43], a counterfactual baseline has been proposed to marginalize one agent's action and keep the other agents' actions fixed. In "Monotonic Value Function Factorisation" (QMIX) [44] each agent maintains its own value function which conditions only on the agent's local observation. The overall value function is estimated via a non-linear combination of an individual agent's value function. Iqbal and Sha [13] propose an attention mechanism

which enables the centralized critic to select relevant information for each agent. However, as discussed in Sec. 1, the output of centralized critics parameterized by classic deep nets differs if the same environment state is encoded with a permuted vector. This makes learning inefficient.

"Graph convolutional RL" (DGN) [45] is concurrent work on arXiv which uses a nearest-neighbor graph net as the Q-function of a deep Q-network (DQN) [3, 4]. However, the nearest-neighbor graph net only has access to local information of an environment. Consequently, the Q-function in DGN is not a fully centralized critic. Therefore, it suffers from the non-stationary environment issue [10, 11]. In addition, due to the DQN formulation, DGN can only be used in environments with discrete action spaces. Note, DGN considers homogeneous cooperative agents and leaves environments with heterogeneous cooperative agents to future work. In contrast, our permutation invariant critic is fully centralized and can be scaled to a large number of agents. Thanks to different node attributes, our approach can handle environments with heterogeneous cooperative agents. In addition, our approach is designed for continuous state and action spaces.

3 Preliminaries

3.1 Deep Deterministic Policy Gradient

In classic single-agent reinforcement learning, an agent interacts with the environment and collects rewards over time. Formally, at each timestep $t \in \{1, \dots, H\}$, with H the horizon, the agent finds itself in state $s^t \in \mathcal{S}$ and selects an action $a^t \in \mathcal{A}$ according to a deterministic policy μ. Hereby, $\mathcal{S}$ is the state space, and $\mathcal{A}$ is the action space. Upon executing an action, the agent runs into the next state $s^{t+1} \in \mathcal{S}$ and obtains a scalar reward $r^t \in \mathbb{R}$. Given a trajectory $\{s^t, a^t, r^t\}_{t=1}^H$ of length H collected by following a policy, we obtain the discounted return $R = \sum_{t=1}^H \gamma^t r^t$, where $\gamma \in (0, 1]$ is the discount factor. The goal of reinforcement learning is to find a policy which maximizes the return R.

Deep deterministic policy gradient (DDPG) [46] is a widely used deep reinforcement learning algorithm for continuous control. In DDPG, a deterministic policy $\mu(s^t|\theta^\mu)$, which is parameterized by θ^μ, maps a state s^t to an action a^t. A critic $Q(s^t, a^t|\theta^Q)$, which is parameterized by θ^Q, is deployed to estimate the return of taking action a^t at state s^t. The parameters θ^μ of the actor policy are updated iteratively so that the action $a^t = \mu(s^t|\theta^\mu)$ maximizes the critic $Q(s^t, a^t|\theta^Q)$, *i.e.*,

$$\max_{\theta^\mu} J(\theta^\mu) \quad \text{with} \quad J(\theta^\mu) := \mathbb{E}_{s^t \sim \mathcal{D}}[Q(s^t, a^t|\theta^Q)|_{a^t=\mu(s^t|\theta^\mu)}]. \tag{1}$$

Here s^t is drawn from the replay buffer $\mathcal{D}$ which stores experience tuples (s^t, a^t, r^t, s^{t+1}), *i.e.*, $\mathcal{D} = \{(s^t, a^t, r^t, s^{t+1})\}_t$. Using the chain rule, the gradient is

$$\nabla_{\theta^\mu} J(\theta^\mu) = \mathbb{E}_{s^t \sim \mathcal{D}}[\nabla_{\theta^\mu} Q(s^t, a^t|\theta^Q)|_{a^t=\mu(s^t|\theta^\mu)}] = \mathbb{E}_{s^t \sim \mathcal{D}}[\nabla_{\theta^\mu} \mu(s^t|\theta^\mu) \nabla_{a^t} Q(s^t, a^t|\theta^Q)|_{a^t=\mu(s^t|\theta^\mu)}].$$

To optimize w.r.t. θ^Q, similar to deep Q-learning [3, 4], we minimize the loss

$$L(\theta^Q) = \mathbb{E}_{(s^t, a^t, r^t, s^{t+1}) \sim \mathcal{D}}[(Q(s^t, a^t|\theta^Q) - y^t)^2]. \tag{2}$$

The experience tuple (s^t, a^t, r^t, s^{t+1}) is drawn from the replay buffer $\mathcal{D}$ and the target value y^t is defined as

$$y^t = r^t + \gamma Q^-(s^{t+1}, a^{t+1}|\theta^{Q^-})|_{a^{t+1}=\mu(s^{t+1})}, \tag{3}$$

where Q^- is a recent Q-network parameterized by a past θ^{Q^-}.

3.2 Multi-agent Markov Decision Process

To extend single-agent reinforcement learning to the multi-agent setting, we first define the multi-agent Markov decision process (MDP). We consider partially observable multi-agent MDPs [47]. An N-agent partially observable multi-agent MDP is defined by a transition function $\mathcal{T}$, a set of reward functions $\{\mathcal{R}_1, \dots, \mathcal{R}_N\}$, a state space $\mathcal{S}$, a set of observation spaces $\{\mathcal{O}_1 \dots, \mathcal{O}_N\}$, and a set of action spaces $\{\mathcal{A}_1, \dots, \mathcal{A}_N\}$. Action space $\mathcal{A}_i$, observation space $\mathcal{O}_i$ and reward function $\mathcal{R}_i$ correspond to agent $i \in \{1, \dots, N\}$. The transition function $\mathcal{T}$ maps the current state and the actions taken by all the agents to a next state, *i.e.*, $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \dots \times \mathcal{A}_N \to \mathcal{S}$. Each agent receives reward $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_1 \dots \times \mathcal{A}_N \to \mathbb{R}$ and observation o_i that is related to the state, *i.e.*, $o_i : \mathcal{S} \to \mathcal{O}_i$. The goal of agent i is to maximize the expected return $\sum_{t=0}^H \gamma^t r_i^t$. Note that, the goal of N cooperative agents is to maximize the collective expected return $\sum_{t=0}^H \gamma^t (\sum_{j=1}^N r_j^t)$.

3.3 Multi-agent Deep Deterministic Policy Gradient

In this paper, we study multi-agent reinforcement learning using the decentralized execution and centralized training framework [16, 10]. Multi-agent deep deterministic policy gradient (MADDPG) [10] is a well-established algorithm for this framework. Consider N agents with policies $\{\mu_1, \ldots, \mu_N\}$, which are parameterized by $\{\theta_1^\mu, \ldots, \theta_N^\mu\}$. The *centralized* critics $\{Q_1, \ldots, Q_N\}$ associated with the N agents are parameterized by $\{\theta_1^Q, \ldots, \theta_N^Q\}$. Following DDPG, the parameters θ_i^μ for policy μ_i are updated iteratively so that the associated critic Q_i is optimized via

$$\max_{\theta_i^\mu} J(\theta_i^\mu) \quad \text{with } J(\theta_i^\mu) := \mathbb{E}_{(\boldsymbol{x}^t, \boldsymbol{a}^t)\sim\mathcal{D}}[Q_i(\boldsymbol{x}^t, a_1^t, \ldots, a_N^t|\theta_i^Q)|_{a_i^t=\mu_j(o_i^t|\theta_i^\mu)}], \tag{4}$$

where $\boldsymbol{x}^t$ and $\boldsymbol{a}^t$ are the concatenation of all agents' observation and action at timestep t, *i.e.*, $\boldsymbol{x}^t = (o_1^t, \ldots, o_N^t)$ and $\boldsymbol{a}^t = (a_1^t, \ldots, a_N^t)$. Note, o_i^t is the observation received by agent i at time step t. Using the chain rule, the gradient is derived as follows:

$$\nabla_{\theta_i^\mu} J(\theta_i^\mu) = \mathbb{E}_{(\boldsymbol{x}^t, \boldsymbol{a}^t)\sim\mathcal{D}}[\nabla_{\theta_i^\mu}\mu_i(o_i^t|\theta_i^\mu))\nabla_{a_i^t} Q_i(\boldsymbol{x}^t, a_1^t, \ldots, a_N^t|\theta_i^Q)|_{a_i^t=\mu_i(o_i^t|\theta_i^\mu)}]. \tag{5}$$

Following DDPG, the centralized critic parameters θ_i^Q are optimized by minimizing the loss

$$L(\theta_i^Q) = \mathbb{E}_{(\boldsymbol{x}^t, \boldsymbol{a}^t, \boldsymbol{r}^t, \boldsymbol{x}^{t+1})\sim\mathcal{D}}[(Q_i(\boldsymbol{x}^t, \boldsymbol{a}^t|\theta_i^Q) - y_i^t)^2], \tag{6}$$

where $\boldsymbol{r}^t$ is the concatenation of rewards received by all agents at timestep t, *i.e.*, $\boldsymbol{r}^t = (r_1^t, \ldots, r_N^t)$. The target value y_i^t is defined as follows

$$y_i^t = r_i + \gamma Q_i^-(\boldsymbol{x}^{t+1}, a_1^{t+1}, \ldots, a_N^{t+1}|\theta_i^{Q^-})|_{a_j^{t+1}=\mu_j(o_i^{t+1}),\ j=1,\ldots,N}, \tag{7}$$

where Q_i^- is a Q-network parameterized by a past $\theta_i^{Q^-}$.

4 Permutation Invariant Critic (PIC) and Environment Improvements

We first describe the proposed permutation invariant critic (PIC), then show improvements for MPE.

4.1 Permutation Invariant Critic (PIC)

Consider training N homogeneous cooperative agents using a centralized critic Q [8, 10–19]. As discussed in Sec. 3, the input to the centralized critic Q is the concatenation of all agents observations and actions. Let $\boldsymbol{x}^t \in \mathbb{R}^{N\times K_o}$ denote the concatenation of all agents' observations at timestep t, where K_o is the dimension of each observation o_i^t. Similarly, we represent the concatenation of all agents' actions at timestep t via $\boldsymbol{a}^t \in \mathbb{R}^{N\times K_a}$, where K_a is the dimension of each agent's action a_i^t. Note that concatenating observations and actions implicitly imposes an agent ordering. Any agent ordering seems plausible. Importantly, shuffling the agents observations and actions doesn't change the state of the environment. One would therefore expect the centralized critic Q to return the same output if the input is concatenated in a different order. Formally, this property is called *permutation invariance*, *i.e.*, we strive for a critic such that

$$Q(M_i\boldsymbol{x}^t, M_i\boldsymbol{a}^t|\theta^Q) = Q(M_j\boldsymbol{x}^t, M_j\boldsymbol{a}^t|\theta^Q) \quad \forall M_i, M_j \in \mathcal{P},$$

where M_i and M_j are two permutation matrices from the set of all possible permutation matrices $\mathcal{P}$.

To achieve permutation invariance, we propose to use a graph convolutional neural net (GCN) as the centralized critic. In the remainder of the section, we describe the GCN model in detail and discuss how to deploy the permutation invariant critic to environments with homogeneous and heterogeneous agents.

Permutation Invariant Critic. We model an N-agent environment as a graph. Each node represents an agent, and the edges capture the relations between agents. To compute the scalar output of the critic we use L graph convolution layers $\{f_{\text{GCN}}^{(1)}, \ldots, f_{\text{GCN}}^{(L)}\}$. A graph convolution layer takes node representations and the graph's adjacency matrix as input and computes a new representation for each node. More specifically, $f_{\text{GCN}}^{(l)}$ maps the input $\mathbf{h}^{(l-1)} \in \mathbb{R}^{N\times K_{\text{in}}^{(l)}}$ to the output

$\mathbf{h}^{(l)} \in \mathbb{R}^{N \times K_{\text{out}}^{(l)}}$, where $K_{\text{in}}^{(l)}$ and $K_{\text{out}}^{(l)}$ are the input and output node representation's dimension for layer l. Formally,

$$\mathbf{h}^{(l)} = f_{\text{GCN}}^{(l)}(\mathbf{h}^{(l-1)}) := \sigma\left(\frac{1}{N} A_{\text{adj}} \mathbf{h}^{(l-1)} W_{\text{other}}^{(l)} + \text{diag}(1)_N \mathbf{h}^{(l-1)} W_{\text{self}}^{(l)}\right), \tag{8}$$

where A_{adj} is the graph's adjacency matrix, $W_{\text{self}}^{(l)}, W_{\text{other}}^{(l)} \in \mathbb{R}^{K_{\text{in}}^{(l)} \times K_{\text{out}}^{(l)}}$ are the layer's trainable weight matrices, σ is an element-wise non-linear activation function, and $\text{diag}(1)_N$ is an identity matrix of size $N \times N$.

Note that in Equation 8, each agent's representation, *i.e.*, each row of $\mathbf{h}^{(l)}$, is multiplied with the same set of weights W_{other} and W_{self}. Due to this weight sharing scheme, a permutation matrix M applied at the input is equivalent to applying it at the output, *i.e.*,

$$f_{\text{GCN}}^{(l)}(M\mathbf{h}) = M f_{\text{GCN}}^{(l)}(\mathbf{h}). \tag{9}$$

Another advantage of the weight sharing scheme is that the number of trainable parameters of PIC does not increase with the number of agents.

Subsequently, a pooling layer is applied to the L-th graph convolutional layer's representation $\mathbf{h}^{(L)}$. Pooling is performed over the agents' representation, *i.e.*, over the rows of $\mathbf{h}^{(L)}$. We refer to the output of the pooling layer as $\mathbf{v} \in \mathbb{R}^{K_{\text{out}}^{(L)}}$. Either average pooling or max pooling is suitable. Average pooling, subsequently denoted f_{avg}, averages the node representations, *i.e.*, $\mathbf{v} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{h}_i^{(L)}$. Max pooling, subsequently referred to as f_{max}, takes the maximum value across the rows for each of the columns, *i.e.*, $\mathbf{v}_j = \max_i \mathbf{h}_{i,j}^{(L)}\ \forall j \in \{1, \ldots, K_{\text{out}}^{(L)}\}$. Both max pooling and average pooling satisfy the permutation invariance property as summation and element-wise maximization are commutative operations. Therefore, an L-layer graph convolutional net is obtained via $f_{\text{max}} \circ f_{\text{GCN}}^{(L)} \circ \ldots \circ f_{\text{GCN}}^{(1)}$.

Homogeneous setting. If agents in an environment are homogeneous, we first concatenate the observations $\boldsymbol{x}^t$ and actions $\boldsymbol{a}^t$ into a matrix $\boldsymbol{z}^t$, *i.e.*, $\boldsymbol{z}^t := [\boldsymbol{x}^t, \boldsymbol{a}^t] \in \mathbb{R}^{N \times (K_o + K_a)}$. Setting $K_{\text{in}}^{(1)} = K_o + K_a$ and $h^{(0)} = \boldsymbol{z}^t$, we construct a permutation invariant critic Q_{PIC} as follows:

$$Q_{\text{PIC}}(\boldsymbol{z}^t) := f_{\text{v}} \circ f_{\text{max}} \circ f_{\text{GCN}}^{(L)} \circ \ldots \circ f_{\text{GCN}}^{(1)}(\boldsymbol{z}^t). \tag{10}$$

Hereby f_{v} maps the output of the graph nets to a real number, which is the estimated scalar critic value for the environment observation $\boldsymbol{x}^t$ and action $\boldsymbol{a}^t$. We model f_{v} with a standard fully connected layer which maintains permutation invariance. To ensure that the permutation invariant critic is fully centralized, *i.e.*, to ensure that we consider all agents' actions and observations, we use an adjacency matrix corresponding to a complete graph, *i.e.*, A_{adj} is a matrix of all ones with zeros on the diagonal. As later mentioned in Sec. 5 we also study other settings but found a complete graph to yield best results.

Heterogeneous setting. Consider N cooperative agents, which are divided into multiple groups. In this heterogeneous setting, agents in different groups have different characteristics, *e.g.*, size and speed, or play different roles in the task. In a heterogeneous setting, a permutation invariant critic is not directly applicable, because the relation between two heterogeneous agents differs from the relation between two homogeneous agents. For instance, the interaction between two fast-moving agents differs from the interaction between a fast-moving and a slow-moving agent. However, in the aforementioned critic Q_{PIC}, relations between all agents are modeled equivalently.

To address this concern, for the heterogeneous setting, we propose to add node attributes to the PIC. With node attributes, the PIC can distinguish agents from different groups. Specifically, for group j, we construct a group attribute $g_j \in \mathbb{R}^{K_g}$, where K_g denotes the dimension of the group attribute. Let $\boldsymbol{z}_i^t \in \mathbb{R}^{(K_a + K_o)}$ denote the input representation of agent i, *i.e.*, the i-th row of $\boldsymbol{z}^t$. We obtain the augmented representation $\hat{\boldsymbol{z}}_i^t$ via $\hat{\boldsymbol{z}}_i^t := [\boldsymbol{z}_i^t, g_{G(i)}]$, where $G(i)$ denotes the group index of agent i. We perform the augmentation for each agent to obtain the augmented representation $\hat{\boldsymbol{z}}^t \in \mathbb{R}^{N \times (K_a + K_o + K_g)}$. Using Equation 10 and setting $K_{\text{in}}^{(1)} = K_a + K_o + K_g$ and $\mathbf{h}^{(0)} = \hat{\boldsymbol{z}}^t$, results in a PIC that can handle environments with heterogeneous agents.

4.2 Improved MPE Environment

The multiple particle environment (MPE) [10, 23] is a multi-agent environment which contains a variety of tasks and provides a challenging open source platform for our community to evaluate new

		MLP Critic		MLP Critic + Data Augaumentation		Ours Permutation Invariant Critic	
	# of agents	final	absolute	final	absolute	final	absolute
Cooperative Navigation	N=3	-362.73	-362.71	-361.76	-361.55	**-355.99**	**-355.74**
	N=6	-3943.2	-3933.3	-4025.4	-4016.2	**-3383.2**	**-3381.8**
	N=15	-6489.7	-6394.3	-6280.0	-6222.4	**-1999.1**	**-1977.6**
	N=30	-20722	-20583	-21205	-20840	**-11363**	**-11294**
	N=100	-128100	-128086	-128024	-128013	**-71495**	**-71074**
	N=200	-502349	-502348	-509963	-507457	**-436215**	**-433846**
Prey & Predator	N=3	38.52	40.91	43.78	44.99	**65.16**	**67.69**
	N=6	26.15	30.34	-24.84	-17.49	**176.70**	**184.22**
	N=15	3982.80	4416.23	4198.41	4401.64	**10139**	**10239**
	N=30	-377.19	-386.49	-93.86	-75.59	**6662.1**	**6745.6**
	N=100	19894	21114	28741	29347	**99391**	**100812**
Cooperative Push	N=3	-171.26	-170.20	-171.96	-171.74	**-155.17**	**-155.10**
	N=6	-561.73	-542.29	-672.80	-672.27	**-401.79**	**-395.70**
	N=15	-2538.3	-2536.0	-2645.2	-2610.1	**-2231.1**	**-2225.7**
	N=30	-3499.7	-3465.9	-3761.5	-3688.3	**-3117.1**	**-3094.3**
Heterogeneous Navigation	N=4	-100.74	-100.35	-98.96	-98.67	**-83.84**	**-83.54**
	N=8	-398.31	-397.66	-683.24	-684.39	**-398.31**	**-397.66**
	N=16	-3410.0	-3405.9	-3479.0	-3470.8	**-1825.8**	**-1820.4**
	N=30	-12944	-12934	-12812	-12809	**-6293**	**-6270**
	N=100	-121563	-121472	-130028	-129436	**-94324**	**-93996**

Table 1: Average rewards of our approach and two MADDPG baselines. 'final' represents final metric [22], which is the average reward over the last 10,000 evaluation episodes, *i.e.*, 1000 episodes for each of the last ten policies during training. 'absolute' represents absolute metric [22], which is the best policy's average reward over 1,000 evaluation episodes.

algorithms. However, the MPE targets only settings with a small number of agents. Specifically, we found it challenging to train more than 30 agents as it takes more than 100 hours. To scale the MPE to more agents, we improve the implementation of the MPE. More specifically, we develop vectorized versions for many of the computations, such as computing the force between agents, computing the collision between agents, *etc*. Moreover, for tasks with global rewards, instead of computing rewards for each agent, we only compute the reward once and send it to all agents. With this improved MPE, we can train up to 200 agents within one day.

5 Experiments

In this section, we first introduce tasks in the multiple particle environment (MPE) [10, 23]. We then present the details of the experimental setup, evaluation protocol, and our results.

Environment. We evaluate the proposed approach on an improved version of MPE [10, 23]. We consider the following four tasks:

- *Cooperative navigation*: N agents move cooperatively to cover L landmarks in the environment. The reward encourages the agents to get close to landmarks.
- *Prey and predator*: N slower predators work together to chase M fast-moving preys. The predators get positive reward when colliding with preys. Preys are environment controlled.
- *Cooperative push*: N agents work together to push a large ball to a landmark. The agents get rewarded when the large ball approaches the landmark.
- *Heterogeneous navigation*: $N/2$ small and fast agents and $N/2$ big and slow agents work cooperatively to cover N landmarks. The reward encourages the agents to get close to landmarks. If a small agent collides with a big agent, a large negative reward is received.

Note *cooperative navigation*, *prey and predator* and *cooperative push* are environments with homogeneous agents. In each task, the reward is global for cooperative agents, *i.e.*, cooperative agents always receive the same reward in each timestep.

Experimental Setup. We use MADDPG [10] with a classic MLP critic as the baseline. We implement MADDPG and the proposed approach in Pytorch [48]. To ensure the correctness of our implementation, we compare it with the official MADDPG code [49] on MPE. Our implementation reproduces the results of the official code. Please see Tab. A1 in the supplementary for the results.

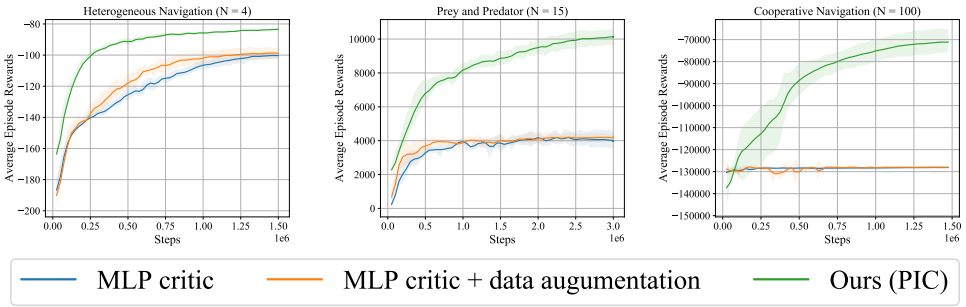

Figure 2: Comparison of average episode reward.

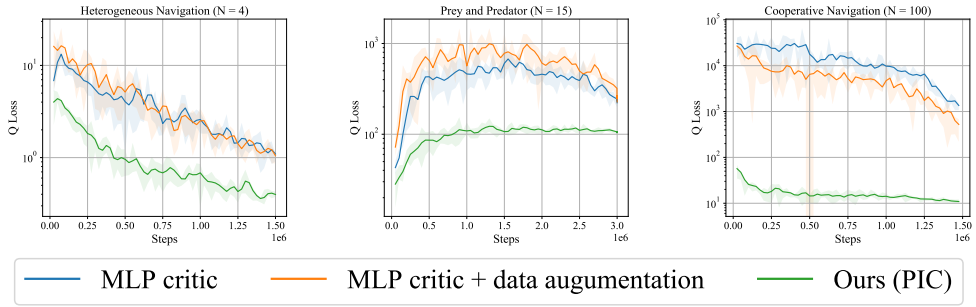

Figure 3: Comparison of average Q loss (Equation 6).

Following MADDPG [10], the actor policy is parameterized by a two-layer MLP with 128 hidden units per layer, and ReLU activation function. The MLP critic of the MADDPG baseline has the same architecture as the actor. Our permutation invariant critic (PIC) is a two-layer graph convolution net (GCN) with 128 hidden units per layer and a max pooling at the top. The activation function for GCN is also a ReLU. Following MADDPG, the Adam optimizer is used. The learning rates for the actor, MLP critic, and our permutation invariant critic are 0.01. The learning rate is linearly decreased to zero at the end of training. Agents are trained for $60,000$ episodes in all tasks (episode length is either 25 or 50 steps). The size of the replay buffer is one million, the batch size is 1024, and the discounted factor $\gamma = 0.95$. In an eight-agent environment, the MLP critic and the PIC have around 44k and 40k trainable parameters respectively. In an 100-agent environment, the MLP critic has 413k trainable parameters while the PIC has only 46k parameters.

Evaluation Protocol. To ensure a fair and rigorous evaluation, we follow the strict evaluation protocols suggested by Colas et al. [22] and Henderson et al. [21]. For each experiment, we report *final metrics* and *absolute metrics*. The final metric [22] is the average reward over the last $10,000$ evaluation episodes, *i.e.*, $1,000$ episodes for each of the last ten policies during training. The absolute metric [22] is the best policy's average reward over $1,000$ evaluation episodes. To analyze the significance of the reported improvement over the baseline, we perform a two-sample t-test and boostrapped estimation of the 95% confidence interval for the mean reward difference obtained by the baseline and our approach. We use the Scipy [50] implementation for the t-test, and the Facebook Boostrapped implementation with $10,000$ boostrap samples for confidence interval estimation. All experiments are repeated for five runs with different random seeds.

Results. We compare the proposed permutation invariant critic (PIC) with an MLP critic and an MLP critic with data augmentation. Data augmentation shuffles the order of agents' observations and actions when training the MLP critic, which is considered to be a simple way to alleviate the ordering issue. The results are summarized in Tab. 1, where N denotes the number of agents, 'final' and 'absolute' are the final metric and absolute metric respectively. We observe that data augmentation does not boost the performance of an MLP critic much. In some cases, it even deteriorates the results. In contrast, as shown in Tab. 1, a permutation invariant critic outperforms the MLP critic and the MLP critic with data augmentation by about $15\% - 50\%$ in all tasks.

The training curves are given in Fig. 2 and Fig. 3. As shown in Fig. 3, the loss (Equation 6) of the permutation invariant critic is lower than that of MLP critics. Please see the supplementary material for more training curves. Fig. 4 (top) shows the ratio of the MLP critic's average loss to the permutation invariant critic's loss on cooperative navigation environments. We observed that the ratio grows when the number of agents increases. This implies that a permutation invariant approach

		t-test	Boostrap C.I.
		t-stat. (p-value)	mean (C.I.)
N=3	abs.	1.42 (1.9e-1)	5.81 (-1.47, 13.09)
	final	1.41 (1.9e-1)	5.76 (-1.65, 13.18)
N=6	abs.	9.46 (3.2e-5)	634.4 (487.9, 757.1)
	final	9.34 (4.1e-5)	642.2 (496.1, 762.3)
N=15	abs.	21.04 (2.5e-5)	4244 (3819, 4575)
	final	20.6 (2.8e-5)	4280 (3854, 4616)
N=30	abs.	3.17 (2.2e-2)	9546 (6719, 12639)
	final	3.10 (2.1e-2)	9841 (6692, 13201)
N=100	abs.	17.8 (2.9e-3)	56939 (51842, 62764)
	final	17.3 (3.0e-3)	56529 (51153, 62343)
N=200	abs.	2.72 (4.9e-2)	73611 (25930, 116198)
	final	2.66 (4.8e-2)	73747 (26557, 116457)

Table 2: T-test and boostrap confidence interval of mean difference between MLP critic and our permutation invariant critic on cooperative navigation.

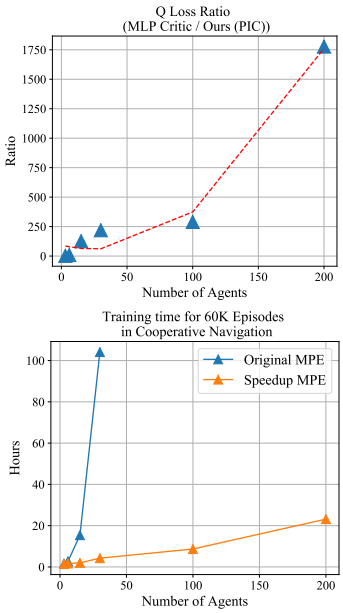

Figure 4: **Top**: Average critic loss ratio (MLP / Ours). **Bottom:** Training time comparison.

gives much more accurate value estimation than the MLP critic particularly when the number of agents is large.

To confirm that the performance gain of our permutation invariant critic is significant, we report the 2-sample t-test results and the 95% boostrapped confidence interval on the mean difference of our approach and the baseline MADDPG with MLP critic. For the 2-sample t-test, the t-statistic and the p-value are reported. The difference is considered significant when the p-value is smaller than 0.05. Tab. 2 summarizes the analysis on cooperative navigation environments. As shown in Tab. 2, the p-value is much smaller than 0.05 which suggests the improvement is significant. In addition, positive confidence interval and means suggest that our approach achieves higher rewards than the baseline. Please see the supplementary material for additional analysis.

In addition to fully connected graphs we also tested K-nearest neighbor graphs, *i.e.*, each node is connected only to its K nearest neighbors. The distance between two nodes is the physical distance between the corresponding agents in the environment. We found that using fully connected graphs achieves better results than using K-nearest neighbor graphs. We report absolute rewards for ($K = 2$, $K = 6$, fully connected graph): the results on cooperative navigation ($N = 15$), prey and predator ($N = 15$), cooperative push ($N = 15$), and heterogeneous navigation ($N = 16$) are $(-2162, -2059, -1977)$, $(3014, 3163, 10239)$, $(-2370, -2270, -2225)$, and $(-2635, -2552, -1820)$ respectively.

Fig. 4 (bottom) compares training time of MADDPG on the original MPE and our improved MPE. MADDPG is trained for $60,000$ episodes with a PIC. As shown in Fig. 4 (bottom), training 30 agents in the original MPE environment takes more than 100 hours. In contrast, with the improved MPE environment, we can train 30 agents within five hours, and scale to 200 agents within a day of training.

6 Conclusion

We propose and study permutation invariant critics to estimate a consistent value for all permutations of the agents' order in multi-agent reinforcement learning. Empirically, we demonstrate that a permutation invariant critic outperforms classical deep nets on a variety of multi-agent environments, both homogeneous and heterogeneous. Permutation invariant critics lead to better sample complexity and permit to scale learning to environments with a large number of agents. We think that the permutation invariance property is important for deep multi-agent reinforcement learning. Going forward we will study other designs to scale to an even larger number of agents.

Acknowledgements: This work is supported in part by NSF under Grant #1718221 and MRI #1725729, UIUC, Samsung, 3M, Cisco Systems Inc. (Gift Award CG 1377144), Adobe, and a Google PhD Fellowship to RY. We thank NVIDIA for providing GPUs used for this work and Cisco for access to the Arcetri cluster.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. G. nad John Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 2017.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *NeurIPS Deep Learning Workshop*, 2013.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[5] A. L. C. Bazzan. Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. In *AAMAS*, 2008.

[6] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *AAMAS*, 2018.

[7] N. Dandanov, H. Al-Shatri, A. Klein, and V. Poulkov. Dynamic self-optimization of the antenna tilt for best trade-off between coverage and capacity in mobile networks. *Wireless Personal Communications*, 2017.

[8] D. T. Nguyen, A. Kumar, and H. C. Lau. Credit assignment for collective multiagent rl with global rewards. In *NeurIPS*, 2018.

[9] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *ICML*, 1993.

[10] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NeurIPS*, 2017.

[11] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *ICML*, 2017.

[12] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. In *AAAI*, 2018.

[13] S. Iqbal and F. Sha. Actor-attention-critic for multi-agent reinforcement learning. In *ICML*, 2019.

[14] J. Jiang and Z. Lu. Learning attentional communication for multi-agent cooperation. In *NeurIPS*, 2018.

[15] A. Das, T. Gervet, J. Romoff, D. Batra, D. Parikh, M. Rabbat, and J. Pineau. Tarmac: Targeted multi-agent communication. In *ICML*, 2019.

[16] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *ICML*, 2016.

[17] D. Kim, S. Moon, D. Hostallero, W. J. Kang, T. Lee, K. Son, and Y. Yi. Learning to schedule communication in multi-agent reinforcement learning. In *ICLR*, 2019.

[18] T. Shu and Y. Tian. M3rl: Mind-aware multi-agent management reinforcement learning. In *ICLR*, 2019.

[19] L. Han, P. Sun, Y. Du, J. Xiong, Q. Wang, X. Sun, H. Liu, and T. Zhang. Grid-wise control for multi-agent reinforcement learning in video game ai. In *ICML*, 2019.

[20] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang. Mean field multi-agent reinforcement learning. In *ICML*, 2018.

[21] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *AAAI*, 2017.

[22] C. Colas, O. Sigaud, and P.-Y. Oudeyer. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. In *ICML*, 2018.

[23] I. Mordatch and P. Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv.*, 2017.

[24] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009.

[25] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *NeurIPS*, 2017.

[26] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[27] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[28] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017.

[29] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv.*, 2018.

[30] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *ESWC*, 2018.

[31] M. Zhang and Y. Chen. Link prediction based on graph neural networks. In *NeurIPS*, 2018.

[32] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[33] M. Narasimhan, S. Lazebnik, and A. G. Schwing. Out of the Box: Reasoning with Graph Convolution Nets for Factual Visual Question Answering. In *Proc. NIPS*, 2018.

[34] I. Schwartz, S. Yu, T. Hazan, and A. G. Schwing. Factor Graph Attention. In *Proc. CVPR*, 2019.

[35] R. A. Yeh, A. G. Schwing, J. Huang, and K. Murphy. Diverse generation for multi-agent sports games. In *CVPR*, 2019.

[36] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*, 2015.

[37] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

[38] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[39] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. In *NeurIPS*, 2016.

[40] Y. Hoshen. Vain: Attentional multi-agent predictive modeling. In *NeurIPS*, 2017.

[41] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems. In *ICML*, 2018.

[42] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. In *AAMAS*, 2005.

[43] Y. han Chang, T. Ho, and L. P. Kaelbling. All learning is local: Multi-agent learning in global reward games. In *NeurIPS*, 2003.

[44] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *ICML*, 2018.

[45] J. Jiang, C. Dun, and Z. Lu. Graph convolutional reinforcement learning for multi-agent cooperation. In *arxiv.*, 2019.

[46] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[47] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *ICML*, 1994.

[48] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NeurIPS*, 2017.

[49] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent deep deterministic policy gradient, 2018. URL https://github.com/openai/maddpg.

[50] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001. URL http://www.scipy.org/.

Supplementary material: PIC: Permutation Invariant Critic for Multi-agent Deep Reinforcement Learning

A MADDPG Baseline

We implement the MADDPG baseline in Pytorch. To ensure we implement MADDPG correctly, we compare the performance of our implementation with MADDPG's official code on the MPE. As shown in Tab. A1, our implementation reproduces the results of MADDPG in all environments.

B Environment Details and Group Embedding

In this section, we first provide details of observation and action space in each environment we considered in our experiments. Subsequently, we discuss a PIC's group embedding.

In all the four environments, the action dimension is five. One dimension is no-op. The other four dimensions represent the left, right, forward, and backward force applied to the particle agent. An agent's observation always contains its location and velocity. Depending on the environment the observation may contain relative location and velocity of neighboring agents and landmarks. In our experiments, the number of visible neighbors in an agent's observation is equal to or less than ten because we empirically found a larger number of visible neighbors to not boost the performance of MADDPG but rather slow the training speed. Note, the number of visible neighbors in an agent's observation is different from the K-nearest neighbor graph discussed in Sec. 5. The K in the K-nearest neighbor graph refers to the number of agent observations and actions which are used as input to the centralized critic, while the number of visible neighbors in an agent's observation is a characteristic of an environment.

The details of the observation representation for each environment are as follows:

- *Cooperative navigation*: An agent observes its location and velocity, and the relative location of the nearest k landmarks and agents. k is 2, 5, 5, 5, 5, 5 for an environment with 3, 6, 15, 30, 100, 200 agents. As a result, the observation dimension is 14, 26, 26, 26, 26, 26 for an environment with 3, 6, 15, 30, 100, 200 agents.
- *Prey and predator*: A predator observes its location and velocity, the relative location of the nearest l landmarks and fellow predators, the relative location and velocity of the k nearest preys. (k, l) is (2, 1), (3, 2), (5, 5), (5, 5) (5, 5) for an environment with 3, 6, 15, 30, 100 agents. As a result, the observation dimension is 16, 28, 34, 34, 34 for an environment with 3, 6, 15, 30, 100 agents.
- *Cooperative push*: An agent observes its location and velocity, the relative location of the target landmark and the large ball, and the relative location of the k nearest agents. k is 2, 5, 10, 10 for an environment with 3, 6, 15, 30 agents. As a result, the observation dimension is 12, 18, 28, 28 for an environment with 3, 6, 15, 30 agents.
- *Heterogeneous navigation*: An agent observes its location and velocity, and the relative location of the nearest k landmarks and agents. k is 2, 5, 5, 5, 5 for an environment with 3, 6, 15, 30, 100 agents. Consequently, the observation dimension is 14, 26, 26, 26, 26 for an environment with 3, 6, 15, 30, 100 agents.

In the heterogeneous environment *Heterogeneous navigation*, the number of groups is two, *i.e.*, two groups of agents that have different characteristics. For the PIC, the group embedding for each group is a two-dimensional vector, *i.e.*, $g_j \in \mathbb{R}^2$ for each group $j \in \{1, 2\}$ in the environment. We train

Table A1: Comparison of the official MADDPG code [49] and our MADDPG implementation on MPE.

	Cooperative Navigation	Prey and Predator		Push	
	good	adversary	good	adversary	good
MADDPG official code	-379.57	2.91	5.48	-7.05	-1.67
Our implementation	-367.75	5.97	14.79	-6.54	-1.81

Table A2: Ours (PIC) *vs*. MLP critic in the cooperative navigation environment.

		t-test	Boostrap C.I.
		t-stat. (p-value)	mean (C.I.)
N=3	abs.	1.42 (1.9e-1)	5.81 (-1.47, 13.09)
	final	1.41 (1.9e-1)	5.76 (-1.65, 13.18)
N=6	abs.	9.46 (3.2e-5)	634.4 (487.9, 757.1)
	final	9.34 (4.1e-5)	642.2 (496.1, 762.3)
N=15	abs.	21.04 (2.5e-5)	4244 (3819, 4575)
	final	20.6 (2.8e-5)	4280 (3854, 4616)
N=30	abs.	3.17 (2.2e-2)	9546 (6719, 12639)
	final	3.10 (2.1e-2)	9841 (6692, 13201)
N=100	abs.	17.8 (2.9e-3)	56939 (51842, 62764)
	final	17.3 (3.0e-3)	56529 (51153, 62343)
N=200	abs.	2.72 (4.9e-2)	73611 (25930, 116198)
	final	2.66 (4.8e-2)	73747 (26557, 116457)

Table A3: Ours (PIC) *vs*. MLP critic in the prey and predator environment.

		t-test	Boostrap C.I.
		t-stat. (p-value)	mean (C.I.)
N=3	abs.	6.61 (1.6e-4)	22.7 (15.2, 29.7)
	final	5.38 (8.0e-4)	21.3 (13.6, 29.6)
N=6	abs.	3.14 (3.4e-2)	201 (88.2, 316)
	final	3.11 (3.5e-2)	201 (86.4, 318)
N=15	abs.	20.34 (1.7e-6)	5838 (5468, 6248)
	final	20.29 (5.7e-7)	5941 (5609, 6303)
N=30	abs.	5.91 (4.0e-3)	6821 (5176, 9006)
	final	5.82 (4.3e-3)	6755 (5106, 8925)
N=100	abs.	6.92 (7.3e-2)	92753 (82974, 102532)
	final	7.44 (7.1e-2)	89980 (80796, 99163)

Table A4: Ours (PIC) *vs*. MLP critic in the cooperative push environment.

		t-test	Boostrap C.I.
		t-stat. (p-value)	mean (C.I.)
N=3	abs.	4.03 (1.4e-2)	16.6 (10.3, 24.4)
	final	4.04 (1.4e-2)	16.7 (10.4, 24.6)
N=6	abs.	66.4 (1.0e-10)	276.5 (269, 287)
	final	67.2 (1.3e-11)	271.0 (263, 281)
N=15	abs.	4.90 (6.3e-3)	384 (250, 507)
	final	5.12 (5.2e-3)	414 (280, 546)
N=30	abs.	4.41 (7.1e-3)	593 (400, 778.316)
	final	4.64 (5.4e-3)	644 (419, 844)

Table A5: Ours (PIC) *vs*. MLP critic in the heterogeneous navigation environment.

		t-test	Boostrap C.I.
		t-stat. (p-value)	mean (C.I.)
N=3	abs.	7.07 (1.7e-3)	15.1 (11.8, 18.4)
	final	6.76 (2.2e-3)	15.1 (11.6, 18.6)
N=6	abs.	19.7 (2.1e-5)	285 (259, 317)
	final	19.6 (2.2e-5)	286 (260, 318)
N=15	abs.	62.2 (3.0e-9)	1650 (1619, 1678)
	final	61.9 (6.5e-9)	1653 (1619, 1684)
N=30	abs.	74.1 (3.7e-9)	6538 (6380, 6725)
	final	76.0 (2.6e-9)	6518 (6362, 6695)
N=100	abs.	5.82 (1.9e-2)	36327 (27272, 50292)
	final	5.65 (1.8e-2)	36618 (26339, 51579)

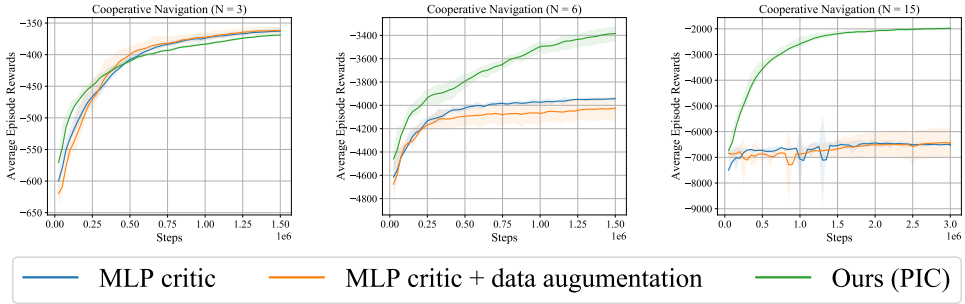

Figure A1: Average episode reward comparison. Our permutation invariant critic (PIC) outperforms the MLP critic in all environment settings.

the embedding along with the network parameters. The group embedding is randomly initialized from a normal distribution $\mathcal{N}(0, \text{diag}(1)_2)$.

C T-test and Training Curves

The two-sample t-test and the 95% confidence interval of the mean difference of the MLP critic and the PIC are summarized in Tab. A2, Tab. A3, Tab. A4, and Tab. A5. p-values smaller than 0.05 and positive confidence intervals indicated that PIC's improvement over the MLP critic is significant. The training curves for MADDPG with MLP critic and our PIC are shown in Fig. A1 and Fig. A2. The PIC outperforms the MLP critic in all the environment settings.

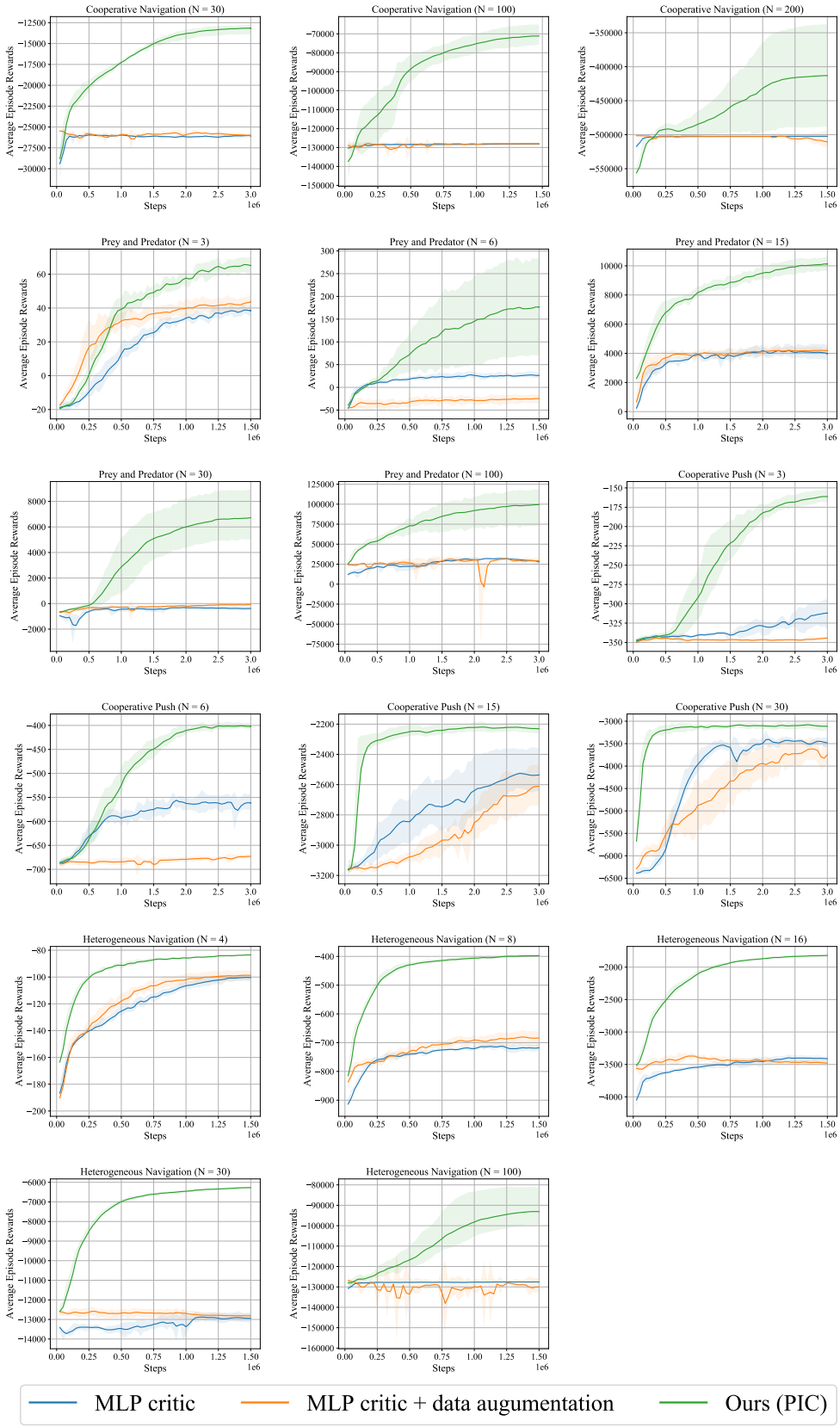

Figure A2: Average episode reward comparison. Our permutation invariant critic (PIC) outperforms the MLP critic in all environment settings.