

# Graph Convolutional Reinforcement Learning for Multi-Agent Cooperation

**Jiechuan Jiang**

Peking University

jiechuan.jiang@pku.edu.cn

**Chen Dun**

Macalester College

cdun@macalester.edu

**Zongqing Lu\***

Peking University

zongqing.lu@pku.edu.cn

## Abstract

Learning to cooperate is crucially important in multi-agent reinforcement learning. The key is to take the influence of other agents into consideration when performing distributed decision making. However, multi-agent environment is highly dynamic, which makes it hard to learn abstract representations of influences between agents by only low-order features that existing methods exploit. In this paper, we propose a graph convolutional model for multi-agent cooperation. The graph convolution architecture adapts to the dynamics of the underlying graph of the multi-agent environment, where the influence among agents is captured by their abstract relation representations. High-order features extracted by relation kernels of convolutional layers from gradually increased receptive fields are exploited to learn cooperative strategies. The gradient of an agent not only backpropagates to itself but also to other agents in its receptive fields to reinforce the learned cooperative strategies. Moreover, the relation representations are temporally regularized to make the cooperation more consistent. Empirically, we show that our model enables agents to develop more cooperative and sophisticated strategies than existing methods in jungle and battle games and routing in packet switching networks.

## Introduction

Cooperation is a widespread phenomenon in nature from viruses, bacteria, and social amoebae to insect societies, social animals, and humans (Melis and Semmann 2010). Human exceeds all other species in terms of the range and scale of cooperation. Therefore, for general artificial intelligence, it is crucially important to learn how to cooperate in multi-agent environments.

Deep reinforcement learning (RL) has shown human level performance in games such as Atari games (Mnih et al. 2015) and GO (Silver et al. 2017). However, motivated by self-interest only, individual agents usually ignore the benefit of other agents, causing the damage to the common goal in multi-agent environments. Moreover, independent RL treats other agents as part of environment and thus the environment becomes unstable from the perspective of individual agents as the strategies of other agents change during training. In addition, the learned policy can easily overfit to the policies of other agents (Lanctot et al. 2017), failing to sufficiently generalize during execution.

\*Corresponding author.

The key to multi-agent reinforcement learning (MARL) is taking the influence of other agents into consideration when performing distributed decision making. Currently, most MARL algorithms are designed based on this intuition. MADDPG (Lowe et al. 2017) directly trains a centralized critic that receives the observations and actions of all agents. Algorithms based on communication, such as CommNet (Sukhbaatar, Fergus, and others 2016), BiCNet (Peng et al. 2017), and ATOC (Jiang and Lu 2018), convey encoded observations and action intentions by information sharing among agents. Mean field (Yang et al. 2018) represents the influence by the mean action of neighboring agents. However, due to the highly dynamic multi-agent environments, existing methods that only explore low-order features fail to learn abstract representations of influences between agents, restraining the range and scale of cooperation. That is to say, existing methods are insufficient to make agents understand their mutual interplay and form a relatively broad view of the environment.

In this paper, we capture the influence between agents by their relation. The intuition is that individuals in games or social networks are related to each other, and agents and their relations can be represented by a graph. Unlike low-dimensional regular grids such as images, the agent graph is irregular and dynamic, even lying on non-Euclidean domains, which makes it challenging to extract features. Inspired by the convolution in the domains, such as social networks (Kipf and Welling 2017), protein structure (Duvenaud et al. 2015) and 3D point cloud (Charles et al. 2017), we apply convolution operations to the graph of agents for cooperative tasks, where each agent is a node, each node connects to its neighbors, and the local observation of agent is the attributes of node. By using multi-head attention (Vaswani et al. 2017) as the convolution kernel, graph convolution is able to extract relation representations, and features from neighboring nodes can be integrated just like the receptive field of a neuron in a normal convolutional neural network (CNN). High-order features extracted from gradually increased receptive fields are exploited to learn cooperative strategies. The gradient of an agent not only backpropagates to itself but also to other agents in its receptive fields to reinforce the learned cooperative strategies. Moreover, the relation representations are temporally regularized to make cooperation more consistent.

Our graph convolutional model, called DGN, is instantiated as an extension of deep  $Q$  network and trained end-to-end, adopting the paradigm of centralized training and distributed execution. DGN abstracts the influence between agents by relation kernels, extracts latent features by convolution, and induces consistent cooperation by temporal relation regularization. Moreover, as DGN shares weights among all agents, it is easy to scale, better suited in large-scale MARL. We empirically show the learning effectiveness of DGN in jungle and battle games and routing in packet switching networks. It is demonstrated DGN agents are able to develop more cooperative and sophisticated strategies than existing methods. To the best of our knowledge, this is the first time that graph convolution is successfully applied to MARL.

## Related Work

MADDPG (Lowe et al. 2017) and COMA (Foerster et al. 2018) are the extension of actor-critic model for multi-agent environments, where MADDPG is designed for mixed cooperative-competitive environments and COMA is proposed to solve multi-agent credit assignment in cooperative settings. A centralized critic that takes as input the observations and actions of all agents are used in MADDPG and COMA. Zhang et al. consider networked critics that are updated via communication. However, all these three models have to train an independent policy network for each agent, which tends to learn a policy specializing specific tasks and easily overfits to the number of agents.

There are several models that have been proposed to learn multi-agent cooperation by communication. These models are end-to-end trainable by backpropagation. CommNet (Sukhbaatar, Fergus, and others 2016) uses continuous communication for full cooperation tasks. At a single communication step, each agent sends its hidden state as the message to the communication channel and then the averaged message from other agents is fed into the next layer. BiCNet (Peng et al. 2017) uses a recurrent neural network (RNN) as the communication channel to connect each individual agent’s policy and value networks. ATOC (Jiang and Lu 2018) enables agents to learn dynamic communication with nearby agents using attention mechanism. The bidirectional-LSTM integrates the hidden thoughts of agents participating in a communication group and produces new thoughts for distributed decision making. These communication models prove that the information sharing does helps, and as we will show later, they can be considered as special instances of our model.

Most existing models are limited to the scale of dozens of agents, while some consider large-scale MARL. When the number of agents increases, learning becomes hard due to the curse of the dimensionality and the exponential growth of agent interactions. Instead of considering the different effects of other individuals on each agent, Mean Field (Yang et al. 2018) approximates the effect of other individuals by their mean action. However, the mean action eliminates the difference among these agents in terms of observation and action and thus incurs the loss of important information that helps cooperative decision making.

## Background

### Graph Convolutional Networks

Many important real-world applications come in the form of graphs, such as social networks, protein-interaction networks, and 3D point cloud. In the last couple of years, several frameworks (Henaff, Bruna, and LeCun 2015; Niepert, Ahmed, and Kutzkov 2016; Kipf and Welling 2017; Velickovic et al. 2017) have been architected to extract locally connected features from arbitrary graphs. Typically, the goal is to learn a function of features on graphs. A graph convolutional network (GCN) takes as input the feature matrix that summarizes the attributes of each node and adjacency matrix outputs a node-level feature matrix. The function is similar to the convolution operation in CNNs, where the kernels are convolved across local regions of the input and produce the feature maps.

### Interaction Networks

Learning common sense knowledge is one of the keys to artificial intelligence. However, it has proven difficult for neural networks. Interaction networks aim to reason the objects, relations and physics in complex systems. Interaction networks predict the future states and underlying properties, which is similar to the way of human thinking. There are several frameworks have been proposed to model the interactions. IN (Battaglia et al. 2016) focuses on the binary relations between entities. The model computes the effect of interaction and predicts the next state by taking the interaction into consideration. VIN (Watters et al. 2017) predicts the future states from raw visual observations. VAIN (Hoshen 2017) models multi-agent relations and predicts the future states with attention mechanism.

### Relational Reinforcement Learning

The core idea of relational reinforcement learning (RRL) is to combine RL with relational learning by representing states and policies based on relations. Neural networks can operate on structured representations of a set of entities, non-locally compute interactions on a set of feature vectors, and perform relational reasoning via iterated message passing (Zambaldi et al. 2018). The relation block, multi-head dot-product attention (Vaswani et al. 2017), is embedded into neural networks to learn the pairwise interaction representation. High-order interactions between entities can be captured by stacking multiple blocks recurrently or deeply.

## Method

DGN is built on the concept of graph convolution. We construct the multi-agent environment as a graph, where agents in the environment are represented by the nodes of the graph, and for each node, there are  $K$  edges connected to its  $K$  nearest neighbors (*e.g.*, in terms of distance or other metrics, depending on the environment). The intuition behind this is nearer neighbors are more likely to interact with and affect each other. Moreover, in large-scale multi-agent environments, it is costly and less helpful to take all agents’ influence into consideration, because receiving a large amount of

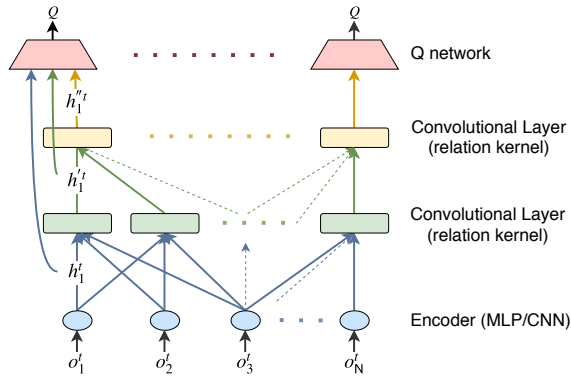


Figure 1: DGN architecture. All agents share weights and gradients are accumulated to update the weights.

information requires high bandwidth and incurs high computational complexity, and agents cannot differentiate valuable information from globally shared information (Jiang and Lu 2018). In addition, as convolution can gradually increase the receptive field of an agent, the scope of cooperation is not restricted. Therefore, it is efficient and effective to consider only  $K$  nearest neighbors. Unlike the static graph considered in GCNs, the graph of multi-agent environment is continuously changing over time as agents move or enter/leave the environment. Therefore, DGN should be capable to adapt the dynamics of the graph and learn as the multi-agent environment evolves.

### Graph Convolution Architecture

We consider the partially observable environment, where at each timestep  $t$  each agent  $i$  receives a local observation  $o_i^t$ , which is the property of node  $i$  in the graph, takes an action  $a_i^t$ , and gets a reward  $r_i^t$ . DGN consists of three types of modules: observation encoder, convolutional layer and  $Q$  network, as illustrated in Figure 1. The local observation  $o_i^t$  is encoded into a feature vector  $h_i^t$  by MLP for low-dimensional input or CNN for visual input. The convolutional layer integrates the feature vectors in the local region (including node  $i$  and its  $K$  neighbors) and generates the latent feature vector  $h_i^{t'}$ . By stacking more convolutional layers, the receptive field of an agent gradually grows, where more information is gathered, and thus the scope of cooperation can also increase. That is, by stacking one convolutional layer, node  $i$  can directly acquire feature vectors from encoders of the nodes in one-hop ( $K$  neighbors). By stacking two layers, node  $i$  can get the output of the first convolutional layer of the nodes in one hop, which contains the information from nodes in two hops. However, more convolutional layers will not increase the local region of node  $i$ , i.e., node  $i$  still only directly receives information from its  $K$  neighbors. This saliency is very important as we consider decentralized execution. Details of the convolution kernel will be explained in next section.

As the number and position of agents vary over time, the underlying graph continuously changes, which brings difficulties to graph convolution. To address the issue, we merge all agents' feature vectors at time  $t$  into a feature matrix  $F^t$

with size  $N \times L$  in the order of index, where  $N$  is the number of agents and  $L$  is the length of feature vector. Then, we construct an adjacency matrix  $C_i^t$  with size  $(K+1) \times N$  for agent  $i$ , where the first row is the one-hot representation of node  $i$ 's index, and the  $j$ th row,  $j = 2, \dots, K+1$ , is the one-hot representation of the index of the  $(j-1)$ th nearest neighbor. Then, we can obtain the feature vectors in the local region of node  $i$  by  $C_i^t \times F^t$ .

Inspired by DenseNet (Huang et al. 2017), for each agent, the features of all the preceding layers are merged and fed into the  $Q$  network, so as to assemble and reuse the observation representation and features from different receptive fields, which respectively have distinctive contributions to the strategy that takes the cooperation at different scopes into consideration. The  $Q$  network selects the action that maximizes the  $Q$ -value with a probability of  $1 - \epsilon$  or acts randomly with a probability of  $\epsilon$ . The gradient of  $Q$ -loss of each agent will backpropagate not only to itself and  $K$  neighbors but also to other agents in its receptive fields. That is to say, the agent not only focuses on maximizing its own expected reward but also considers how its policy affects other agents, and hence agents are enabled to learn cooperation. Moreover, each agent receives the encoding of observations and intentions of influential agents, which makes the environment more stable from the perspective of individual agent.

In DGN, all agents share weights, which significantly reduces the number of parameters. However, this does not prevent the emergence of complex cooperative strategies, as we will show in the experiments. We adopt the paradigm of centralized training and distributed execution. During training, at each timestep, we store the tuple  $(\mathcal{O}, \mathcal{A}, \mathcal{O}', \mathcal{R}, \mathcal{C})$  in the replay buffer  $\mathcal{B}$ , where  $\mathcal{O} = \{o_1, \dots, o_N\}$ ,  $\mathcal{A} = \{a_1, \dots, a_N\}$ ,  $\mathcal{O}' = \{o'_1, \dots, o'_N\}$ ,  $\mathcal{R} = \{r_1, \dots, r_N\}$ , and  $\mathcal{C} = \{C_1, \dots, C_N\}$ . Note that we drop time  $t$  in the notations for simplicity. Then, we sample a random minibatch of  $S$  samples from  $\mathcal{B}$  and minimize the loss

$$\mathcal{L}(\theta) = \frac{1}{S} \sum_s \frac{1}{N} \sum_{i=1}^N (y_i - Q(O_i, a_i; \theta))^2,$$

where

$$y_i = r_i + \gamma \max_{a'} Q(O'_i, a'_i; \theta'),$$

$O_i \subseteq \mathcal{O}$  denotes the set of observations of all the agents in  $i$ 's receptive fields,  $\gamma$  is the discount factor, and the model is parameterized by  $\theta$ . To make the learning process more stable, we keep  $\mathcal{C}$  unchanged in two successive timesteps when computing the  $Q$ -loss in training. The gradients of  $Q$ -loss of all agents are accumulated to update the parameters. Each agent not only minimizes its own  $Q$ -loss but also  $Q$ -loss of other agents who the agent collaborates with. Then, we softly update the target network as

$$\theta' = \beta \theta + (1 - \beta) \theta'.$$

During execution, all agents share the parameters and each agent only requires the information from its  $K$  neighbors (e.g., via communication), regardless of the number of agents. Therefore, our model can easily scale and thus is suitable for large-scale MARL.

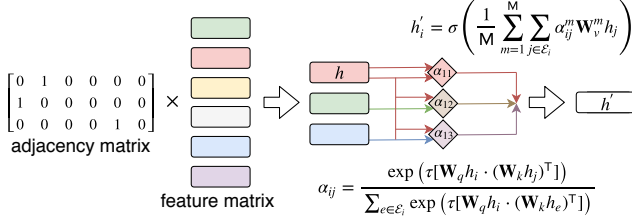


Figure 2: Illustration of computation of the convolutional layer with relation kernel.

## Relation Kernel

Convolution kernels integrate the information in the receptive field to extract the latent feature among agents. One of the most important properties is that the kernel should be independent from the order of the input feature vectors. Mean operation meets this requirement. If we take mean operation as the kernel, the model is the CommNet with graph convolution architecture. However, without learnable parameters, mean kernel leads to only slight performance gain. BiCNet uses the learnable kernel, RNN. However, the input order feature vectors of RNN severely impacts the performance, though the affect is alleviated by bi-direction mechanism.

Adopting the idea from RRL, we use multi-head dot-product attention as the kernel to compute interactions between entities. Unlike RRL, we take each agent rather than pixel as an entity. For each agent  $i$ , there are a set of entities  $\mathcal{E}_i$  ( $K$  neighbors and itself) in the local region. Then, we can compute the interaction between  $i$  and  $j \in \mathcal{E}_i$  for each independent attention head,

$$\alpha_{ij} = \frac{\exp(\tau[\mathbf{W}_q h_i \cdot (\mathbf{W}_k h_j)^T])}{\sum_{e \in \mathcal{E}_i} \exp(\tau[\mathbf{W}_q h_i \cdot (\mathbf{W}_k h_e)^T])},$$

where  $\tau$  is a scaling factor.  $\alpha$  can be considered as the relation representation between entities. The features of  $M$ -head attention are averaged and fed into function  $\sigma$  (one-layer MLP with ReLU non-linearities) to produce the output of the convolutional layer,

$$h'_i = \sigma \left( \frac{1}{M} \sum_{m=1}^M \sum_{j \in \mathcal{E}_i} \alpha_{ij}^m \mathbf{w}_v^m h_j \right).$$

Figure 2 illustrates the computation of the convolutional layer with relation kernel.

Multi-head attention extracts multiple representations of the relation between individual agents, which makes the kernel independent from the order of input feature vectors, and allows the model to jointly attend to information from different representation subspaces for different agents. Moreover, with multiple convolutional layers, high-order relation representations can be extracted, which effectively capture the influence of other agents and greatly help to make cooperative decision.

## Temporal Relation Regularization

As we train our model using deep  $Q$  learning, we use future value estimate as target for the current estimate. We follow this insight and apply it to the relation kernel in our model.

Intuitively, if the relation representation produced by the relation kernel of upper layer truly captures the abstract relation between surrounding agents and itself, such relation representation should be stable/consistent for at least a short period of time, even when the state/feature of surrounding agents changes. Since in our relation kernel, the relation is represented as the attention weight distribution to the state of surrounding agents, we use the attention weight distribution in the next state as the target for the current attention weight distribution to encourage the agent to form the consistent relation representation. As the relation in different states should not be the same but similar, we use KL divergence to compute the distance between the attention weight distributions in the two states.

It should be noted that we do not use the target network to produce the target relation representation as in normal deep  $Q$  learning. This is because relation representation is highly correlated with weights of feature extraction. But update of such weights in target network always lags behind that of the current network. Since we only focus on the self-consistent of the relation representation based on the current feature extraction network, we apply current network to the next state to produce the new relation representation instead of the target network as in deep  $Q$  learning.

Let  $\mathcal{G}^\kappa(O_i; \theta)$  denotes the attention weight distribution of relation representations at convolutional layer  $\kappa$  for agent  $i$ , where  $\mathcal{G}^\kappa(O_i; \theta)_j = \alpha_{ij}^\kappa$ . Therefore, with temporal relation regularization, the loss is modified as below

$$\mathcal{L}(\theta) = \frac{1}{S} \sum_S \frac{1}{N} \sum_{i=1}^N ((y_i - Q(O_i, a_i; \theta))^2 + \lambda D_{\text{KL}}(\mathcal{G}^\kappa(O_i; \theta) || z_i),$$

where  $z_i = \mathcal{G}^\kappa(O'_i; \theta)$  and  $\lambda$  is the coefficient for the regularization loss.

Temporal relation regularization of upper layer in DGN helps the agent to form long-term and consistent action policy in the highly dynamical environment with a lot of moving agents. This will further help agents to form cooperative behavior since many cooperation tasks need long-term consistent actions of the collaborated agents to get the final reward. We will further analyze this in the experiments.

## Experiments

For the experiments, we adopt a large-scale gridworld platform MAgent (Zheng et al. 2017). In the environment, each agent corresponds to one grid and has a local observation that contains a square view with  $11 \times 11$  grids centered at the agent and its own coordinates. The discrete actions are moving or attacking. Two scenarios, *jungle* and *battle*, are considered to investigate the cooperation among agents. Also, we build an environment, *routing*, that simulates the routing in packet switching networks to verify the applicability of our model in read-world applications. These three scenarios are illustrated in Figure 3. The hyperparameters of DGN in the three scenarios are summarized in Table 1. In the experiments, we compare DGN with DQN, CommNet, and Mean Field  $Q$ -learning (MFQ). For fair comparison, all



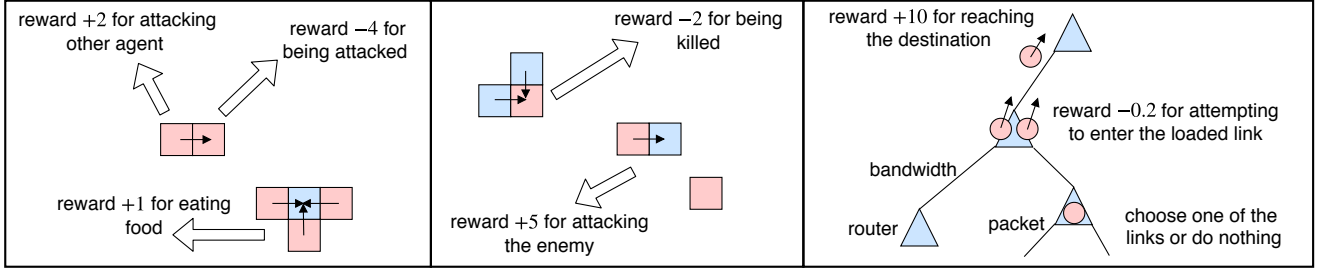


Figure 3: Illustration of experimental scenarios: *jungle* (left), *battle* (mid), *routing* (right).

Table 1: Hyperparameters

Hyperparameter	Jungle	Battle	Routing
discount ( $\gamma$ )	0.96	0.96	0.98
batch size		10	
buffer capacity		$2 \times 10^5$	
$\beta$		0.01	
$\epsilon$ and decay		0.6/0.996	
optimizer		Adam	
learning rate		$10^{-4}$	
# neighbors (K)		3	
# convolutional layers		2	
# attention heads		8	
$\tau$		0.25	
$\lambda$		0.03	
$\kappa$		2	
# encoder MLP layers		2	
# encoder MLP units		(512, 128)	
MLP activation		ReLU	
initializer		random normal	

Table 2: Jungle

(N, L)		DGN	DGN-M	MFQ	CommNet	DQN
(20, 12)	mean reward	<b>0.70</b>	0.66	0.62	0.30	0.24
	# attacks	<b>0.91</b>	1.89	2.74	5.44	7.35
(50, 12)	mean reward	<b>0.67</b>	0.63	0.57	0.27	0.20
	# attacks	<b>0.91</b>	1.88	3.13	6.35	9.02

the models have similar parameter scale. MADDPG is not considered as a baseline, because training an independent policy network for each agent makes MADDPG infeasible in large-scale scenarios. More importantly, most real-world applications are open systems, *i.e.*, agents come and go as in battle, and thus it is impossible to train a model for every new agent. Please refer to the video at <https://goo.gl/AFV9qi> for more details about the experiments, and the code of DGN is available at <https://github.com/PKU-AI-Edge/GraphConv4MARL.git/>.

## Jungle

This scenario is a moral dilemma. There are  $N$  agents and  $L$  foods in the field, where foods are fixed. An agent gets positive reward by eating food, but gets higher reward by attacking other agent. At each timestep, each agent can move to or attack one of four neighboring grids. The reward is 0 for moving, +1 for attacking (eating) the food, +2 for attacking other agent, -4 for being attacked, and -0.01 for attacking a blank grid (inhibiting excessive attacks). This experiment is to examine whether agents can learn the strategy of collaboratively sharing resources rather than attacking each other.

We trained all the models with the setting of  $N = 20$  and  $L = 12$  for 2000 episodes. Figure 4a shows their learning curves, where DGN-M is graph convolution with mean kernel, and each model is with three training runs. Ta-

ble 2 shows the mean reward (averaged over all agents and timesteps) and number of attacks between agents (averaged over all agents) over 30 test runs, each game unrolled with 120 timesteps.

DGN outperforms all the baselines during training and test in terms of mean reward and number of attacks between agents. It is observed that DGN agents can properly select the close food and seldom hurt each other, and the food can be allocated rationally by the surrounding agents, as shown in Figure 5a. Moreover, attacks between DGN agents are much less than others, *i.e.*,  $2\times$  and  $3\times$  less than DGN-M and MFQ, respectively. Sneak attack, fierce conflict, and hesitation are the characteristics of CommNet and DQN agents, as illustrated in Figure 5b, verifying their failure of learning cooperation. Although DGN-M and CommNet both use mean operation, DGN-M greatly outperforms CommNet. This is attributed to the graph convolution that can effectively extract latent features from surrounding agents. Moreover, comparing DGN with DGN-M, we can conclude that the relation kernel that abstracts the relation representation between agents does help to learn cooperative strategy.

We directly apply the trained model with  $N = 20$  and  $L = 12$  to the scenario of  $N = 50$  and  $L = 12$ . Higher agent density and food shortage make the moral dilemma more complicated. The slight drop of mean reward of all the models is because food is not enough to supply each agent. DGN maintains the number of attacks, which means agents who cannot obtain the food will stay in a waiting state without attacking other. However, agents of MFQ, CommNet, and DQN attack each other more frequently when there are more agents sharing food.

## Battle

This scenario is a fully cooperative task, where  $N$  agents learn to fight against  $L$  enemies who have superior abilities than the agents. The agent's moving or attacking range is the four neighbor grids, however, the enemy can move to one of twelve nearest grids or attack one of eight neighbor grids. Each agent/enemy has six hit points (*i.e.*, being

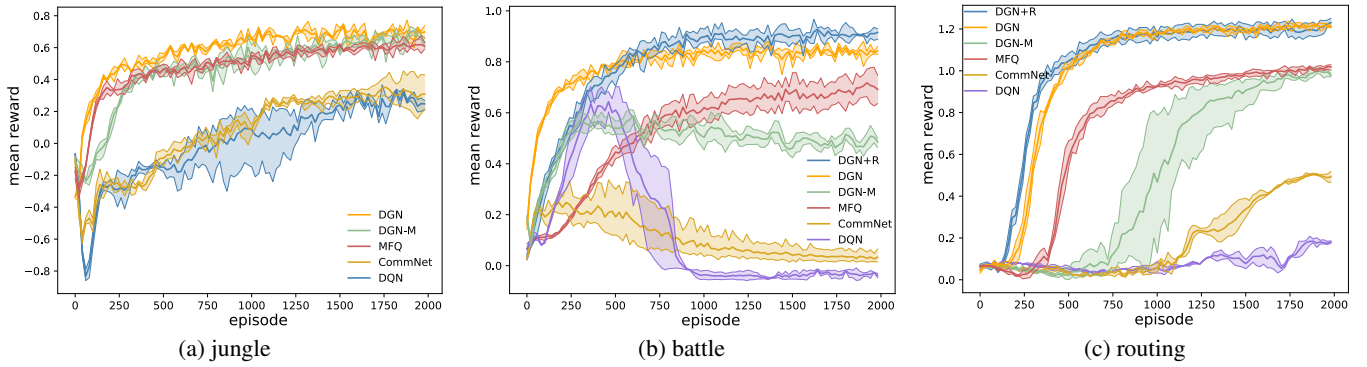


Figure 4: Learning curves in terms of mean reward in jungle, battle, and routing.

killed by six attacks). After the death of an agent/enemy, a new agent/enemy will be added at a random location. The reward is +5 for attacking the enemy, -2 for being killed, and -0.01 for attacking a blank grid. The pretrained DQN model built-in MAgent takes the role of enemy. As individual enemy is much powerful than individual agent, an agent has to collaborate with others to develop coordinated tactics to fight enemies. Moreover, as the hit point of enemy is six, agents have to continuously cooperate to kill the enemy. Therefore, the task is much more challenging than jungle in terms of learning to cooperate.

We trained all the models with the setting of  $N = 20$  and  $L = 12$  for 2000 episodes. Figure 4b shows the learning curves of all the models in terms of mean reward. DGN converges to much higher mean reward than other baselines, and its learning curve is more stable. For CommNet and DQN, they first get relative high reward, but they eventually converge to much lower reward than others. As observed in the experiment, at the beginning of training, DQN and CommNet learn sub-optimum policies such as gathering as a group in a corner to avoid being attacked, since such behaviors generate relatively high reward. However, since the distribution of reward is uneven, *i.e.*, agents at the exterior of the group are easily attacked, learning from the “low reward experiences” produced by the sub-optimum policy, DQN and CommNet converge to more passive policies, which lead to much lower reward. We evaluate DGN and the baselines by running 30 test games, each game unrolled with 300 timesteps. Table 3 shows the mean reward, kills, deaths, and kill-death ratio.

DGN agents learn a series of tactical maneuvers, such as

Table 3: Battle

	DGN+R	DGN	DGN-M	MFQ	CommNet	DQN
mean reward	<b>0.91</b>	<b>0.84</b>	0.50	0.70	0.03	-0.03
# kills	<b>220</b>	<b>208</b>	121	193	7	2
# deaths	<b>97</b>	<b>101</b>	84	92	27	74
kill-death ratio	<b>2.27</b>	<b>2.06</b>	1.44	2.09	0.26	0.03

encircling and envelopment of a single flank. For single enemy, DGN agents learn to encircle and attack it together. For a group of enemies, DGN agents learn to move against and attack one of the enemy’s open flanks, as depicted in Figure 5c. CommNet agents adopt an active defense strategy. They seldom launch an attack but rather run away or gather together to avoid being attacked. DQN agents driven by self-interest fail to learn a rational policy. They are usually forced into a corner and passively react to the enemy’s attack, as shown in Figure 5d. MFQ agents do not effectively cooperate with each other since there is no gradient backpropagated among agents to reinforce the cooperation during MFQ training.

In DGN, relation kernels can extract the interaction and influence among agents from gradually increased receptive fields, which can be easily exploited to yield cooperation. Moreover, the gradient backpropagation from an agent to other agents in the receptive field enforces the cooperation. Therefore, DGN outperforms other baselines.

DGN with temporal relation regularization, *i.e.*, DGN+R, achieves consistently better performance compared to DGN as shown in Figure 4b and Table 3. In the experiment, it is observed that DGN+R agents indeed behave more con-

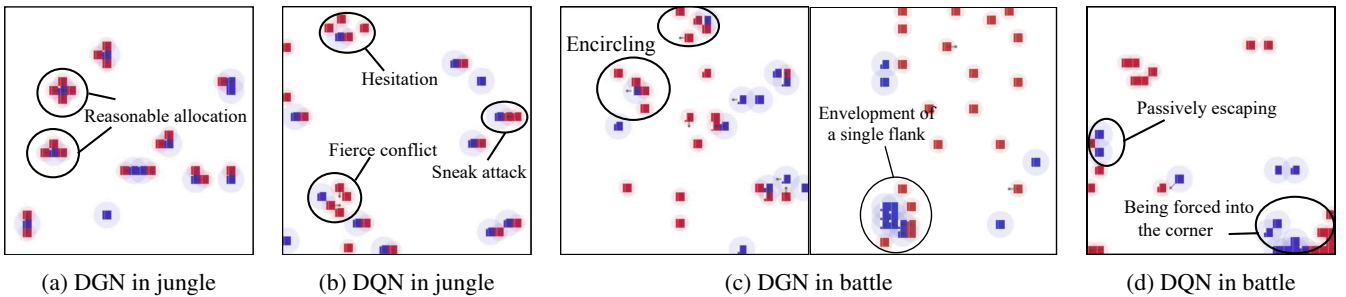


Figure 5: Illustration of representative behaviors of DGN and DQN agents in jungle and battle.

Table 4: Routing

(N, L)		Floyd	Floyd with BL	DGN+R	DGN	DGN-M	MFQ	CommNet	DQN
(20, 20)	mean reward			<b>1.23</b>	<b>1.21</b>	0.99	1.02	0.49	0.18
	delay	6.3	8.7	<b>8.0</b>	<b>8.1</b>	9.8	9.4	18.6	46.7
	# delivered packets	317	230	<b>250</b>	<b>247</b>	204	213	108	43
(40, 20)	mean reward			<b>0.94</b>	<b>0.90</b>	0.78	0.76	0.35	0.05
	delay	6.3	13.7	<b>10.2</b>	<b>10.5</b>	12.2	12.8	21.2	112.2
	# delivered packets	634	291	<b>392</b>	<b>381</b>	327	312	186	35

sistently and synchronously with each other, while DGN agents are more likely to be distracted by the new appearance of enemy or friend nearby and abandon its original intended trajectory. This results in fewer appearances of successful formation of encircling of a moving enemy, which might need consistent cooperation of agents to move across the field. DGN+R agents often overcome such distraction and show more long-term strategy and aim by moving more synchronously to chase the enemy until encircle and destroy it. From this experiment, we can see that temporal relation regularization indeed helps agents to form more consistent cooperation.

## Routing

This scenario is an abstraction of routing in packet switching networks, where the routing protocol tries to optimize the mean delay of data packets by making distributed decision at each router (*i.e.*, by determining only the next hop of a packet at a router). The network consists of  $L$  routers. Each router is randomly connected to a constant number of routers (three in the experiment), and the network topology is stationary. The bandwidth of each link is the same and set to 1. There are  $N$  data packets with a random size between 0 and 1, and each packet is randomly assigned a source and destination router. If there are multiple packets with the sum size larger than 1, they cannot go through a link simultaneously.

In the experiment, data packets are agents, and they aim to quickly reach the destination while avoiding congestion. At each timestep, the observation of a packet is its own attributes (*i.e.*, current location, destination, and data size), the attributes of cables connected to its current location (*i.e.*, load, length), and neighboring data packets (on the connected cable or routers). It takes some timesteps for a data packet to go through a cable, a linear function of the cable length. The action space of a packet is the choices of next hop. If the link to the selected next hop is overloaded, the data packet will stay at the current router and be punished with a reward  $-0.2$ . Once the data packet arrives at the destination, it leaves the system and gets a reward  $+10$  and another data packet enters the system with random initialization.

We trained all the models with the setting of  $N = 20$  and  $L = 20$  for 2000 episodes. Figure 4c shows the learning curves in terms of mean reward. DGN and DGN+R converge to much higher mean reward and more quickly than the baselines. DGN-M and MFQ have similar mean reward at the end, though MFQ converges faster than DGN-M. As

expected, DQN performs the worst, which is much lower than others.

We evaluate all the models by running 30 test games, each game unrolled with 100 timesteps. Table 4 shows the mean reward, mean delay of data packets, and number of delivered packets, where the delay of a packet is measured by the timesteps taken by the packet from source to destination. To better interpret the performance of the models, we calculate the shortest path for every pair of nodes in the network using Floyd algorithm. Then, during test, we directly calculate the mean delay based on the shortest path of each packet, which is 6.3 (Floyd in Table 4). Note that this delay is without considering the bandwidth limitation (*i.e.*, data packets can go through any link simultaneously). Thus, this is the ideal case for the routing problem. When considering the bandwidth limit, we let each packet follow its shortest path, and if a link is congested, the packet will wait at the router until the link is unblocked. The resulted delay is 8.7 (Floyd with BL in Table 4), which can be considered as the practical solution.

As shown in Table 4, the performance of DGN-M, MFQ, CommNet, and DQN are worse than Floyd with BL. However, the delay and number of delivered packets of DGN are much better than other models and also better than Floyd with BL. In the experiment, it is observed that DGN agents tend to select the shortest path to the destination, and more interestingly, learn to select different paths when congestion is about to occur. DQN agents cannot learn the shortest path due to myopia and easily cause congestion at some links without considering the influence of other agents. Information sharing indeed helps as DGN-M, MFQ, and CommNet all outperform DQN. However, they are unable to develop the sophisticated routing protocol as DGN does. DGN+R has slightly better performance than DGN. This is because data packets with different destinations seldom cooperate continuously (sharing many links) along their paths.

To investigate how the traffic pattern affects the performance of the models, we perform the experiments with  $N = 40$  and  $L = 20$ , *i.e.*, heavier data traffic, where all the models are retrained. From Table 4, we can see that DGN+R and DGN outperform other models and Floyd with BL. Under heavier traffic, DGN+R and DGN are much better than Floyd with BL, and DGN-M and MFQ are also better than Floyd with BL. The reason is that the strategy of Floyd with BL (*i.e.*, simply following the shortest path) is favorable when traffic is light and congestion is rare, while this does not work well when traffic is heavy and congestion easily occurs. The performance of DGN+R and DGN under var-

ious traffic patterns demonstrates the learning effectiveness of our model.

## Conclusions

We have proposed a graph convolutional model for multi-agent cooperation. DGN adapts the dynamics of the underlying graph of multi-agent environment and exploits convolution with relation kernels to extract relation representations from gradually increased receptive fields. Different orders of abstract relations are exploited to learn cooperative strategies. The gradient of an agent not only backpropagates to itself but also to other agents in its receptive fields to reinforce the learned cooperative strategies. Moreover, the relation representations are temporally regularized to make the cooperation more consistent. Empirically, DGN outperforms existing methods in a variety of cooperative multi-agent environments.

## Acknowledges

This work was supported in part by Peng Cheng Laboratory and NSFC under grant 61872009. We thank Zhehan Fu for sharing the GPUs for our experiments.

## References

- [Battaglia et al. 2016] Battaglia, P.; Pascanu, R.; Lai, M.; Rezende, D. J.; et al. 2016. Interaction networks for learning about objects, relations and physics. In *NIPS’16*, 4502–4510.
- [Charles et al. 2017] Charles, R. Q.; Su, H.; Kaichun, M.; and Guibas, L. J. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR’17*, 77–85.
- [Duvenaud et al. 2015] Duvenaud, D. K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS’15*, 2224–2232.
- [Foerster et al. 2018] Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; and Whiteson, S. 2018. Counterfactual multi-agent policy gradients. In *AAAI’18*.
- [Henaff, Bruna, and LeCun 2015] Henaff, M.; Bruna, J.; and LeCun, Y. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*.
- [Hoshen 2017] Hoshen, Y. 2017. Vain: Attentional multi-agent predictive modeling. In *NIPS’17*, 2701–2711.
- [Huang et al. 2017] Huang, G.; Liu, Z.; van der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *CVPR’17*, 2261–2269.
- [Jiang and Lu 2018] Jiang, J., and Lu, Z. 2018. Learning attentional communication for multi-agent cooperation. *NIPS’18*.
- [Kipf and Welling 2017] Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR’17*.
- [Lancot et al. 2017] Lancot, M.; Zambaldi, V.; Gruslys, A.; Lazaridou, A.; Perolat, J.; Silver, D.; and Graepel, T. 2017. A unified game-theoretic approach to multiagent reinforcement learning. In *NIPS’17*, 4193–4206.
- [Lowe et al. 2017] Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, O. P.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NIPS’17*, 6379–6390.
- [Melis and Semmann 2010] Melis, A. P., and Semmann, D. 2010. How is human cooperation different? *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 365(1553):2663–2674.
- [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.
- [Niepert, Ahmed, and Kutzkov 2016] Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *ICML’16*, 2014–2023.
- [Peng et al. 2017] Peng, P.; Wen, Y.; Yang, Y.; Yuan, Q.; Tang, Z.; Long, H.; and Wang, J. 2017. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*.
- [Silver et al. 2017] Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354.
- [Sukhbaatar, Fergus, and others 2016] Sukhbaatar, S.; Fergus, R.; et al. 2016. Learning multiagent communication with backpropagation. In *NIPS’16*, 2244–2252.
- [Vaswani et al. 2017] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NIPS’17*, 5998–6008.
- [Velickovic et al. 2017] Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.
- [Watters et al. 2017] Watters, N.; Tacchetti, A.; Weber, T.; Pascanu, R.; Battaglia, P.; and Zoran, D. 2017. Visual interaction networks. *arXiv preprint arXiv:1706.01433*.
- [Yang et al. 2018] Yang, Y.; Luo, R.; Li, M.; Zhou, M.; Zhang, W.; and Wang, J. 2018. Mean field multi-agent reinforcement learning. In *ICML’18*, 5571–5580.
- [Zambaldi et al. 2018] Zambaldi, V.; Raposo, D.; Santoro, A.; Bapst, V.; Li, Y.; Babuschkin, I.; Tuyls, K.; Reichert, D.; Lillicrap, T.; Lockhart, E.; et al. 2018. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*.
- [Zhang et al. 2018] Zhang, K.; Yang, Z.; Liu, H.; Zhang, T.; and Başar, T. 2018. Fully decentralized multi-agent reinforcement learning with networked agents. In *ICML’18*.
- [Zheng et al. 2017] Zheng, L.; Yang, J.; Cai, H.; Zhang, W.; Wang, J.; and Yu, Y. 2017. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. *arXiv preprint arXiv:1712.00600*.