Multi-Agent Policy Transfer via Task Relationship Modeling

Rongjun Qin[*1 2] **Feng Chen**[*1] **Tonghan Wang**[*3] **Lei Yuan**[1 2] **Xiaoran Wu**[4] **Zongzhang Zhang**[1]
Chongjie Zhang[3] **Yang Yu**[1 2]

Abstract

Team adaptation to new cooperative tasks is a hallmark of human intelligence, which has yet to be fully realized in learning agents. Previous work on multi-agent transfer learning accommodate teams of different sizes, heavily relying on the generalization ability of neural networks for adapting to unseen tasks. We believe that the relationship among tasks provides the key information for policy adaptation. In this paper, we try to discover and exploit common structures among tasks for more efficient transfer, and propose to learn effect-based task representations as a common space of tasks, using an alternatively fixed training scheme. We demonstrate that the task representation can capture the relationship among tasks, and can generalize to unseen tasks. As a result, the proposed method can help transfer learned cooperation knowledge to new tasks after training on a few source tasks. We also find that fine-tuning the transferred policies help solve tasks that are hard to learn from scratch.

1. Introduction

Cooperation in human groups is characterized by resiliency to unexpected changes and purposeful adaptation to new tasks. This flexibility and transferability of cooperation is a hallmark of human intelligence. Computationally, multi-agent reinforcement learning (Tan, 1993) provides an important means for machines to imitate human cooperation. Although recent multi-agent reinforcement learning research has made prominent progress in many aspects of cooperation, such as policy decentralization (Lowe et al., 2017; Rashid et al., 2018; Wang et al., 2021a;c; Cao et al., 2021), communication (Foerster et al., 2016; Jiang & Lu, 2018; Wang et al., 2020b), and organization (Wang et al., 2020a; 2021b; Jiang et al., 2019), how to realize the ability of group knowledge transfer is still an open question.

Compared to single-agent knowledge reuse (Zhu et al., 2020), a unique challenge faced by multi-agent transfer learning is the varying size of agent groups. The number of agents and the length of observation inputs in unseen tasks may differ from those in source tasks. To solve this problem, existing multi-agent transfer learning approaches build population-invariant (Long et al., 2019) and input-length-invariant (Wang et al., 2020d) learning structures using graph neural networks (Agarwal et al., 2020) and attentional mechanics like transformers (Hu et al., 2021; Zhou et al., 2021). Although these methods handle varying populations and input lengths well, knowledge transfer to unseen tasks mainly depends on the inherent generalization ability of neural networks. The relationship among tasks is not fully used for more efficient transfer.

To make up for this shortage, we study the discovery and utilization of common structures in multi-agent tasks and propose Multi-Agent Transfer reinforcement learning via modelling TAsk Relationship (MATTAR). In this learning framework, we capture the common structure of tasks by modeling the similarity among transition and reward functions of different tasks. Specifically, we train a forward model for all source tasks to predict the observation, state, and reward at the next timestep given the current observation, state, and actions. The question is how to embody the similarity as well as the difference among tasks in this forward model. We introduce difference by giving each source task a unique representation and model the similarity by generating the parameters of the forward model via a shared hypernetwork, which we call the representation explainer.

To learn a well-formed representation space that encodes task relationship, an alternative-fixed training method is proposed to learn the task representation and representation explainer. During training, representations of source tasks are pre-defined and fixed as mutual orthogonal vectors, and the representation explainer is learned. When facing an unseen task, we fix the representation explainer and backpropagate gradients through the fixed forward model to learn

[*] Equal contribution [1] National Key Laboratory of Novel Software Technology, Nanjing, China [2] Polixir Technologies, Nanjing, China [3] IIIS, Tsinghua University, Beijing, China [4] Department of Computer Science and Technology, Tsinghua University, Beijing, China. Correspondence to: Chongjie Zhang <chongjie@tsinghua.edu.cn>, Zongzhang Zhang <zzzhang@nju.edu.cn>.

the representation of the new task by a few samples.

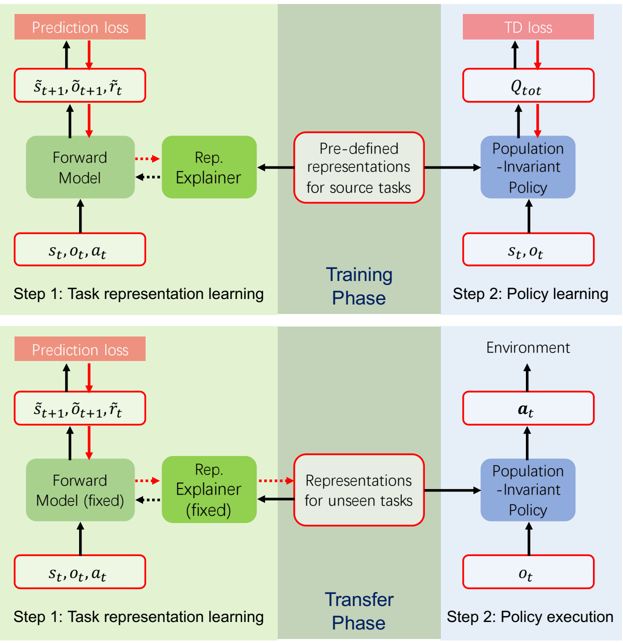
Furthermore, we condition the agent policies on the task representation. During training, the task representation is fixed, and the policy is updated to maximize the expected return. On unseen tasks, we obtained the transferred policy by simply inserting the new task representation. The structure of the policy is also designed to be adaptable to population and observation inputs of different sizes.

We design experiments to demonstrate that the learned knowledge from three to four source tasks can be transferred to a series of unseen tasks with great success rates. We also pinpoint several advantages brought by our method other than knowledge transfer. First, fine-tuning the transferred policy on unseen tasks achieves better performance than learning from scratch, indicating that the task representation provides a good initialization point. Second, training on multiple source tasks gets better performance compared to training on them individually. This result shows that MATTAR also provides a method for multi-agent multi-task learning. Finally, although not designed for this goal, our structure enables better learning performance against single-task learning algorithms when trained on single tasks.

2. Method

In this paper, we focus on knowledge transfer among fully cooperative multi-agent tasks that can be modelled as a Dec-POMDP (Oliehoek et al., 2016) consisting of a tuple G=$\langle I, S, A, P, R, \Omega, O, n, \gamma\rangle$, where I is the finite set of n agents, $s \in S$ is the true state of the environment, and $\gamma \in [0, 1)$ is the discount factor. At each timestep, each agent i receives an observation $o_i \in \Omega$ drawn according to the observation function $O(s, i)$ and selects an action $a_i \in A$. Individual actions form a joint action $\boldsymbol{a} \in A^n$, which leads to a next state s' according to the transition function $P(s'|s, \boldsymbol{a})$, a reward $r = R(s, \boldsymbol{a})$ shared by all agents. Each agent has local action-observation history $\tau_i \in \mathrm{T} \equiv (\Omega \times A)^* \times \Omega$. Agents learn to collectively maximize the global action value function $Q_{tot}(s, \boldsymbol{a}) = \mathbb{E}_{s_{0:\infty},a_{0:\infty}}[\sum_{t=0}^{\infty} \gamma^t R(s_t, \boldsymbol{a}_t)|s_0 = s, \boldsymbol{a}_0 = \boldsymbol{a}]$.

Overall, our framework first trains on several source tasks $\mathcal{S} = \{\mathcal{S}_i\}$ and then transfers the learned cooperative knowledge to unseen tasks $\mathcal{T} = \{\mathcal{T}_j\}$ from the same task distribution. As shown in Fig. 1, our learning framework achieves this by designing two modules for task representation learning and task policy learning, respectively. In the following sections, we first introduce how we design the representation learning module and its learning scheme in different phases. Then, we describe the details of policy learning, including the population-invariant structure for dealing with inputs of different sizes.



Prediction loss
$\tilde{s}_{t+1}, \tilde{o}_{t+1}, \tilde{r}_t$
Forward Model
Rep. Explainer
s_t, o_t, a_t
Step 1: Task representation learning
Pre-defined representations for source tasks
Training Phase
TD loss
Q_{tot}
Population -Invariant Policy
s_t, o_t
Step 2: Policy learning
Prediction loss
$\tilde{s}_{t+1}, \tilde{o}_{t+1}, \tilde{r}_t$
Forward Model (fixed)
Rep. Explainer (fixed)
s_t, o_t, a_t
Step 1: Task representation learning
Representations for unseen tasks
Transfer Phase
Environment
$\boldsymbol{a}_t$
Population -Invariant Policy
o_t
Step 2: Policy execution

Figure 1. Transfer learning scheme of our method.

2.1. Task representation learning

Our main idea in achieving knowledge transfer among multi-agent tasks is to capture and exploit both the common structure and the unique characteristics of tasks by learning task representation. A task distinguishes itself from other tasks by its transition and reward functions. Therefore, we incorporate our task representation learning component into the learning of a forward model that predicts the next state, local observations, and reward given the current state, local observations, and actions.

We associate each task i with a representation $z_i \in \mathbb{R}^m$ and expect it to reflect different properties of tasks. For modeling task similarity, all source and unseen tasks share a representation explainer, which takes as input the task representations and outputs the parameters of the forward model. The representation explainer is trained on all source tasks. Concretely, for a source task $\mathcal{S}_i$, parameters of the forward model are generated as $\eta_i = f_\theta(z_i)$, where θ denotes the parameters of the representation explainer. The forward model contains three predictors: $f^s_{\eta_i}$, $f^o_{\eta_i}$, and $f^r_{\eta_i}$. Given the current state s, agent j's observation o_j, and action a_j, these predictors estimate the next state s', the next observation o'_j, and the global reward r, respectively.

A possible method for training on source tasks is to backpropagate the forward model's prediction error to update both the representation explainer and task representations. However, this training scheme leads to unexpected results in practice, which is mainly attributable to the fact that the network may ignore the task representation, and the representations may have a very small norm.

To solve this problem, we propose pre-determining the task representation for each source task and learning the representation explainer by backpropagating the prediction error. Such a method can help form an informative task representation space and build a mapping from task representation space to the space of forwarding model parameters. The question is how we pre-define these source task representations. In practice, we initialize source task representations as mutually orthogonal vectors. Specifically, we first randomly generate vectors in $\mathbb{R}^m$ for source tasks, and then use the Schimidt orthogonalization (Björck, 1994) on these vectors to obtain source task representations.

With pre-defined task representations, the representation explainer is optimized to minimize the following loss function:

$$J(\theta) := \sum_{i}^{N_{src}} J_{\mathcal{S}_i}(\theta), \tag{1}$$

where N_{src} is the number of source tasks, and

$$\begin{aligned} J_{\mathcal{S}_i}(\theta) = \mathbb{E}_{\mathcal{D}}\big[\|f^s_{\eta_i}(s, o_i, a_i) - s'\|^2 \\ + \lambda_1 \sum_j \|f^o_{\eta_i}(s, o_i, a_i) - o'_i\|^2 + \lambda_2 (f^r_{\eta_i}(s, o_i, a_i) - r)^2\big] \end{aligned} \tag{2}$$

is the per-task prediction loss of the forward model. Here, $\mathcal{D}$ is the replay buffer, and λ_1, λ_2 are scaling factors.

We fix the source task representations and learn the representation explainer during the training phase. When it comes to the transfer phase, we aim to find a good task representation that can reflect the similarity of the new task to source tasks. To achieve this goal, we fix the trained representation explainer and learn the task representation by minimizing the prediction loss of the forward model on the new task. Specifically, we randomly initialize a task representation z, keep θ fixed, and get the forward model parameterized by $\eta = f_\theta(z)$. The same as the case of training the representation explainer, we compute the prediction loss for both transition and reward functions. Still, the backpropagated gradient this time is not used to update θ, but to update the task representation z.

To keep the new task representation in the well-formed space of source task representations, we learn new task representation as a linear combination of source task representations:

$$z = \sum_{i=1}^{N_{src}} \mu_i z_i \;\; \text{s.t.} \;\; \mu_i \geq 0, \sum_{i=1}^{N_{src}} \mu_i = 1. \tag{3}$$

In this way, what we are learning is the weight vector μ. To make the learning more stable, we additionally optimize an entropy regularization term $\mathcal{H}(\mu)$. The final loss function for learning z is:

$$\begin{aligned} J_{\mathcal{T}}(\mu) = \lambda\mathcal{H}(\mu) + \mathbb{E}_{\mathcal{D}}\big[\|f^s_{\eta}(s, o_i, a_i) - s'\|^2 \\ + \lambda_1 \sum_j \|f^o_{\eta}(s, o_i, a_i) - o'_i\|^2 + \lambda_2 (f^r_{\eta}(s, o_i, a_i) - r)^2\big], \end{aligned} \tag{4}$$

where $\eta = f_\theta(z)$ and $z = \sum_{i=1}^{N_{src}} \mu_i z_i$.

The detailed architectures for task representation learning and forward model are described in Appendix A.4.

2.2. Task policy learning

After the task representation is learned by modeling the transition and reward functions, it will be used to learn and transfer the policy on the source and unseen tasks.

Another difficulty faced by transferable multi-agent policy learning is that the dimension of state and observation varies across tasks with the number of agents. Thus, it is infeasible for some popular MARL algorithms like QMIX (Rashid et al., 2018) and MADDPG (Lowe et al., 2017) to be directly applied to transfer knowledge by reloading the policy model since their network input sizes are fixed.

To solve this problem, we propose a **population-invariant network** structure (Fig. 2), which can deal with the varying input sizes. This network structure is also designed to be conditioned on the learned task representations.

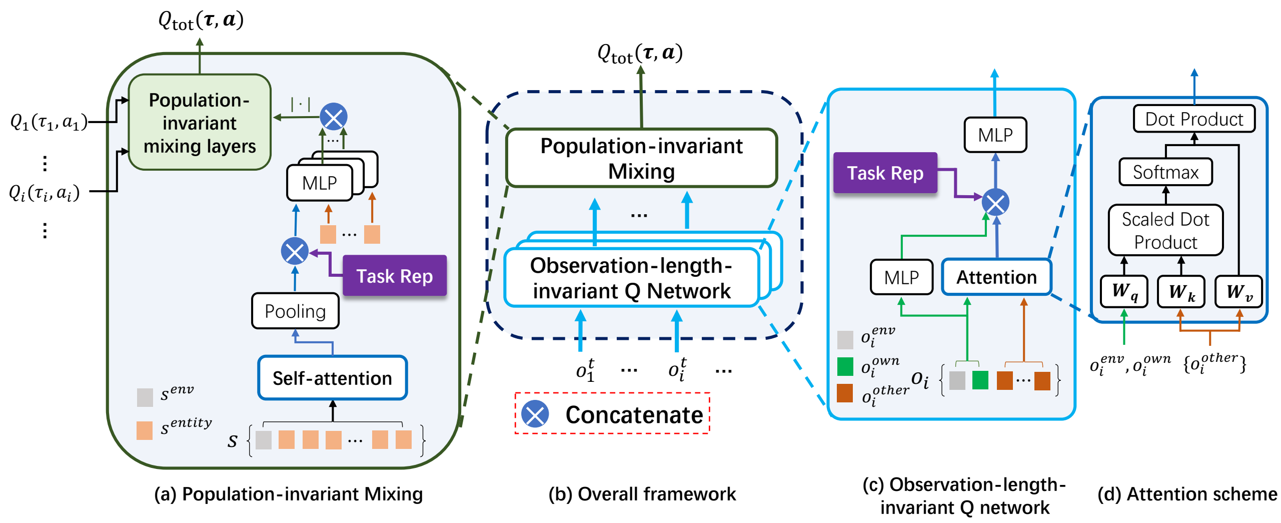
The population-invariant network uses the value decomposition learning framework and mainly consists of two components. Agents share an individual Q-network, and the global Q-function is learned as a combination of local Q-values. In this work, we adopt a monotonic mixing network in the style of QMIX (Rashid et al., 2018), but our method is readily applied to other value decomposition methods.

In most multi-agent problems, state and observation consist of different parts corresponding to the environment information and information about other entities. We exploit this property to design the individual Q-network and the mixing network to enable them to deal with tasks with a different number of agents.

For individual Q-network, we disassemble the observation o_i into the part relating to the environment o_i^{env}, the part relating to agent i itself o_i^{own}, and the parts relating to other entities $\{o_j^{other}\}$. We adopt an attention mechanism where the query is generated by o_i^{env} and o_i^{own}. This scheme learns to which entities should we pay attention:

$$\begin{aligned} q &= \mathtt{MLP}_q([o_i^{env}, o_i^{own}]), \\ \boldsymbol{K} &= \mathtt{MLP}_K([o_1^{other}, \ldots, o_j^{other}, \ldots]), \\ \boldsymbol{V} &= \mathtt{MLP}_V([o_1^{other}, \ldots, o_j^{other}, \ldots]), \\ h &= \mathtt{softmax}(\frac{q\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V}, \end{aligned} \tag{5}$$

where $[\cdot, \cdot]$ is the vector concatenation operation, d_k is the dimension of the query vector, and bold symbols are matrices.



(a) Population-invariant Mixing

(b) Overall framework

(c) Observation-length-invariant Q network

(d) Attention scheme

Figure 2. Population-invariant network structure for policy learning.

This attention mechanism helps get a fixed-dimension embedding h for inputs with varying sizes. We concatenate h with an embedding vector of $[o_i^{env}||e_i^{own}]$ and obtain a final fixed-dimension hidden vector. This vector is then fed into the subsequent network together with the task representation of the action value estimates.

In some multi-agent tasks, there exist interaction actions that involve other opponents. As a result, the number of actions also varies in different tasks, presenting a challenge for knowledge transfer because conventional Q-networks have outputs with a fixed length. To handle this problem, we design a new mechanism inspired by other popular population-invariant networks (Wang et al., 2020c; Hu et al., 2021), where we calculate Q-values for non-interaction actions and interaction actions separately. For non-interaction actions, we calculate the Q-values by directly applying a conventional deep Q-network for that the number of non-interaction actions is usually fixed. For interaction actions, we utilize a sharing network that takes as input the observation part relating to the corresponding entity, together with the concatenation of h and task representation z. This sharing network outputs the Q-value estimation for the corresponding interaction action. We empirically compare MATTAR against ASN (Wang et al., 2020c) in our experiments, and the detailed description of our structure for handling varying numbers of actions can be found in Appendix A.6.

For the mixing network, we disassemble the state s into parts relating to different entities in the environment $\{s_j^P\}$. We again apply a self-attention mechanism to integrate information from these parts of the state:

$$\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} = \texttt{MLP}_{Q,K,V}([s_1^P, \dots, s_j^P, \dots]), \quad (6)$$

$$\boldsymbol{H} = \texttt{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V}, \quad (7)$$

$$h^{state} = \texttt{pooling}([h_1, \dots, h_j, \dots]), \ h_j = \boldsymbol{H}_j. \quad (8)$$

The pooling operation in the last step guarantees a fixed-dimension embedding vector. We use this embedding, together with the task representation, to generate parameters for the mixing network.

During the whole process of policy learning, we fix the task representation z. Compared to policy learning, which typically lasts for 2M timesteps, the training of task representation costs few samples. In practice, we collect 50K samples for learning task representations and the representation explainer.

When **transferring to new tasks**, we use the individual Q-network and the representation explainer trained on source tasks. We learn the task representation for 50K timesteps and insert it into the individual Q-network. Agents execute in a decentralized manner according to this Q-network. In section 4, we evaluate the performance of this scheme.

3. Related Work

3.1. Multi-agent Transfer Learning

Transfer learning holds the promise in improving the sample efficiency of reinforcement learning (Zhu et al., 2020) and multi-agent reinforcement learning (Silva & Costa, 2021). The basic idea behind multi-agent transfer learning is reusing knowledge from other tasks or other learning agents, which corresponds to inter-agent transfer and intra-agent transfer, respectively. It is expected that this knowledge reuse can accelerate coordination compared to learning from scratch.

The intra-agent transfer paradigm aims at investigating how to best reuse knowledge from other agents (Yang et al.,

2021) with different sensors or (possibly) internal representations via communication. DVM (Wadhwania et al., 2019) treats the multi-agent problem as a multi-task problem to combine knowledge from multiple tasks, and then distills this knowledge by a value matching mechanism. LeCTR (Omidshafiei et al., 2019) learns to teach in a multi-agent environment. Yang et al. (2021) takes a further step by proposing an option-based policy transfer for multi-agent cooperation.

On the other hand, inter-agent transfer refers to reusing knowledge from previous tasks, focusing on transferring knowledge across multi-agent tasks. The varying populations and input lengths impede the transfer among agents, with which the graph neural networks (Wang et al., 2020d) and transformer mechanism (Zhou et al., 2021) play promising roles. DyMA-CL (Wang et al., 2020d) designs various transfer mechanisms across curricula to accelerate the learning process based on a dynamic agent-number network. EPC (Long et al., 2019) proposes a curriculum learning paradigm via an evolutionary approach to scale up the population number of agents. UPDeT (Hu et al., 2021) and PIT (Zhou et al., 2021) make full use of the generalization ability of the transformer to accomplish the multi-agent cooperation and transfer between tasks. Although these methods can accurately the learning efficiency for multi-agent reinforcement learning somehow, they neglect task representation in multi-agent tasks, resulting in low transfer efficiency in complex scenarios.

3.2. Multi-agent Representation Learning

Learning effective representation in MARL is receiving significant attention for its effectiveness in solving many important problem. CQ-Learning (Hauwere et al., 2010) learns to adapt the state representation for multi-agent systems to coordinate with other agents. Grover et al. (2018) learns useful policy representations to model agents behavior in a multi-agent system. LILI (Xie et al., 2020) learns latent representations to capture the relationship between its behavior and the other agent's future strategy and uses it to influence the other agent. RODE (Wang et al., 2021b) uses an action encoder to learn action representations and applies clustering methods to decompose joint action spaces into restricted role action spaces to reduce the policy search space for multi-agent cooperation. MAR (Zhang et al., 2021) learns meta representation for multi-agent generalization. Unlike previous work, our approach focuses on learning meaningful task representation for efficient transfer learning in multi-agent reinforcement learning.

4. Experiments

In this section, we design experiments to evaluate the following properties of the proposed method. (1) Generalizability to unseen tasks. Can our learning framework extract knowledge from multiple source tasks and transfer the cooperation knowledge to unseen tasks? Do task representations play an indispensable role in transfer (see Sec. 4.1)? (2) A good initialization for policy fine-tuning. Fine-tuning the transferred policy can help us succeed in super hard tasks, which can not be solved effectively when learning from scratch (see Sec. 4.2). (3) Benefits of multi-task training. Our multi-task learning paradigm helps the model better leverage knowledge of different source tasks to boost the learning performance compared to training on source tasks individually (see Sec. 4.3). (4) Performance advantages on single tasks. Although we did not design our framework for better performance on single-task training. We find our network performs better against the underlying algorithms (see Sec. 4.4).

We evaluate MATTAR on the benchmark of SMAC (Samvelyan et al., 2019) based on PyMARL[1]. To better fit the multi-task training setting, we extend the original SMAC maps and sort out three task series. The first series consists of tasks with varying numbers of Marines. The second series involves ally and enemy teams of Stalkers and Zealots. The third series contains tasks with teams consisting of Marines, Maneuvers, and Medivacs. The detailed description of these tasks is described in Appendix A.1. To test the transferability of MATTAR, we divide the tasks in each series into source tasks and unseen tasks.

To ensure fair evaluation, we carry out all the experiments with five random seeds, and the results are shown with a 95% confidence interval. For a more comprehensive description of experimental details, please refer to Appendix A.3.

4.1. Generalizability to unseen tasks

As the major desiderata of our method, we expect that MATTAR has better transfer performance on unseen tasks. We compare our method against the state-of-the-art multi-agent transfer method UPDeT (Hu et al., 2021) which is based on the transformer.

UPDeT transfers knowledge from a single source task. For a fair comparison, we transfer from each source task to every unseen task and calculate the best (UPDeT-b) and mean (UPDeT-m) performance on each unseen task. For the test phase, we conduct transfer experiments on both source tasks and unseen tasks. We carry out experiments on all the three series of tasks, and record results in Tables 1∼3.

We find that UPDeT-b outperforms UPDeT-m in all cases, indicating that the similarity between source task and target

[1] Our experiments are all based on the PyMARL framework, which uses SC2.4.6.2.6923. Note that performance is not always comparable among versions.

Table 1. Transfer performance on a series of SMAC maps involving Stalkers and Zealots.

	MATTAR	w/o task rep.	UPDeT-b	UPDeT-m
		Source Tasks		
2s3z	**1.0**	0.15	0.97	0.67
3s5z	**1.0**	0.14	0.91	0.52
35_36	**0.53**	0.0	0.24	0.08
		Unseen Tasks		
23_24	**0.02**	0.0	0.0	0.0
3s4z	**0.99**	0.2	0.98	0.57
4s7z	**0.73**	0.01	0.29	0.14
47_48	**0.14**	0.0	0.0	0.0

Note: 35_36 is short for 3s5z_vs_3s6z, 23_24 is short for 2s3z_vs_2s4z, and 47_48 is short for 4s7z_vs_4s8z.

Table 2. Transfer performance on a series of SMAC maps involving Marines, Maneuvers, and Medivacs.

	MATTAR	w/o task rep.	UPDeT-b	UPDeT-m
		Source Tasks		
MMM	**1.0**	0.36	**1.0**	0.96
MMM2	**0.92**	0.14	0.78	0.39
MMM4	**0.93**	0.07	0.76	0.36
		Unseen Tasks		
MMM0	**1.0**	0.39	0.43	0.4
MMM1	**0.99**	0.04	0.77	0.51
MMM3	**0.86**	0.01	0.7	0.35
MMM5	**0.24**	0.0	0.0	0.0
MMM6	**0.02**	0.0	0.0	0.0

task significantly influences the transfer performance, and a good source task selection is important for this kind of transfer algorithm. In contrast, our algorithm obtains a general model without requiring much prior knowledge about task relationships. Moreover, MATTAR shows better transfer performance on unseen tasks than UPDeT-b, especially in complex settings requiring sophisticated coordination like the MMM series.

To investigate the role of task representations in our method, we ablate this component by inserting a zero vector of the same dimension as our task representation into the policy network. We can see that this ablation (`w/o task rep.`) dramatically underperforms MATTAR on most unseen tasks. For each example, after trained on `2s3z`, `3s5z`, and `3s5z_vs_3s6z`, MATTAR achieves a win rate of 0.99 on `3s4z` but a win rate of 0.20 without the help of task representation. We thus conclude that **task representations play an indispensable role** in policy transfer.

4.2. A good initialization for policy fine-tuning

When evaluating the performance of our method on unseen tasks, we only train the task representations but remain other parts of our framework unchanged. In this section, we investigate the performance of MATTAR after fine-tuning.

Specifically, we train the task representations for an unseen task for 50K timesteps, then we fix it and train the policy network for 2M timesteps. In Fig. 3, we compare this performance against learning from scratch on three unseen tasks from three different task series.

We observe that the task representation provides a good initialization. For example, on `10m_vs_12m`, after 2M training samples, MATTAR with task representation converges to the win rate of around 0.86, while training solely on this task can only achieve a win rate of about 0.4. Furthermore, with the help of task representation, MATTAR solves `3s5z_vs_3s7z`, a task harder than the super hard `3s5z_vs_3s6z`, which cannot be solved when learning from scratch.

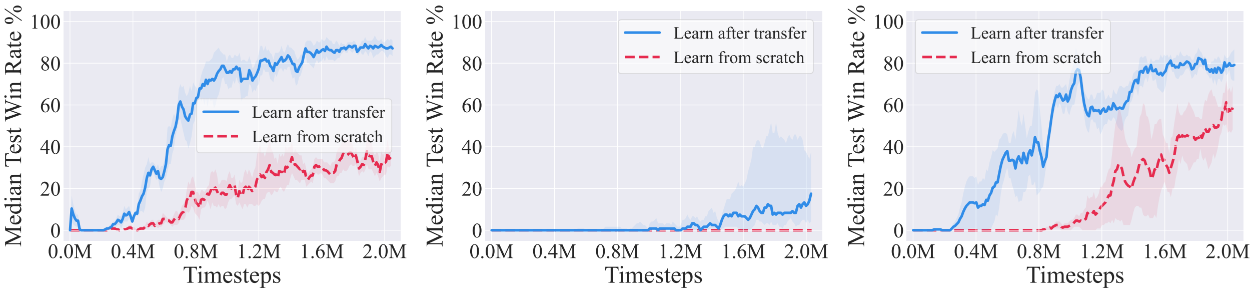
On other tasks where learning from scratch can obtain a satisfactory win rate, inserting the task representation can also improve the sample efficiency. For example, on `MMM6`, after experiencing 0.8M training samples, MATTAR wins in around 35% of the games, compared to the zero win rate obtained by learning from scratch.

4.3. Benefits of multi-task learning

During training, MATTAR adopts a scheme where multiple sources are learned simultaneously. Our aim is to leverage knowledge from more tasks and to be able to generalize the learned knowledge to a larger set of unseen tasks. Empirically, we find that this multi-task training setting helps not only unseen tasks but also the source tasks themselves.

In Fig. 4, we present the performance of MATTAR on source tasks when training with multiple tasks and a single task. The experiments are carried out three tasks from three different series. We can see that training on multiple tasks significantly boosts learning performance. For example, on `3s5z_vs_3s6z`, after 2M training samples, MATTAR with multiple tasks converges to the win rate of around 0.6, while training solely on this task can only achieve a win rate of about 0.05.

These results demonstrate that *MATTAR also provides a good learning framework for multi-agent multi-task learning*. It can leverage experience on other tasks to improve performance on a similar task.

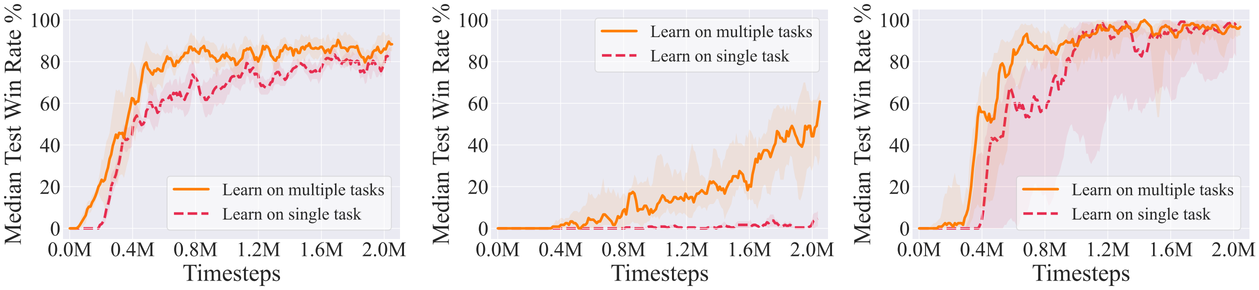


(a) 10m_vs_12m

(b) 3s5z_vs_3s7z

(c) MMM6

Figure 3. On unseen tasks: task representations provide a good initialization. Fine-tuning the policy can effectively learn cooperation policies on tasks which cannot be solved effectively when learning from scratch.



(a) 5m_vs_6m

(b) 3s5z_vs_3s6z

(c) MMM2

Figure 4. On source tasks: MATTAR provides a framework for multi-agent multi-task learning. Training on multiple tasks helps improve performance than learning on a single task.

4.4. Bonus: performance on single-task training

Although not designed for this goal, we find that MATTAR can outperform state-of-the-art MARL algorithms when trained on some single tasks. Specifically, we use random task representations and train MATTAR from scratch. We compare our method with two state-of-the-art value-based MARL baselines (QMIX (Rashid et al., 2018) and QPLEX (Wang et al., 2021a)), a role-based learning algorithm (RODE) (Wang et al., 2021b), and the underlying algorithm of MATTAR which considers the Q values of interaction actions separately (ASN) (Wang et al., 2020c).

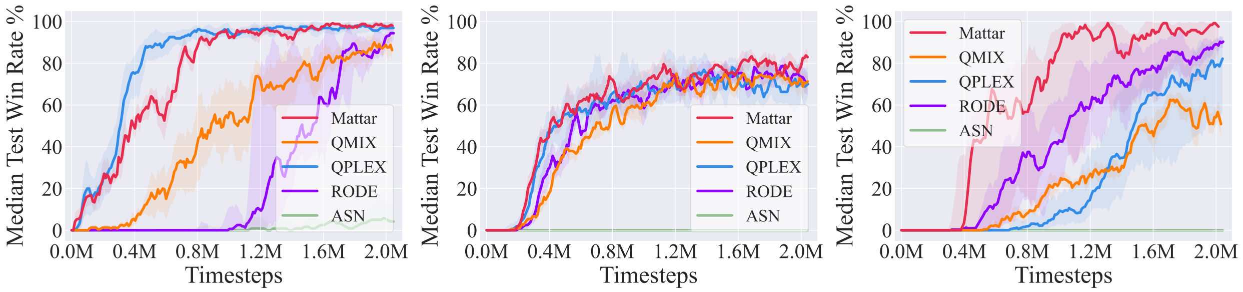
Figure 5 shows the learning curves of different methods. We find that our population-invariant network structure achieves comparable performance in all tasks. It is worth noting that this structure even significantly outperforms other algorithms on the super hard map `MMM2`.

In Appendix A.5, we show the performance of MATTAR on more SMAC maps. It can be observed there that MATTAR also has comparable performance against baseline algorithms on these maps. Given that our underlying algorithm is QMIX, this is an inspiring result. We hypothesize that this result is because our self-attention scheme increases the representational capacity by learning to attend to appropriate entities in the environment.

4.5. Learned linear combination of the representations for unseen tasks

When encountering an unseen task, we first learn its representation as a linear combination of the representations for all source tasks. Specifically, we directly update the coefficients of this linear combination by backpropagating the prediction error of the forward model. For a deeper understanding of how our method transfers the learned knowledge, we are curious about the learned coefficients of the linear combination because they contain much information about the relationship between source and unseen tasks.

For each series of tasks, we show the coefficients of two unseen tasks in Table 4. We observe that the largest coefficient typically corresponds to the source task, which is the most similar to the unseen task. In the first series, `5m` is the closest source task to the unseen task `4m`, and the coefficient of `5m` takes up 61% of all the coefficients. In the second series, `3s5z_vs_3s6z` is the closest source task to the unseen task `3s5z_vs_3s7z`. Correspondingly, the coefficient of `3s5z_vs_3s6z` also takes up 61% of all the coefficients,



(a) 3s5z (b) 5m_vs_6m (c) MMM2

Figure 5. On single tasks: when learning from scratch on single tasks, MATTAR exhibits superior performance. For performance on the whole SMAC benchmark, please refer to Appendix A.5.

Table 3. Transfer performance on a series of SMAC maps involving Marines.

	MATTAR	w/o task rep.	UPDeT-b	UPDeT-m
		Source Tasks		
5m	**1.0**	0.94	**1.0**	0.82
5m_6m	0.69	0.08	**0.93**	0.24
8m_9m	**0.96**	0.67	0.84	0.4
10m_11m	0.81	0.62	**0.92**	0.48
		Unseen Tasks		
3m	**0.94**	0.72	0.42	0.17
4m	**0.99**	0.94	0.97	0.57
4m_5m	**0.02**	0.0	0.0	0.0
6m	**1.0**	**1.0**	**1.0**	0.98
6m_7m	0.76	0.4	**0.82**	0.33
7m	**1.0**	**1.0**	**1.0**	0.97
7m_8m	**0.83**	0.78	0.7	0.36
8m	**1.0**	**1.0**	**1.0**	0.85
9m	**1.0**	0.99	**1.0**	0.8
9m_10m	0.84	0.77	**0.92**	0.46
10m	**1.0**	**1.0**	**1.0**	0.72
10m_12m	**0.08**	0.01	0.07	0.02

Note: x_y is short for x_vs_y, e.g. 5m_6m is short for 5m_vs_6m.

Table 4. Task representations for unseen tasks are learned as a linear combination of source tasks' representations. This table shows the learned coefficients of the linear combination. *Unseen tasks mainly leverage knowledge from the most similar source task.*

Unseen	Source Tasks			
	5m	5m_6m	8m_9m	10m_11m
4m	**0.61**	0.13	0.14	0.12
10m_12m	**0.43**	0.07	0.08	**0.43**
	2s3z	3s5z	3s5z_3s6z	
3s4z	0.21	**0.59**	0.21	
3s5z_3s7z	0.18	0.21	**0.61**	
	MMM	MMM2	MMM4	
MMM0	**0.44**	0.15	0.40	
MMM6	0.30	0.14	**0.56**	

indicating an important role of this source task. A similar phenomenon can be observed in the case of the third series and in MMM/MMM0 and MMM4/MMM6.

There are also some exceptions. For example, in the unseen task 10m_vs_12m, the coefficients of two source tasks, 5m and 10m_vs_11m, are equal, and they together take up 86% of all the coefficients. While 10m_vs_11m is very similar to 10m_vs_12m, the policy for solving 5m is very different from that for 10m_vs_12m. However, in 10m_vs_12m, agents usually first form some groups to set up an attack curve quickly. Therefore, on a local battlefield, there are around five allies fighting against a similar number of enemies. In this case, the policy learned from 5m can be used locally. We hypothesize that MATTAR can learn a mixing of source task policies to solve a new task.

We conclude that, in our learning framework, unseen tasks can effectively leverage cooperation knowledge from the most similar source tasks and occasionally use knowledge from mixing of source tasks.

5. Closing Remarks

In this paper, we study the problem of cooperative multi-agent transfer reinforcement learning. Previous work on multi-agent transfer mainly deals with the varying population and input lengths, relying on the generalization ability of neural networks for cooperation knowledge transfer, ignoring the task relationship. Our method improves the transfer performance by learning task representations that capture the difference and similarities among tasks. When facing a new task, our method only needs to obtain a new representation before transferring the learned knowledge to it. Taking advantage of task relationship mining, MATTAR achieves the best transfer performance and exhibits some other advantages of this algorithm.

An important direction in the future is the transfer among tasks from different task distributions. This paper does not investigate whether the learned cooperation policies can be transferred to very dissimilar tasks. Another interesting problem is whether a linear combination of source tasks' representations can fully represent unseen tasks.

References

Agarwal, A., Kumar, S., Sycara, K., and Lewis, M. Learning transferable cooperative behavior in multi-agent teams. In *International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1741–1743, 2020.

Björck, Å. Numerics of gram-schmidt orthogonalization. *Linear Algebra and Its Applications*, 197:297–316, 1994.

Cao, J., Yuan, L., Wang, J., Zhang, S., Zhang, C., Yu, Y., and Zhan, D.-C. Linda: Multi-agent local information decomposition for awareness of teammates. *arXiv preprint arXiv:2109.12508*, 2021.

Foerster, J., Assael, I. A., de Freitas, N., and Whiteson, S. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2137–2145, 2016.

Grover, A., Al-Shedivat, M., Gupta, J. K., Burda, Y., and Edwards, H. Learning policy representations in multiagent systems. In *International Conference on Machine Learning*, pp. 1797–1806, 2018.

Hauwere, Y. D., Vrancx, P., and Nowé, A. Learning multiagent state space representations. In *International Conference on Autonomous Agents and MultiAgent Systems*, pp. 715–722, 2010.

Hu, S., Zhu, F., Chang, X., and Liang, X. Updet: Universal multi-agent reinforcement learning via policy decoupling with transformers. In *International Conference on Learning Representations*, 2021.

Jiang, J. and Lu, Z. Learning attentional communication for multi-agent cooperation. In *Advances in Neural Information Processing Systems*, pp. 7254–7264, 2018.

Jiang, J., Dun, C., Huang, T., and Lu, Z. Graph convolutional reinforcement learning. In *International Conference on Learning Representations*, 2019.

Long, Q., Zhou, Z., Gupta, A., Fang, F., Wu, Y., and Wang, X. Evolutionary population curriculum for scaling multiagent reinforcement learning. In *International Conference on Learning Representations*, 2019.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pp. 6379–6390, 2017.

Oliehoek, F. A., Amato, C., et al. *A concise introduction to decentralized POMDPs*. Springer, 2016.

Omidshafiei, S., Kim, D., Liu, M., Tesauro, G., Riemer, M., Amato, C., Campbell, M., and How, J. P. Learning to teach in cooperative multiagent reinforcement learning. In *AAAI Conference on Artificial Intelligence*, pp. 6128–6136, 2019.

Rashid, T., Samvelyan, M., Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4292–4301, 2018.

Samvelyan, M., Rashid, T., de Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., and Whiteson, S. The StarCraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

Silva, F. L. d. and Costa, A. H. R. Transfer learning for multiagent reinforcement learning systems. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 15(3):1–129, 2021.

Tan, M. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *International Conference on Machine Learning*, pp. 330–337, 1993.

Wadhwania, S., Kim, D., Omidshafiei, S., and How, J. P. Policy distillation and value matching in multiagent reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 8193–8200, 2019.

Wang, J., Ren, Z., Liu, T., Yu, Y., and Zhang, C. QPLEX: Duplex dueling multi-agent Q-learning. In *International Conference on Learning Representations*, 2021a.

Wang, T., Dong, H., Lesser, V., and Zhang, C. ROMA: Multi-agent reinforcement learning with emergent roles. In *International Conference on Machine Learning*, 2020a.

Wang, T., Wang, J., Zheng, C., and Zhang, C. Learning nearly decomposable value functions with communication minimization. In *International Conference on Learning Representations*, 2020b.

Wang, T., Gupta, T., Mahajan, A., Peng, B., Whiteson, S., and Zhang, C. RODE: Learning roles to decompose multi-agent tasks. In *International Conference on Learning Representations*, 2021b.

Wang, W., Liu, T. Y. Y., Hao, J., Hao, X., Hu, Y., Chen, Y., Fan, C., and Gao, Y. Action semantics network: Considering the effects of actions in multiagent systems. In *International Conference on Learning Representations*, 2020c.

Wang, W., Yang, T., Liu, Y., Hao, J., Hao, X., Hu, Y., Chen, Y., Fan, C., and Gao, Y. From few to more: Large-scale dynamic multiagent curriculum learning. In *AAAI Conference on Artificial Intelligence*, pp. 7293–7300, 2020d.

Wang, Y., Han, B., Wang, T., Dong, H., and Zhang, C. Dop: Off-policy multi-agent decomposed policy gradients. In *International Conference on Learning Representations*, 2021c.

Xie, A., Losey, D. P., Tolsma, R., Finn, C., and Sadigh, D. Learning latent representations to influence multi-agent interaction. In *Conference on Robot Learning*, pp. 575–588, 2020.

Yang, T., Wang, W., Tang, H., Hao, J., Meng, Z., Mao, H., Li, D., Liu, W., Chen, Y., Hu, Y., et al. An efficient transfer learning framework for multiagent reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Zhang, S., Shen, L., and Han, L. Learning meta representations for agents in multi-agent reinforcement learning. *arXiv preprint arXiv:2108.12988*, 2021.

Zhou, T., Zhang, F., Shao, K., Li, K., Huang, W., Luo, J., Wang, W., Yang, Y., Mao, H., Wang, B., et al. Cooperative multi-agent transfer learning with level-adaptive credit assignment. *arXiv preprint arXiv:2106.00517*, 2021.

Zhu, Z., Lin, K., and Zhou, J. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

A. Appendix

A.1. StarCraft II Micromanagement Benchmark (SMAC)

SMAC (Samvelyan et al., 2019) is a combat scenario of StarCraft II unit micromanagement tasks, which is a popular benchmark for multi-agent reinforcement learning algorithms. We consider a partial observation setting, where an agent can only see a circular area around it with a radius equal to the sight range, which is default to 9. We train the ally units with reinforcement learning algorithms to beat enemy units controlled by the built-in AI. At the beginning of each episode, allies and enemies are generated at specific regions on the map. Every agent takes action from the discrete action space at each timestep, including the following actions: no-op, move [direction], attack [enemy id], and stop. Under the control of these actions, agents can move and attack in continuous maps. Agents will get a shard reward equal to the total damage done to enemy units at each timestep. Killing each enemy unit and winning the combat (killing all the enemies) will bring additional bonuses of 10 and 200, respectively. We consider three settings which contain multiple micromanagement maps, and each includes various single tasks, the detailed descriptions are shown in Tables 5∼7.

Table 5. Description of MMM_series SMAC maps.

Map Name	Ally Units	Enemy Units	Type	Difficulty
MMM0	1 Medivac, 2 Marauders, 5 Marines	1 Medivac, 2 Marauders, 5 Marines	Asymmetric & Heterogeneous	Easy
MMM	1 Medivac, 2 Marauders, 7 Marines	1 Medivac, 2 Marauders, 7 Marines	Asymmetric & Heterogeneous	Easy
MMM1	1 Medivac, 1 Marauders, 7 Marines	1 Medivac, 2 Marauders, 7 Marines	Asymmetric & Heterogeneous	Hard
MMM2	1 Medivac, 2 Marauders, 7 Marines	1 Medivac, 3 Marauders, 8 Marines	Asymmetric & Heterogeneous	Super Hard
MMM3	1 Medivac, 2 Marauders, 8 Marines	1 Medivac, 3 Marauders, 9 Marines	Asymmetric & Heterogeneous	Super Hard
MMM4	1 Medivac, 3 Marauders, 8 Marines	1 Medivac, 4 Marauders, 9 Marines	Asymmetric & Heterogeneous	Super Hard
MMM5	1 Medivac, 3 Marauders, 8 Marines	1 Medivac, 4 Marauders, 10 Marines	Asymmetric & Heterogeneous	Super Hard
MMM6	1 Medivac, 3 Marauders, 8 Marines	1 Medivac, 4 Marauders, 11 Marines	Asymmetric & Heterogeneous	Super Hard

A.2. Network Architecture and Hyperparameters

Our implementation of Mattar is based on PyMARL[2] (Samvelyan et al., 2019) with StarCraft 2.4.6.2.69232 and uses its default hyperparameter settings. We apply the default ϵ-greedy action selection algorithm to every algorithm, as ϵ decays from 1 to 0.05 in 50K timesteps. We also adopt typical Q-learning training tricks like target networks and double Q-learning. Mattar has additional hyperparameters λ_1, λ_2, and λ for doing representation learning, the scaling factors for observation prediction loss, reward prediction loss, and an entropy regularization term, respectively. We set these additional parameters to $1, 10$, and 0.1 across all experiments. We use the default configurations for QMIX in the PyMARL framework. For RODE (Wang et al., 2021b), ASN (Wang et al., 2020c), QPLEX (Wang et al., 2021a), QMIX (Rashid et al., 2018), and UPDET (Hu et al., 2021), we use the codes provided by the authors from their original papers with default hyperparameters settings. For the hyperparameters concerning network structure, our selection is listed in Table 8. We used this set of hyperparameters in

[2] https://github.com/oxwhirl/pymarl

Table 6. Description of sz_series SMAC maps.

Map Name	Ally Units	Enemy Units	Type	Difficulty
2s3z	2 Stalkers, 3 Zealots	2 Stalkers, 3 Zealots	Symmetric & Heterogeneous	Easy
2s3z_vs_2s4z	2 Stalkers, 3 Zealots	2 Stalkers, 4 Zealots	Symmetric & Heterogeneous	Hard
3s4z	3 Stalkers, 5 Zealots	3 Stalkers, 4 Zealots	Symmetric & Heterogeneous	Easy
3s5z	3 Stalkers, 5 Zealots	3 Stalkers, 5 Zealots	Symmetric & Heterogeneous	Easy
3s5z_vs_3s6z	3 Stalkers, 5 Zealots	3 Stalkers, 6 Zealots	Symmetric & Heterogeneous	Super Hard
3s5z_vs_3s7z	3 Stalkers, 5 Zealots	3 Stalkers, 7 Zealots	Symmetric & Heterogeneous	Super Hard
4s7z	4 Stalkers, 7 Zealots	4 Stalkers, 7 Zealots	Symmetric & Heterogeneous	Easy
4s7z_vs_4s8z	4 Stalkers, 7 Zealots	4 Stalkers, 8 Zealots	Symmetric & Heterogeneous	Super Hard

all experiments.

A.3. Experimental Details

Our experiments were performed on a desktop machine with 2 NVIDIA GTX 2080 Ti GPUs. For all the performance curves in our paper, we pause training every 10K timesteps and evaluate for 32 episodes with decentralized greedy action selection. We evaluate the test win rate, the percentage of episodes in which the agents defeat all enemies within the time limit in 32 testing episodes for SMAC. For each part of experiments in our paper, descriptions about experimental details are as follows:

Generalizability to unseen tasks For each compared algorithm, we carried out 5 experiments with different random seeds. In each experiment, we evaluate trained model for 32 episodes on target task. The results recorded in Tables 1∼3 are the mean results for these 5 random seeds.

Task representations provide a good initialization for fine-tune For the performance of transfer learning, we trained 2 source models with different random seeds for each map and carried out transfer learning experiments with 2 random seeds for each source model. For the performance of learning from scratch, we carried out 4 experiments with different random seeds for each map.

Benefits of multi-task learning We carried out 5 experiments with different random seeds for both multi-task learning and learning on single task. For the experiments of multi-task learning on three tasks shown in the paper, the sets of tasks are {5m, 5m_vs_6m, 8m_vs_9m, 10m_vs_11m}, {2s3z, 3s5z, 3s5z_vs_3s6z}, and {MMM, MMM2, MMM4}, respectively.

Bonus: performance on single-task training For this experiment, we carried out 5 experiments for each compared algorithm, and did evaluation during training process as we described above.

A.4. Forward Model for Task Representation Learning

In our method, we utilize forward model learning to help build task representations which can capture the similarity between different tasks. We use a hypernetwork as representation explainer to generate the parameters of forward model. In practical implementation, we design the forward model as two components, an encoder and a decoder (Fig. 6(a)). We use similar techniques to those used in designing Q-value functions to make the encoder a population-invariant structure and let the decoder be a task-specific structure.

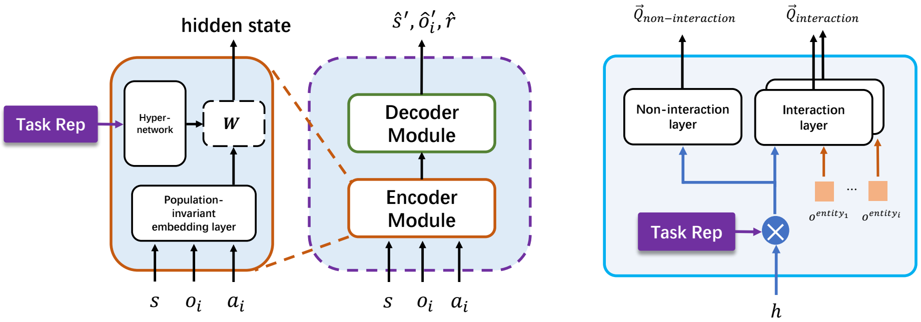
For the encoder network, we first use the population-invariant embedding layer to get a fixed-dimension embedding vector and feed it into a fully-connected layer generated by the hypernetwork representation explainer. The output hidden variable is then fed into the decoder to predict the next state, the next observation, and the global reward. The encoder module

Table 7. Description of m_series SMAC maps.

Map Name	Ally Units	Enemy Units	Type	Difficulty
3m	3 Marines	5 Marines	Symmetric & Homogeneous	Easy
4m	4 Marines	5 Marines	Symmetric & Homogeneous	Easy
4m_vs_5m	4 Marines	5 Marines	Asymmetric & Homogeneous	Hard
5m	5 Marines	5 Marines	Symmetric & Homogeneous	Easy
5m_vs_6m	5 Marines	6 Marines	Asymmetric & Homogeneous	Hard
6m	6 Marines	6 Marines	Symmetric & Homogeneous	Easy
6m_vs_7m	6 Marines	7 Marines	Asymmetric & Homogeneous	Hard
7m	7 Marines	7 Marines	Symmetric & Homogeneous	Easy
7m_vs_8m	7 Marines	8 Marines	Asymmetric & Homogeneous	Hard
8m	8 Marines	8 Marines	Symmetric & Homogeneous	Easy
8m_vs_9m	8 Marines	9 Marines	Asymmetric & Homogeneous	Easy
9m	9 Marines	9 Marines	Symmetric & Homogeneous	Easy
9m_vs_10m	9 Marines	10 Marines	Asymmetric & Homogeneous	Easy
10m	10 Marines	10 Marines	Symmetric & Homogeneous	Easy
10m_vs_11m	10 Marines	11 Marines	Asymmetric & Homogeneous	Easy
10m_vs_12m	10 Marines	12 Marines	Asymmetric & Homogeneous	Super Hard

Table 8. Hyperparameters concerning network structure in experiments.

name	value
mixing_embed_dim	32
hypernet_layers	2
hypernet_embed	64
id_length	4
task_repre_dim	32
state_latent_dim	32
entity_embed_dim	64
attn_embed_dim	8

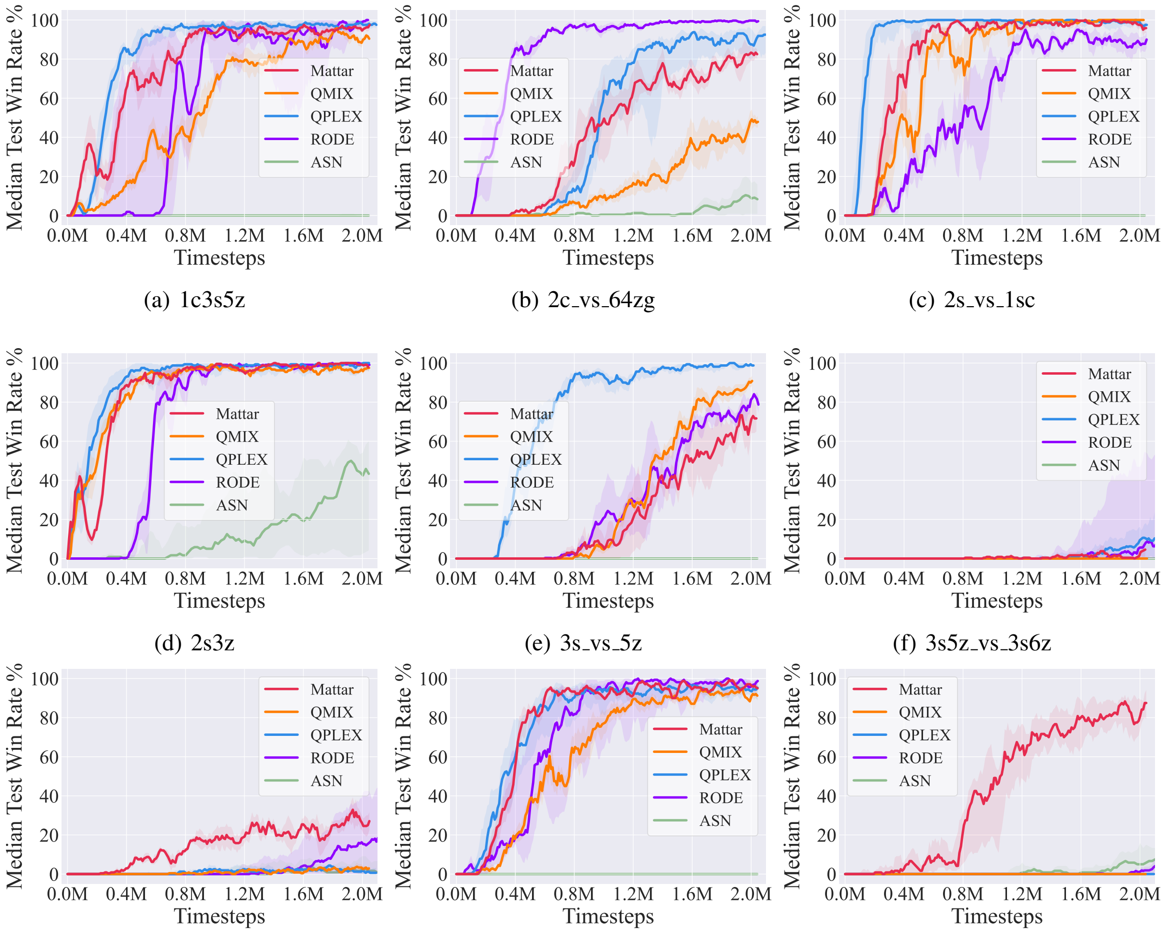


(a) Network architecture for forward model

(b) Network architecture for interaction actions

Figure 6. Supplemental descriptions about network architecture for forward model and interaction actions.

and the hypernetwork are shared among tasks and are fixed when learning new task representations, while the decoder module is task-specific, and we allow the decoder to be optimized together with task representations when adapting to unseen tasks. This solution is a trade-off between allowing a designed forward model expressive enough to solve the forward-prediction problems in different tasks by using individual decoders and capturing the similarity between tasks by sharing the hypernetwork and the encoder module which are core components of the forward model.



(a) 1c3s5z

(b) 2c_vs_64zg

(c) 2s_vs_1sc

(d) 2s3z

(e) 3s_vs_5z

(f) 3s5z_vs_3s6z

(g) 6h_vs_8z

(h) 10m_vs_11m

(i) corridor

Figure 7. Supplemental results for the performance of MATTAR on SMAC benchmark when learning from scratch on single tasks.

For the population-invariant structure in the encoder module, we decompose the input state and observation into several entity-specific components, pass them through an embedding layer, respectively, and do pooling operation for their output vectors. We also decompose the action input to deal with the situation of dynamic-dimension action input. We incorporate the decomposed action to observation o_i, concatenating non-interaction portions with agent i's own observation component o_i^{own} and interaction portions with corresponding entity's observation component. We claim that other population-invariant structures can be applied to our approach to polish our work further.

A.5. Performance of Mattar on More SMAC Maps for Single-Task Learning

The additional results are shown in Fig. 7.

A.6. Dynamic Action Dimensions

In some multi-agent environments, there exist interaction actions that have semantics relating to an entity. In this case, the action dimension is related to the number of agents in the environment. For this problem, we design a particular structure to calculate Q-values for these interaction actions, as shown in Fig. 6(b).

With the help of previous techniques, we have already obtained a fixed-dimension embedding vector h for dynamic-dimension observation. For non-interaction actions, we use a fully-connected network to compute the Q-value vector with the concatenation of h and task representation as input. For an interaction action, we use an action-sharing network, which takes as input the concatenation of h, task representation z, and observation component relating to the entity for that interaction action, to output a scalar as the Q-value of that action.