From Few to More: Large-scale Dynamic Multiagent Curriculum Learning

Weixun Wang[1] *, Tianpei Yang[1] *, Yong Liu[2] *, Jianye Hao[1] ✉, Xiaotian Hao[1], Yujing Hu[3], Yingfeng Chen[3], Changjie Fan[3], Yang Gao[2],
[1]Tianjin University, {wxwang, tpyang, jianye.hao, xiaotianhao}@tju.edu.cn
[2]Nanjing University, lucasliunju@gmail.com, gaoy@nju.edu.cn
[3]NetEase Fuxi AI Lab, {huyujing, chenyingfeng1, fanchangjie}@corp.netease.com

Abstract

A lot of efforts have been devoted to investigating how agents can learn effectively and achieve coordination in multiagent systems. However, it is still challenging in large-scale multiagent settings due to the complex dynamics between the environment and agents and the explosion of state-action space. In this paper, we design a novel Dynamic Multiagent Curriculum Learning (DyMA-CL) to solve large-scale problems by starting from learning on a multiagent scenario with a small size and progressively increasing the number of agents. We propose three transfer mechanisms across curricula to accelerate the learning process. Moreover, due to the fact that the state dimension varies across curricula,, and existing network structures cannot be applied in such a transfer setting since their network input sizes are fixed. Therefore, we design a novel network structure called Dynamic Agent-number Network (DyAN) to handle the dynamic size of the network input. Experimental results show that DyMA-CL using DyAN greatly improves the performance of large-scale multiagent learning compared with state-of-the-art deep reinforcement learning approaches. We also investigate the influence of three transfer mechanisms across curricula through extensive simulations.

Introduction

Reinforcement learning (RL) (Sutton and Barto 2018) has achieved great success in achieving human-level control in complex tasks (Mnih et al. 2015). However, there also exist a lot of challenges in multiagent systems (MASs) where a group of autonomous agents in a shared environment from which they learn what to do according to the reward signals received while interacting with each other. (Claus and Boutilier 1998; Busoniu, Babuska, and Schutter 2008). Furthermore, in large-scale multiagent systems, the dynamics and stochasticity of the environment become more complex, which makes it more challenging to achieve coordination among agents (Singh, Jain, and Sukhbaatar 2019; Yang et al. 2018a; Jiang and Lu 2018).

One efficient way to address large-scale multiagent learning problems is to leverage the concept of Curriculum Learning (CL), which has been an active field of research in the past few years, especially regarding its application to RL. The Curriculum Learning, consists of defining a set of source tasks and training the agent on each of them individually before progressing to learning on the full task.

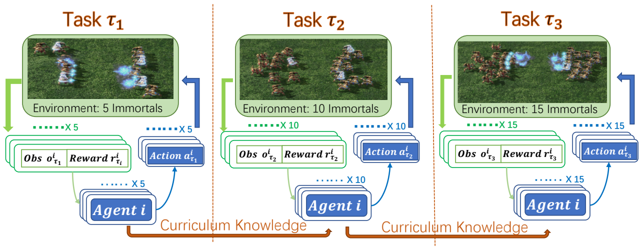


Figure 1: An example of DyMA-CL in StarCraft II.

One major direction of applying CL to RL focuses on how to deal with increasingly complicated tasks. Andreas et al. (2017) used curriculum learning to make their model scale up smoothly from simple tasks to difficult ones, the difficulty of each task is associated with sketches of different length. Later, Wu and Tian (2017) integrated RL with CL for the complex video game Doom, and developed an adaptive curriculum training that samples from a varying distribution of tasks to train the model, which achieves higher scores than learning the target task directly. However, these methods simply manually design the curricula which requires piror knowledge. Another direction of CL is to automatically design the curriculum. Narvekar et al. (2017a) proposed formulating the selection of tasks using a Curriculum Markov Decision Process (CMDP). However, whether the curriculum policy could actually be learned is not demonstrated. Later, they (2019) addressed this problem by exploring various representations to learn the curriculum policy.

However, all the above approaches focus on designing the curricula manually or automatically in single-agent learning tasks. Although some existing works consider CL in multiagent settings, the way they utilize CL is quite simple, which is not the focus of these works (Agarwal, Kumar, and Sycara 2019). To address the growing challenges as the increase of agent-number in large-scale MASs, in this paper, we firstly propose a novel multiagent CL, named Dynamic Multiagent Curriculum Learning (DyMA-CL) as shown in Figure

* Equal contribution. ✉Corresponding author. We provide a video introducing our DyMA-CL and experimental demonstrations in https://github.com/wwxFromTju/wwxFromTju.github.io/blob/master/video/DyMA.mp4?raw=true.

1. DyMA-CL solves large-scale problems by starting from learning on a small-size multiagent scenario and progressively increasing the number of agents to learn the target task finally. Three kinds of transfer mechanisms (Buffer Reuse, Curriculum Distillation, and Model Reload) are proposed across different tasks to accelerate the curriculum learning process. The first two mechanisms do not require a specific network structure, while the last one does since existing network architectures cannot be directly used in such a multiagent CL setting due to the fixed size of network input and the state dimension in our settings varies across curricula. Thus, we design a novel network structure called Dynamic Agent-number Network (DyAN) by combining graph neural network to handle the dynamic size of the network input. Experimental results in Starcraft-II (Samvelyan et al. 2019) and MAgent (Zheng et al. 2018) show that DyMA-CL greatly improves the performance on large-scale problems compared with state-of-the-art DRL approaches; and three kinds of transfer mechanisms across curricula greatly boost the performance of DyMA-CL.

Background

Partially Observable Stochastic Games

A natural multiagent extension of Markov decision processes (MDPs) are Stochastic Games (SGs) (Littman 1994), which model the dynamic interactions among multiple agents. In this paper, we follow previous work's settings and model the multiagent learning problems as Partially Observable Stochastic Games (POSGs) (Hansen, Bernstein, and Zilberstein 2004) considering that agents may not have access to the complete environmental information.

A *Partially Observable Stochastic Game* (POSG) is defined as a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}^1, \cdots, \mathcal{A}^n, T, \mathcal{R}^1, \cdots, \mathcal{R}^n, \mathcal{O}^1, \cdots, \mathcal{O}^n \rangle$, where $\mathcal{N}$ is the set of n agents; $\mathcal{S}$ is the state set; $\mathcal{A}^i$ is the set of actions available to agent i ($\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2 \times \cdots \times \mathcal{A}^n$ is the joint action space); T is the transition function that defines transition probabilities between states: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$; $\mathcal{R}^i$ is the reward function for agent i: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $\mathcal{O}^i$ is the observation set of agent i.

Note that each state $s \in \mathcal{S}$ contains the possible configurations of the environment and all agents, while each agent i draws a private observation o^i correlated with the state: $\mathcal{S} \mapsto \mathcal{O}^i$, e.g., an agent's observation includes the agent's private information and the relative distance between itself and other agents. Formally, an observation of agent i at step t can be constructed as follows: $o_t^i = \{o_t^{i,env}, m_t^i, o_t^{i,1}, \cdots, o_t^{i,i-1}, o_t^{i,i+1}, \cdots, o_t^{i,n}\}$, where $o_t^{i,env}$ describes the surrounding environmental information, m_t^i is agent i's private property (e.g., in robotics, m_t^i includes agent i's location, the battery power and the healthy status of each component) and the rest are the observations of agent i on other agents (e.g., in robotics, $o_t^{i,i-1}$ includes the agent i's observation about the relative location, the exterior of agent $i-1$). A policy π_i: $\mathcal{O}^i \times \mathcal{A}^i \to [0; 1]$ specifies the probability distribution over the action space of agent i. The goal of agent i is to learn the optimal policy π_i^* that maximizes the expected return with a discount factor γ: $J = \mathbb{E}_{\pi_i^*}\left[\sum_{t=0}^{\infty} \gamma^t r_t^i\right]$.

Curriculum Learning

Curriculum Learning (CL) is firstly introduced in (Bengio et al. 2009) which is defined as a Machine Learning notion to improve the performance of Supervised Learning. The idea of CL is inspired by observing the way humans learn that starts with simple, small problems and gradually progresses to more complex, difficult tasks. In Curriculum Learning, the goal is to generate a series of training tasks, beginning from the simplest one and then gradually increasing the difficulty of training to improve the final asymptotic performance or decrease the training time.

Narvekar et al. (2016) firstly applied CL to RL and proposed a new CL framework. They generated a sequence of RL source tasks, named "Curriculum", trained the agent on each of the source tasks and then on the target task. Different from CL in Supervised Learning, each task in the RL curriculum is defined as an Markov Decision Process (MDP). The difficulty of each task is controlled by eliminating certain actions or states, modifying the transition or reward function, or changing the starting or terminal distributions of MDPs. The sequence of source tasks can be manually designed or automated generated (Narvekar, Sinapov, and Stone 2017a; Narvekar, Sinapov, and Stone 2017b). In this paper, we focus on CL in multiagent RL settings and design a dynamic multiagent curriculum learning to solve large-scale multiagent learning problems.

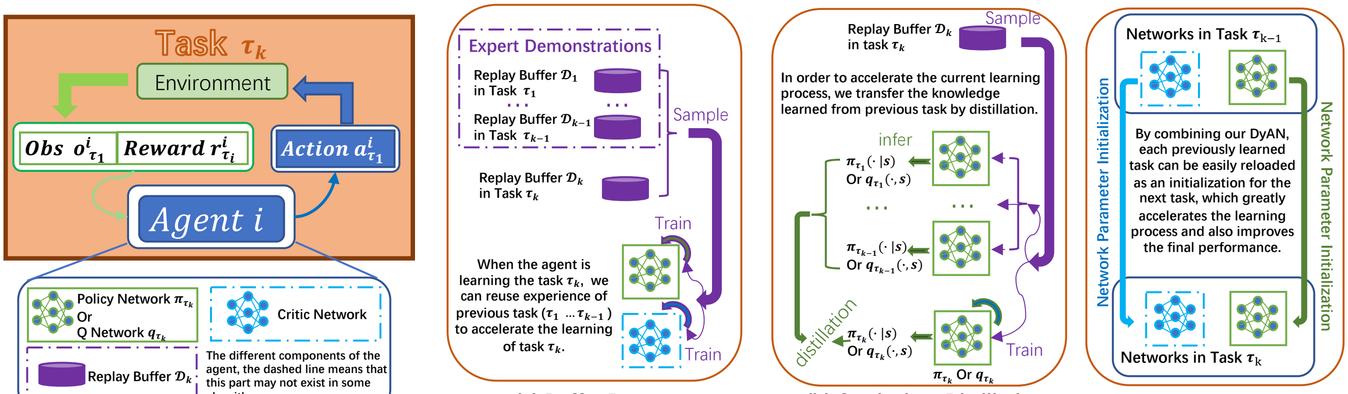
Dynamic Multiagent Curriculum Learning

Large-scale Multiagent Systems

Multiagent learning receives much attention and how to achieve multiagent coordination is the key problem. Recent researches have found that the difficulty of multiagent learning is exponentially increasing as the number of agents increases (Samvelyan et al. 2019). Moreover, in large-scale multiagent systems, the dynamics and stochasticity of the environment become more complex, which makes it more challenging to achieve coordination among agents (Chen et al. 2018). We first propose several multiagent properties in nature which are commonly existing in MASs, and then utilize these properties to address large-scale multiagent learning problems.

Property 1 *Partial Observability: In MASs, agents make decisions based on their local observations, in which way large-scale problems can be reduced to relatively independent but correlated small-size ones.*

The common settings are to model multiagent learning problems as partially observable stochastic games (POSGs). In such partially observable environments, each agent selects an action based on its local (partial) observation, and the number of agents in each agent's vision is changing all the time as agents move and execute actions. For example, when the agent drives a car on the road, the number of cars in his local vision changes (Singh, Jain, and Sukhbaatar 2019). The number of cars in the driver's vision decreased when the road is crowded at first and then some cars go out of his vision, learning in this situation is similar to learning in a scenario with a small number of cars. Therefore, large-scale learning problems can be naturally transformed into small ones based on *Partial Observability*.


Task τ_k
Environment
Obs $o^i_{\tau_1}$
Reward $r^i_{\tau_i}$
Action $a^i_{\tau_1}$
Agent i
Policy Network π_{τ_k} Or Q Network q_{τ_k}
Critic Network
Replay Buffer $\mathcal{D}_k$
The different components of the agent, the dashed line means that this part may not exist in some algorithms.
Expert Demonstrations
Replay Buffer $\mathcal{D}_1$ in Task τ_1
Replay Buffer $\mathcal{D}_{k-1}$ in Task τ_{k-1}
Replay Buffer $\mathcal{D}_k$ in Task τ_k
Sample
Train
When the agent is learning the task τ_k, we can reuse experience of previous task ($\tau_1 \dots \tau_{k-1}$) to accelerate the learning of task τ_k.
Replay Buffer $\mathcal{D}_k$ in task τ_k
In order to accelerate the current learning process, we transfer the knowledge learned from previous task by distillation.
infer
$\pi_{\tau_1}(\cdot|s)$ Or $q_{\tau_1}(\cdot,s)$
$\pi_{\tau_{k-1}}(\cdot|s)$ Or $q_{\tau_{k-1}}(\cdot,s)$
$\pi_{\tau_k}(\cdot|s)$ Or $q_{\tau_k}(\cdot,s)$
π_{τ_k} Or q_{τ_k}
distillation
Networks in Task τ_{k-1}
By combining our DyAN, each previously learned task can be easily reloaded as an initialization for the next task, which greatly accelerates the learning process and also improves the final performance.
Network Parameter Initialization
Networks in Task τ_k

(a) Buffer Reuse

(b) Curriculum Distillation

(c) Model Reload (DyAN)

Figure 2: An illustration of DyMA-CL using different transfer mechanisms.

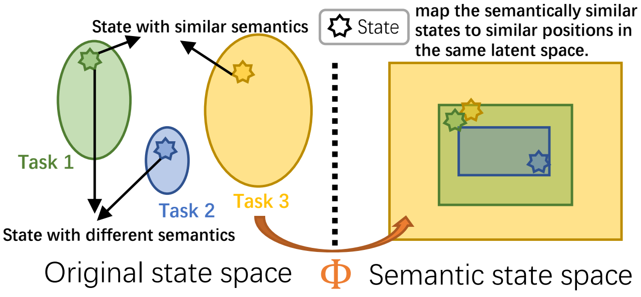

State with similar semantics
State
map the semantically similar states to similar positions in the same latent space.
Task 1
Task 2
Task 3
State with different semantics
Original state space
Φ
Semantic state space

Figure 3: Mapping original states to the semantic state space.

Property 2 *Sparse Interactivity: From the perspective of the global view, each agent only interacts with some of the agents in MASs at the same time, and the interactions do not happen all the time.*

In multiagent systems, agents often only need to cooperate with their neighbors and finally achieve the overall coordination, which makes a sparse-interaction environment. For example, in the predator-prey environment (Yang et al. 2018b), each predator would cooperate with its neighboring predators to catch the surrounding prey, without considering other preys in a larger distance.

Property 3 *State Semanticity: Each state contains semantic information which can be utilized to measure the similarity between states.*

Although states in different POSGs hold different dimensions, they may contain semantically similar information, which can be utilized to measure the similarity of these states. For example, in StarCraft II, with the dynamics of the game continue, the number of agents would decrease if either side of soldiers die in the battle, in which situation learning is similar to that in a small-size battlefield. As shown in Figure 3, given three tasks with their state spaces in different colors respectively, we can learn a mapping function Φ to represent the relations of these states in the semantic state space. Thus, the state semanticity property can be naturally used for transfer across different multiagent scenarios.

With the increase of the number of agents, the difficulties and challenges mentioned above become more severe, which makes it harder or even impossible to learn from scratch in such large-scale multiagent systems (Samvelyan et al. 2019). Inspired by the above properties, we design a dynamic multiagent curriculum learning to address large-scale multiagent learning problems, i.e., starting from learning in an environment with a small number of agents, and then progressively increasing the number of agents, and finally finishing the curriculum which is described in detail in the following section.

Knowledge Transfer across DyMA-CL

In this section, we propose a novel curriculum learning mechanism called dynamic multiagent curriculum learning (DyMA-CL) for efficient large-scale multiagent learning. To the best of our knowledge, it is challenging and difficult to learn on a large-scale multiagent scenario, e.g., win the battle in large-scale StartCraft II scenarios (e.g., a 15 Immortals vs 15 Immortals scenario) using existing methods (Samvelyan et al. 2019; Rashid et al. 2018). Therefore, we build the curriculum with the increase of the agent-number to learn on a large-scale multiagent scenario. The sequence of tasks can be manually designed or automated generated. Figure 1 illustrates an example of the DyMA-CL with 3 tasks in StarCraft II. The target task is to win on a 15 immortals vs 15 immortals scenario. We first learn the task I on a 5 immortals vs 5 immortals scenario, then learn the task II on a 10 immortals vs 10 immortals scenario and finally learn the target task. We also incorporate different knowledge transfer mechanisms across neighboring curricula which are described in detail as follows.

Figure 2 shows the whole framework of DyMA-CL using different transfer mechanisms. The simplest transfer is to directly reload the model trained in previous curricula as an initialization for current task learning (Figure 2(c)). However, **Model Reload** is infeasible since the input of regular training networks is fixed while different curricula have different state spaces which makes the input size changing. The policy network needs to be specially designed to be suitable for different input sizes. We first propose two kinds of transfer mechanisms without any constraints on the network design: Buffer Reuse (Figure 2(a)) and Distillation via KL Divergence (Figure 2(b)). How to redesign the network to support parameter transfer will be discussed in the next section.

Buffer Reuse Inspired by deep Q-learning from demonstrations (Hester et al. 2018) which incorporates extra expert demonstrations as the supervision, we propose a novel transfer mechanism called Buffer Reuse. For the agent learns the sequence of tasks $\tau_1, \tau_2, \cdots, \tau_k$ using one of the off-policy RL algorithm, e.g., DQN (Mnih et al. 2015), it is equipped with an experience replay buffer $\mathcal{D}_i$ for each task τ_i, which stores the corresponding transition samples. When the agent is learning the task τ_k, we can reuse experience of previously learned tasks to accelerate learning c_k. Specifically, we keep a sequence of replay buffers $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_{k-1}$ for previously learned tasks $\tau_1, \tau_2, \cdots, \tau_{k-1}$ and sample a batch of b transitions from each replay buffer equally to reuse these good transition samples as expert demonstrations, we also sample a batch from the current buffer $\mathcal{D}_k$ and then minimize the following loss in each training step:

$$\text{Loss} = \sum_{i=1}^{k} \sum_{j=1}^{b} \left[\left(r_i^j + \gamma \max_{a_i^{\prime j}} q_{\tau_i}(s_i^{\prime j}, a_i^{\prime j}) - q_{\tau_i}(s_i^j, a_i^j) \right)^2 \right] \tag{1}$$

where $\left(s_i^j, a_i^j, s_i^{\prime j}, r_i^j\right)$ is the jth transition sample from the replay buffer $\mathcal{D}_i$ for task i, γ is a discount factor.

Note that the state space is different across tasks since the number of agents varies, i.e., the dimension of a state in task τ_i is larger than that in task τ_j if $i > j$. Therefore, these samples cannot be collected together to calculate the loss in Equation (1) directly. Here we modify the sampled transitions to reshape them as the same dimension first, e.g., add zero-padding for those samples with a smaller size of states, and then execute the buffer reuse mechanism to accelerate the learning process.

Curriculum Distillation The second transfer mechanism adopts the distillation via Kullback-Leibler (KL) divergence (Rusu et al. 2016) as the supervision which is a more general pattern suitable for both on-policy and off-policy RL algorithms.

Given a learned sequence of tasks $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_{k-1}\}$ and the current task τ_k, in order to accelerate the current learning process, we transfer the knowledge from previously learned tasks by distillation. Specifically, we add an extra distillation loss L_{Distil} to the regular RL loss L_{RL} using the KL divergence with some temperature ω: Loss $= L_{\text{RL}} + L_{\text{Distil}}$, where we can distil either Q-values or policies:

$$L_{\text{Distil}} = \sum_{i=1}^{k-1} \text{KL}(\pi_{\tau_i} || \pi_{\tau_k}) \quad \text{or}$$

$$L_{\text{Distil}} = \sum_{i=1}^{k-1} \sum_{j=1}^{|\mathcal{D}_k|} \text{softmax}\left(\frac{\mathbf{q}_{\tau_i}(s_j)}{\omega}\right) \ln \frac{\text{softmax}\left(\frac{\mathbf{q}_{\tau_i}(s_j)}{\omega}\right)}{\text{softmax}(\mathbf{q}_{\tau_k}(s_j))} \tag{2}$$

Where, π_{τ_i} is the policy for task τ_i and ω is the temperature that controls the proportion transferred to the curriculum τ_k. Similar to the cases in Buffer Reuse mechanism, states as the network input for different curricula should be reshaped to the same size first.

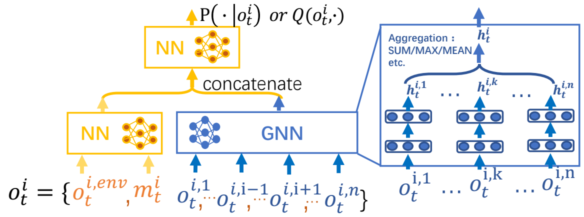

$P(\cdot|o_t^i)$ or $Q(o_t^i, \cdot)$
NN
concatenate
NN
GNN
Aggregation : SUM/MAX/MEAN etc.
h_t^i
$h_t^{i,1}$ … $h_t^{i,k}$ … $h_t^{i,n}$
$o_t^i = \{o_t^{i,env}, m_t^i$ $o_t^{i,1}, \cdots o_t^{i,i-1}, \cdots o_t^{i,i+1}, \cdots o_t^{i,n}\}$
$o_t^{i,1}$ … $o_t^{i,k}$ … $o_t^{i,n}$

Figure 4: The network structure of DyAN.

Dynamic Number Agent Network

As mentioned above, each kind of transfer mechanisms cannot directly be used in our DyMA-CL, since the number of agents varies across curricula and the dimension of each agent i's observation o_t^i at each step t changes, i.e., the number of observations for other agents changes as the number of agents changes.

Although the state space is different in two environments due to the different number of agents, according to the Property 3, some states in a large-scale environment often contain similar semantic information to that in a small-size one. Therefore, we provide a formal definition of the semantic mapping function $\Phi(\cdot)$ which extracts the semantic information from each agent's observation and indicates the mapping between different state spaces.

Definition 1 *Semantic Mapping Function*
Given three tasks τ_e, τ_f and τ_g with different state dimensions, if state $s_e^{\tau_e}$ and state $s_f^{\tau_f}$ contain similar semantic information while $s_g^{\tau_g}$ does not, then through the mapping function $\Phi(\cdot)$, there exists a latent space that makes the following inequation establish: $dis(\Phi(s_e^{\tau_e}), \Phi(s_f^{\tau_f})) < dis(\Phi(s_e^{\tau_e}), \Phi(s_g^{\tau_g}))$, $dis(\Phi(s_e^{\tau_e}), \Phi(s_f^{\tau_f})) < dis(\Phi(s_f^{\tau_f}), \Phi(s_g^{\tau_g}))$, where $dis(,)$ is the distance between two vectors.

By the definition of Semantic Mapping Function $\Phi(\cdot)$, the states in each task τ_i (each POSG) can be transformed into the same semantic state space, which is also suitable for mapping the local observation of each agent to the same latent space. Thus, we can transfer knowledge across POSGs with different state dimensions. This concept is widely used in domain adaptation area (Higgins et al. 2017; Arnekvist, Kragic, and Stork 2019), while they focus on how to transfer from different tasks with the same state dimension. However, the biggest challenge for DyMA-CL is how to deal with different network input dimensions caused by the different number of agents, and map the semantically similar states to similar positions in the same latent space.

If the network is not restricted by the state/observation of different dimensions, or the states/observations with the same semantics of different dimensions can be mapped to similar positions in the same latent space, then we can easily transfer knowledge from different numbers of POSGs using any of the above mechanisms for efficient curriculum learning. Inspired by Graph Neural Network (GNN) (Xu et al. 2019), to this end, we propose a novel network architecture named **D**ynamic **A**gent-number **N**etwork (DyAN) to address the above problems.

Figure 4 shows the network structure of the DyAN. Given an observation o_t^i of agent i at step t, the left part of DyAN is the general neural networks, e.g., the fully-connected layers, with the environmental information $o_t^{i,env}$ and its private property m_t^i as input. While the reset of o_t^i contains several observations about other agents which change among different curricula. The right part of DyAN incorporates the GNN to handle this dynamic dimensions of input. Specifically, we learn a representation $h_t^{i,j}$ for the agent i's observation $o_t^{i,j}$ on each other agent j, which is achieved after several neural network layers; and then using an aggregation operator to get the output of GNN. Formally, the output of a GNN is:

$$h_t^i = \text{AGGREGATE}\left(\{h_t^{i,j} : j \in N^{-i}\}\right) \tag{3}$$

where, N^{-i} is the set of agents excluding agent i. Note that we use one layer of GNN, a multiple layers of a GNN with neighborhood communication is also suitable here. There are several alternatives for the AGGREGATE operator (Xu et al. 2019), e.g., the MAX operator that represents an element-wise max-pooling, the MEAN operator representing an element-wise mean pooling and the SUM operator, which performance is investigated in the following section. Next, the outputs of two parts of DyAN are concatenated to input to the following neural network layers. The final output is the Q-values or the policy respectively which is subject to the specific RL algorithms.

As we described earlier, the simplest transfer mechanism of model reload cannot be directly used in our curriculum learning due to the dynamic dimensions of the network input. By combining our DyAN, each previously learned model can be easily reloaded as an initialization for the next curriculum learning, which greatly accelerates the learning process and also improves the final performance. In the next section, we investigate the performance of three transfer mechanisms in our curriculum learning in detail.

Simulations

In this section, we evaluate the performance of our DyMA-CL on two large-scale scenarios: 1) StarCraft II, which contains various scenarios for a number of agents to learn coordination to solve complex tasks; and 2) MAgent (Zheng et al. 2018), which is a simulated battlefield with two large-scale armies (groups), e.g., each army consists of 50 soldiers who would be arrayed in the battlefield (a grid world). We first select two representative DRL algorithms from the perspective of Independent learning and Joint-action Learning respectively: IQL (Tampuu et al. 2017), VDN (Sunehag et al. 2018) to investigate the performance of these approaches with and without DyMA-CL on large-scale StarCraft II scenarios. We further compare the performance of various existing DRL approaches (IQL, PPO (Schulman et al. 2017), A2C (Mnih et al. 2016), and ACER (Wang et al. 2017)) with DyMA-CL on large-scale MAgent scenarios to validate the performance of DyMA-CL since independent learning is more difficult to learn in such large-scale multiagent settings without considering the coexistence of other agents. The details of neural network structures, parameter settings and the curriculum schedule are in supplementary materials.

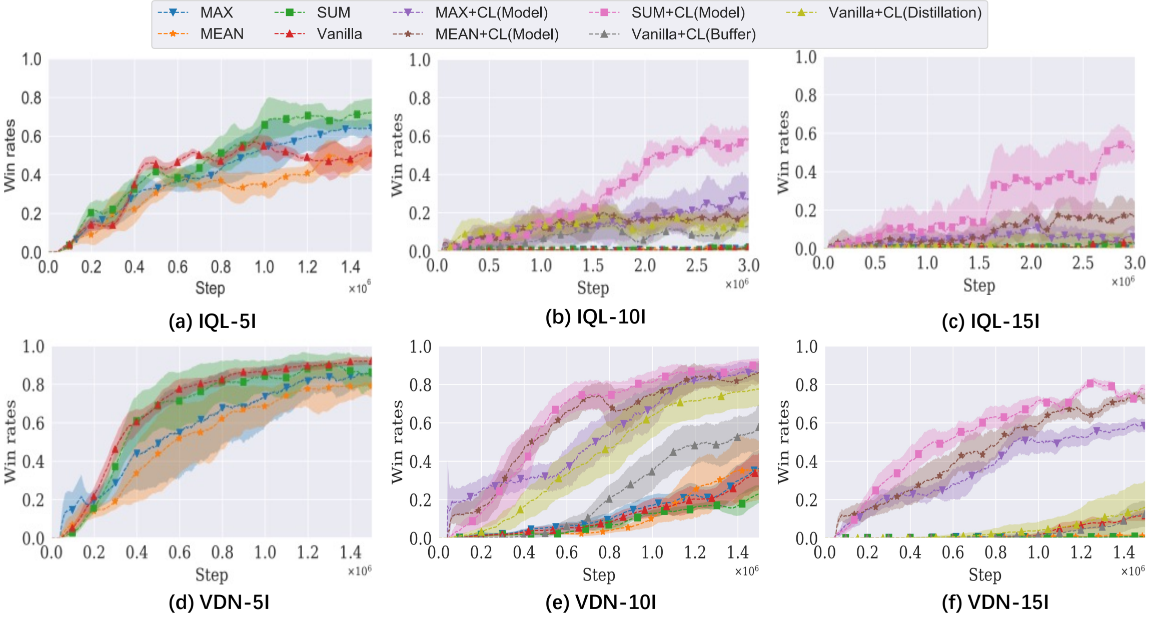
StarCraftII

StarCraft II is a real-time strategy game with one or more humans competing against each other or a built-in game AI. At each step, each agent observes the local game state which consists of the following information for all units in its field of view: relative distance between other units, the position and unit type (detailed in supplementary materials) and selects one of the following actions: move north, south, east or west, attack one of the grid units, stop and the null action. Agents belonging to the same side receive the same joint reward at each time step that equals to the total damage on the enemy units. Agents also receive a joint reward of 10 points after killing each opponent, and 200 points after killing all opponents. The game ends when all agents on one side die or the time exceeds a fixed period.

Note that previous StarCraft II settings enable an agent to attack one of its enemies by choosing one of id numbers (Samvelyan et al. 2019). In this paper, we design the attack action is to choose one of the grid units by dividing the battlefield into several grids, in which case the coordination among agents is much more difficult to achieve. We mainly consider combat scenarios and design a multiagent curriculum learning with the number of agents increasing (see Figure 1) to achieve the victory on a 15 Immortals vs 15 Immortals scenario.

Figure5(a-c) show the average win rate of IQL with and without our DyMA-CL under different network structures (i.e., Vanilla network does not contain the GNN part, and MAX means the GNN uses MAX as the aggregation operator). We can see from Figure5(a) that SUM performs better than other kinds of network structures on the first task of a 5 Immortals vs 5 Immortals scenario. As for the task II on a 10 Immortals vs 10 Immortals battlefield (Figure5(b)), our DyMA-CL with all three transfer mechanisms perform better than learning from scratch. Note that the model reload mechanism performs best among all transfer mechanisms, this is because our proposed DyAN successfully learns the similar semantics of states across curricula, then the model from previously learned curriculum can be directly reused, which leads to a higher win rate. The SUM operator performs best among all three aggregations which means the capability of learning state semantics is different for these three aggregations and the GNN with SUM as aggregation learns more accurate state semantics. This will be explained in detail in the following section. For our last curriculum (Figure5(c)), our DyMA-CL with model reload mechanism performs best among all transfer mechanisms. Similar results as in curriculum II can be found that learning from scratch is too difficult as the increase of the agent number, and the winning rate is never increased. We have conducted the simulation on larger steps ($1e + 7$ steps) and learning from scratch still fails in such a large-scale scenario.

Figure5(d-f) depict the average win rate of VDN with and without our DyMA-CL. We can find different network architectures perform similarly on the first task learning (Figure5(d)), and perform better than that combining IQL in universal. This is because VDN explicitly considers how to coordinate multiple agents using a team reward. Figure5(e) and (f) show the similar and more outstanding performance of DyMA-CL than that in IQL, and the GNN with SUM



(a) IQL-5I

(b) IQL-10I

(c) IQL-15I

(d) VDN-5I

(e) VDN-10I

(f) VDN-15I

Figure 5: Average win rate of IQL and VDN on DyMA-CL.

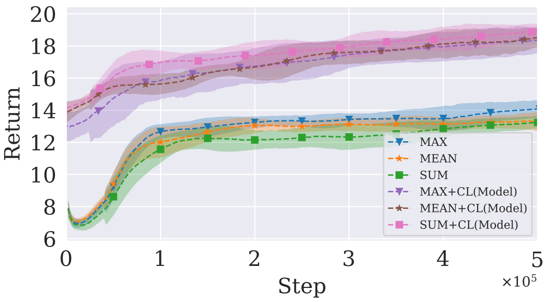


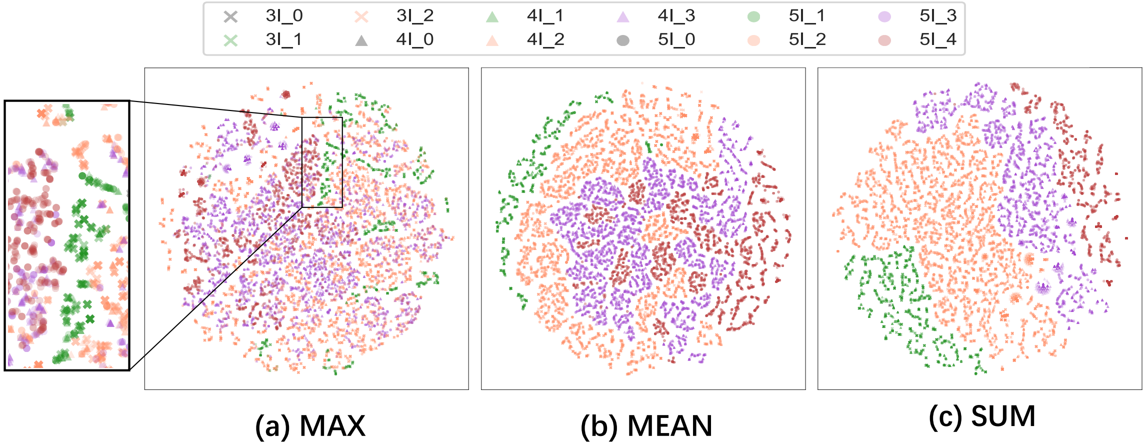
Figure 6: The average rewards of VDN on 15 Immortals vs 15 Immortals.

as aggregation learns best among all three mechanisms of DyMA-CL, and the reason will be discussed in the following section in detail. Note that the common measurement for StarCraft II is the average win rate (Samvelyan et al. 2019), which may hinder the phenomenon of a jumpstart on the performance of DyMA-CL with model reload mechanism. Therefore, we further present the results of average rewards as shown in Figure 6. We can see a jumpstart average reward of our DyMA-CL with model reload mechanism than learning from scratch, which indicates the agents can kill more enemies and achieve higher average rewards at the beginning than learning from scratch, confirming the effectiveness of model reload mechanism across curricula.

Analysis We further investigate the influence of different aggregations on the performance of DyMA-CL. As we discussed in Definition 1, if the two states with different dimensions contain similar semantic information, we can map them to the same neighborhood position in the same latent space. Here we illustrate whether these aggregations learn the semantics of states using three StartCraft II scenarios as examples, each of which contains two groups of 3, 4, 5 agents respectively. Then we input the observation about teammates to DyAN, and use t-SNE (Wattenberg, Viégas, and Johnson 2016) to map the embedding output of the GNN part to a 2-dimension space, as shown in Figure 7(a-c). The different colors denote the state contains different semantic information, e.g., the green color represents that the local observation only contains one teammate, which is actually the same semantics while the input size is different across 3 scenarios. The different shapes represent states in different scenarios, e.g., the triangle denotes the observations from a 4 Immortals vs 4 Immortals scenario. We can see that the mapping result on the SUM aggregation is best among all aggregations, which means SUM learns more accurate state semantics so that the states with the same semantics across different scenarios are mapped to the similar position and each kind of semantics is distinguished clearly. Thus this explains why SUM performs best among three aggregation operations shown in Figure 5.

MAgent

MAgent is a Mixed Cooperative-Competitive scenario with two armies fighting against each other, which supports hundreds to millions of agents. The goal of each army is to get more rewards by collaborating with teammates to destroy all opponents. Each agent selects one of the following actions: moving to some grid unit or attacking some grid unit based on its local observation which contains the following information for all units: the hit points (HP), the positions. We adopt the default reward setting: -0.005 for every move, 0.2 for attacking an enemy, 5 for killing an enemy, -0.1 for attacking an empty grid, and -0.1 for being attacked or killed. We design a curriculum containing 5 tasks, each of which learns on a battlefield with different number of agents



(a) MAX (b) MEAN (c) SUM

Figure 7: Embedding analysis for different aggregation mechanisms.

Table 1: Mean and Standard Error in MAgent ('w/' denotes with and 'w/o' denotes without).

Results / Methods			Survivors	Kill count
IQL	Max	w/ CL	20.35±4.87	50±0
		w/o CL	0.54±2.65	31.83±8.52
	Mean	w/ CL	2.33±2.9	43.58±8.29
		w/o CL	0	5.37±6.92
	Sum	w/ CL	10.52±6.27	49.34± 2.23
		w/o CL	0.05± 0.42	26.01± 7.11
PPO	Max	w/ CL	0.21±1.32	22.76±9.06
		w/o CL	0	4.18±4.84
	Mean	w/ CL	0.22±0.95	23.66±7.52
		w/o CL	0	0.34±0.61
	Sum	w/ CL	1.48±2.69	38.25±7.33
		w/o CL	0	1.06±1.12
A2C	Max	w/ CL	16.5±10.92	47.71±5.61
		w/o CL	3.65±5.58	36.4±13.79
	Mean	w/ CL	6.8±7.39	43.77±10.36
		w/o CL	0.28±1.04	25.21±11.7
	Sum	w/ CL	8.96±8.52	44.59±9.17
		w/o CL	1.07 ±3.75	17.1±14.44
ACER	Max	w/ CL	0	9.14±4.52
		w/o CL	0	4.19±2.52
	Mean	w/ CL	0	12.62±3.68
		w/o CL	0	4.84±2.8
	Sum	w/ CL	0	9.67±3.68
		w/o CL	0	4.84±2.68

$(10vs10, 20vs20, 30vs30, 40vs40, 50vs50)$.

We validate the performance of various independent learning algorithms with or without DyMA-CL. Table 1 presents the average survival teammates and kill count of various approaches in the target task of a 50 agents vs 50 agents scenario. We can see that DyMA-CL with model reload mechanism greatly improves the final performance of IQL than learning from scratch, achieving more survival teammates and a higher average kill count. Similar results can be found in PPO, A2C, and ACER that DyMA-CL boosts the performance of these approaches and outperforms learning from scratch. Note that the performance of ACER is worse than other methods, which is caused by the policy adjustment using samples from its replay buffer. This mechanism is only considered from the perspective of independent learning, ignoring the non-stationary environment caused by other agents. Moreover, DyMA-CL still improves the performance of ACER and achieves a higher average kill count.

Discussion

As noted, we manually design the curriculum for both two domains, StarCraft II and MAgent. Experimental results have shown the great improvement of DyMA-CL on large-scale MASs. However, the boost in the performance in this paper is the first step that validates the effectiveness of DyMA-CL. We have found that the design of the curriculum is a critical factor in the performance of DyMA-CL. How to select an appropriate curriculum schedule (including how to decide on the training step-size for each task, how to select the suitable learned model to reload and so on) is crucial. Researches about automatic generation of the curriculum are still investigated at an initial stage and not considered in multiagent settings. Perhaps the major remaining limitations are how to automatically generate the multiagent curriculum, which will be further investigated as our future work.

Conclusion and Future Work

In this paper, we propose a novel algorithm, Dynamic Multiagent Curriculum Learning (DyMA-CL) to address large-scale multiagent learning problems. We also propose three transfer mechanisms across different curricula to accelerate the learning process, which is extensively validated by simulations. Furthermore, we design a novel network structure, Dynamic Agent-number Network (DyAN) to handle the dynamic size of network input. Experimental results show that DyMA-CL greatly improves the performance in large-scale problems compared with state-of-the-art DRL approaches. As future work, it is worthwhile investigating how to achieve automatically multiagent curriculum learning to accelerate large-scale multiagent learning. Another direction is how to design more efficient transfer mechanisms to facilitate robust multiagent curriculum learning.

References

[Agarwal, Kumar, and Sycara 2019] Agarwal, A.; Kumar, S.; and Sycara, K. P. 2019. Learning transferable cooperative behavior in multi-agent teams. *CoRR* abs/1906.01202.

[Andreas, Klein, and Levine 2017] Andreas, J.; Klein, D.; and Levine, S. 2017. Modular multitask reinforcement learning with policy sketches. In *Proceedings of ICML*, 166–175.

[Arnekvist, Kragic, and Stork 2019] Arnekvist, I.; Kragic, D.; and Stork, J. A. 2019. VPE: variational policy embedding for transfer reinforcement learning. In *Proceedings of ICRA*, 36–42.

[Bengio et al. 2009] Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of ICML*, 41–48.

[Busoniu, Babuska, and Schutter 2008] Busoniu, L.; Babuska, R.; and Schutter, B. D. 2008. A comprehensive survey of multiagent reinforcement learning. *IEEE T SYST MAN CY C* 38(2):156–172.

[Chen et al. 2018] Chen, Y.; Zhou, M.; Wen, Y.; Yang, Y.; Su, Y.; Zhang, W.; Zhang, D.; Wang, J.; and Liu, H. 2018. Factorized q-learning for large-scale multi-agent systems. *arXiv preprint arXiv:1809.03738*.

[Claus and Boutilier 1998] Claus, C., and Boutilier, C. 1998. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI* 1998:746–752.

[Hansen, Bernstein, and Zilberstein 2004] Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. In *Proceedings of AAAI*, 709–715.

[Hester et al. 2018] Hester, T.; Vecerík, M.; Pietquin, O.; Lanctot, M.; Schaul, T.; Piot, B.; Horgan, D.; Quan, J.; Sendonaris, A.; Osband, I.; Dulac-Arnold, G.; Agapiou, J.; Leibo, J. Z.; and Gruslys, A. 2018. Deep q-learning from demonstrations. In *Proceedings of AAAI*, 3223–3230.

[Higgins et al. 2017] Higgins, I.; Pal, A.; Rusu, A. A.; Matthey, L.; Burgess, C.; Pritzel, A.; Botvinick, M.; Blundell, C.; and Lerchner, A. 2017. DARLA: improving zero-shot transfer in reinforcement learning. In *Proceedings of ICML*, 1480–1490.

[Jiang and Lu 2018] Jiang, J., and Lu, Z. 2018. Learning attentional communication for multi-agent cooperation. In *Proceedings of NeurIPS*, 7254–7264.

[Littman 1994] Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of ICML*, 157–163.

[Mnih et al. 2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

[Mnih et al. 2016] Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *Proceedings of ICML*, 1928–1937.

[Narvekar and Stone 2019] Narvekar, S., and Stone, P. 2019. Learning curriculum policies for reinforcement learning. In *Proceedings of AAMAS*, 25–33.

[Narvekar et al. 2016] Narvekar, S.; Sinapov, J.; Leonetti, M.; and Stone, P. 2016. Source task creation for curriculum learning. In *Proceedings of AAMAS*, 566–574.

[Narvekar, Sinapov, and Stone 2017a] Narvekar, S.; Sinapov, J.; and Stone, P. 2017a. Autonomous task sequencing for customized curriculum design in reinforcement learning. In *Proceedings of IJCAI*, 2536–2542.

[Narvekar, Sinapov, and Stone 2017b] Narvekar, S.; Sinapov, J.; and Stone, P. 2017b. Autonomous task sequencing for customized curriculum design in reinforcement learning. In *Proceedings of IJCAI*, 2536–2542.

[Rashid et al. 2018] Rashid, T.; Samvelyan, M.; Witt, C. S.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of ICML*, 4292–4301.

[Rusu et al. 2016] Rusu, A. A.; Colmenarejo, S. G.; Gülçehre, Ç.; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; and Hadsell, R. 2016. Policy distillation. In *Proceedings of ICLR*.

[Samvelyan et al. 2019] 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. 2019. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*.

[Schulman et al. 2017] Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[Singh, Jain, and Sukhbaatar 2019] Singh, A.; Jain, T.; and Sukhbaatar, S. 2019. Individualized controlled continuous communication model for multiagent cooperative and competitive tasks. In *Proceedings of ICLR*.

[Sunehag et al. 2018] Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W. M.; Zambaldi, V. F.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J. Z.; Tuyls, K.; and Graepel, T. 2018. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of AAMAS*, 2085–2087.

[Sutton and Barto 2018] Sutton, R. S., and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

[Tampuu et al. 2017] Tampuu, A.; Matiisen, T.; Kodelja, D.; Kuzovkin, I.; Korjus, K.; Aru, J.; Aru, J.; and Vicente, R. 2017. Multiagent cooperation and competition with deep reinforcement learning. *PLOS ONE* 12:1–15.

[Wang et al. 2017] Wang, Z.; Bapst, V.; Heess, N.; Mnih, V.; Munos, R.; Kavukcuoglu, K.; and de Freitas, N. 2017. Sample efficient actor-critic with experience replay. In *Proceedings of ICLR*.

[Wattenberg, Viégas, and Johnson 2016] Wattenberg, M.; Viégas, F.; and Johnson, I. 2016. How to use t-sne effectively. *Distill* 1(10):e2.

[Wu and Tian 2017] Wu, Y., and Tian, Y. 2017. Training agent for first-person shooter game with actor-critic curriculum learning. In *Proceedings of ICLR*.

[Xu et al. 2019] Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How powerful are graph neural networks? In *Proceedings of ICLR*.

[Yang et al. 2018a] Yang, Y.; Luo, R.; Li, M.; Zhou, M.; Zhang, W.; and Wang, J. 2018a. Mean field multi-agent reinforcement learning. In *Proceedings of ICML*, volume 80, 5571–5580.

[Yang et al. 2018b] Yang, Y.; Yu, L.; Bai, Y.; Wen, Y.; Zhang, W.; and Wang, J. 2018b. A study of AI population dynamics with million-agent reinforcement learning. In *Proceedings of AAMAS*, 2133–2135.

[Zheng et al. 2018] Zheng, L.; Yang, J.; Cai, H.; Zhou, M.; Zhang, W.; Wang, J.; and Yu, Y. 2018. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In *Proceedings of AAAI*.

Supplementary Materials

Experimental Description

StarCraft II In StarCraft II, we follow the settings of previous works (Rashid et al. 2018; Samvelyan et al. 2019). The local observation of each agent is drawn within their field of view, which encompasses the circular area of the map surrounding units and has a radius equal to the sight range. The input vector of each agent consists of the following features for all units in its observation range (both teammates and enemy): distance, relative x, relative y, and the unit type. We design the curriculum that each agent first learns on a 5 Immortals vs 5 Immortals battlefield for $1.5e+6$ steps, then learns on a 10 Immortals vs 10 Immortals battlefield for $1.5e+6$ steps, and learns on the target task of a 15 Immortals vs 15 Immortals battlefield for $1.5e+6$ steps. We add $1.5e+6$ training steps for IQL in last two tasks since it is more difficult to learn in such large-scale multiagent settings without considering the coexistence of other agents.

MAgent We following the settings of previous work on MAgent (Zheng et al. 2018), the action space includes 13 move actions, each of which will leads to a corresponding direction; and 8 attack actions, each of which attacks a corresponding grid unit (see Figure 8). The observation range for each agent is a 13×13 range of grids. The input vector of each agent includes the position and the Hit Point (HP) of the agent; the relative position of teammates and enemies, which is represented as a one-hot vector; the Hit Points of teammates and enemies, the number of teammates and enemies; the action, reward and normalized position of the agent at previous step. We design the curriculum that each agent learns the sequence of tasks as follows: learning on a 10 agents vs 10 agents battlefield for 7500 steps; learning on a 20 agents vs 20 agents battlefield for 4500 steps; learning on a 30 agents vs 30 agents battlefield for 1500 steps; learning on a 40 agents vs 40 agents battlefield for 750 steps; learning the target task of a 50 agents vs 50 agents battlefield for $1e+4$ steps.

An illustration of DyAN for StarCraft II is shown in Figure 9. For the vanilla network which does not contain the right part of DyAN, it contains a fully-connected layer with 64 units, following a GRU layer with 64 units, and then an output layer that outputs the state-action values of each action. Our DyAN divides the observation of each agent into two parts, the environmental information $o_t^{1,env}$ and itself information m_t^1 are input to a fully-connected layer with 64 units; the rest of information is input to the right part of DyAN. Specifically, we separate the observation for its teammates ($o_t^{1,2}, o_t^{1,3}$) and enemies ($o_t^{1,4}, o_t^{1,5}, o_t^{1,6}$), and input to two fully-connected layers with 64 units respectively, each of which follows an aggregation function, then all three parts are concatenated together and input to a GRU layer with 64 units, the output layer is a fully-connected layer that outputs the state-action values of each action.

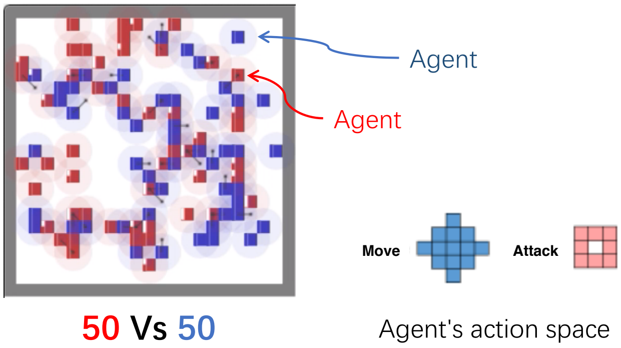


Figure 8: An illustration of action range on MAgent.

Network Structure

StarCraft II

MAgent The network structure of DyAN for MAgent is similar to that for StarCraft II, except that the unit size for each neural network layer is 16. The output can be either state-action values for each action, or the probability of choosing each action through a SOFTMAX activation. For actor-critic approaches, e.g., A2C (Mnih et al. 2016), it also outputs the state-values.

Parameter Settings

Here we provide the hyperparameters for StarCraft II and MAgent.

Table 2: Hyperparameters used for StarCraft II.

Hyperparameter	Value
Batch-size	32
Replay memory size	5000
Discount factor(γ)	0.99
Optimizer	RMSProp
Learning rate	$5e-4$
α	0.99
e	$1e-5$
Gradient-norm-clip	10
Action-selector	ϵ-greedy
ϵ-start	1.0
ϵ-finish	0.05
ϵ-anneal-time	50000 step
target-update-interval	200

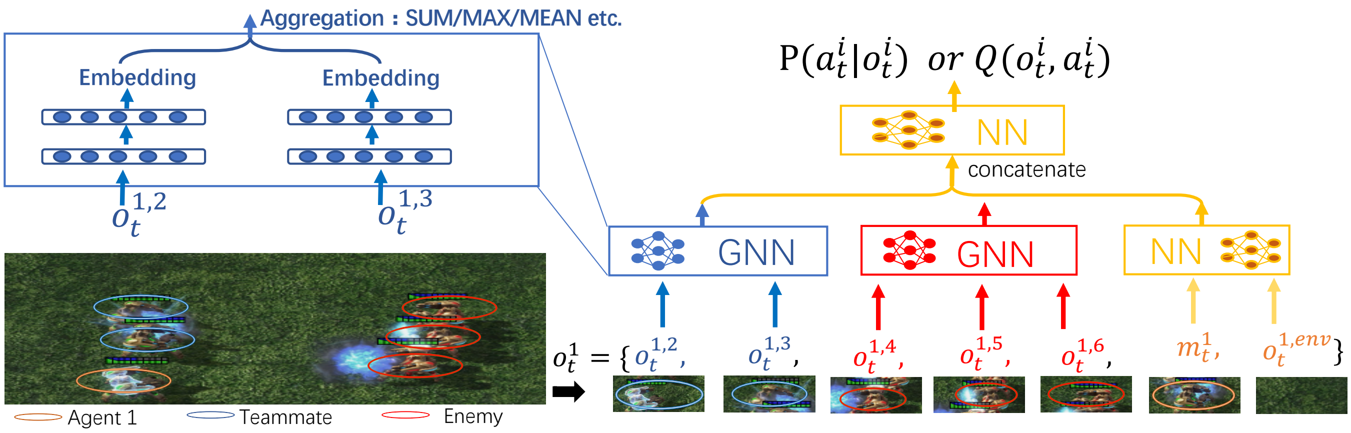

Aggregation : SUM/MAX/MEAN etc.
Embedding
Embedding
$o_t^{1,2}$
$o_t^{1,3}$
Agent 1
Teammate
Enemy
$P(a_t^i|o_t^i)$ or $Q(o_t^i, a_t^i)$
NN
concatenate
GNN
GNN
NN
$o_t^1 = \{o_t^{1,2}, o_t^{1,3}, o_t^{1,4}, o_t^{1,5}, o_t^{1,6}, m_t^1, o_t^{1,env}\}$

Figure 9: An illustration of the network structure of DyAN for StarCraft II.

Table 3: IQL hyperparameters used for MAgent.

Hyperparameter	Value
Batch-size	32
Replay memory size	100000
Replay memory size at the start of training	5000
Discount factor(γ)	0.98
Optimizer	Adam
Learning rate	$1e-4$
e	$1e-8$
Action-selector	ϵ-greedy
ϵ-start	1.0
ϵ-finish	0.01
ϵ-anneal-time	99 episodes
target-update-interval	20

Table 4: A2C hyperparameters used for MAgent.

Hyperparameter	Value
Training interval (T horizon)	20 step
Discount factor(γ)	0.98
Optimizer	Adam
Learning rate	$1e-3$
e	$1e-8$
Entropy term coefficient	0.1
Value loss coefficient	1
Actor loss coefficient	1

Table 5: PPO hyperparameters used for MAgent.

Hyperparameter	Value
Training interval (T horizon)	10 step
Discount factor(γ)	0.98
Clip hyperparameter ϵ	0.2
GAE λ	0.95
Optimizer	Adam
Learning rate	$2e-3$
e	$1e-8$
Entropy term coefficient	1e-3
Value loss coefficient	1
Actor loss coefficient	1

Table 6: ACER hyperparameters used for MAgent.

Hyperparameter	Value
Batch-size	64
Replay memory size	100000
Replay memory size at the start of training	100
Discount factor(γ)	0.98
Training interval (T horizon)	10 step
Discount factor(γ)	0.98
Truncating importance sampling ratio c	1.0
Optimizer	Adam
Learning rate	$1e-2$
e	$1e-8$
Entropy term coefficient	1e-2
Value loss coefficient	1
Bias correction term	1
Actor loss coefficient	1