Graph Convolutional Reinforcement Learning for Multi-Agent Cooperation

Jiechuan Jiang [1] **Chen Dun** [2] **Zongqing Lu** [1]

Abstract

Learning to cooperate is crucially important in multi-agent environments. The key is to understand the mutual interplay between agents. However, multi-agent environments are highly dynamic, which makes it hard to learn abstract representations of their mutual interplay. In this paper, we propose graph convolutional reinforcement learning for multi-agent cooperation, where graph convolution adapts to the dynamics of the underlying graph of the multi-agent environment, and relation kernels capture the interplay between agents by their relation representations. Latent features produced by convolutional layers from gradually increased receptive fields are exploited to learn cooperation, and the cooperation is further boosted by temporal relation regularization for consistency. Empirically, we show that our method substantially outperforms existing methods in a variety of cooperative scenarios.

1. Introduction

Cooperation is a widespread phenomenon in nature from viruses, bacteria, and social amoebae to insect societies, social animals, and humans (Melis & Semmann, 2010). Human exceeds all other species in terms of the range and scale of cooperation. The development of human cooperation is facilitated by the underlying graph of human societies (Ohtsuki et al., 2006; Apicella et al., 2012), where the mutual interplay between humans is abstracted by their relations.

It is crucially important to enable agent to learn to cooperate in multi-agent environments for many applications, *e.g.*, autonomous driving (Shalev-Shwartz et al., 2016), traffic light control (Wiering, 2000), smart grid control (Yang et al., 2018a), and multi-robot control (Matignon et al., 2012). Multi-agent reinforcement learning (MARL) facilitated by communication (Sukhbaatar et al., 2016; Peng et al., 2017; Jiang & Lu, 2018), mean field theory (Yang et al., 2018b), and causal influence (Jaques et al., 2018) have been exploited for multi-agent cooperation. However, communication among all agents (Sukhbaatar et al., 2016; Peng et al., 2017) makes it hard to extract valuable information for cooperation, while communication with only nearby agents (Jiang & Lu, 2018) may restrain the range of cooperation. MeanField (Yang et al., 2018b) captures the interplay of agents by mean action, but the mean action eliminates the difference among agents and thus incurs the loss of important information that could help cooperation. Causal influence (Jaques et al., 2018) is a measure of action influence, which is the policy change of an agent in the presence of an action of another agent. However, causal influence is not directly related to the reward of environment and thus may not encourage cooperation. Nevertheless, none of existing work studies multi-agent cooperation from the perspective of the underlying graph, which could potentially help understand agents' mutual interplay and promote their cooperation as it does in human cooperation.

In this paper, we propose graph convolutional reinforcement learning for multi-agent cooperation, where the multi-agent environment is modeled as a graph, each agent is a node, and the encoding of local observation of agent is the feature of node. We apply convolution operations to the graph of agents. By employing multi-head attention (Vaswani et al., 2017) as the convolution kernel, graph convolution is able to extract the relation representation between nodes and convolve the features from neighboring nodes just like a neuron in a convolutional neural network (CNN). Latent features extracted from gradually increased receptive fields are exploited to learn cooperative policies. The gradient of an agent not only backpropagates to itself but also to other agents in its receptive fields to reinforce the learned cooperative policies. Moreover, the relation representation is temporally regularized to help the agent develop consistent cooperative policy.

Graph convolutional reinforcement learning, namely DGN, is instantiated based on deep Q network, but it can also be realized using policy gradient or actor-critic. DGN is trained end-to-end, adopting the paradigm of centralized training and distributed execution. Moreover, as DGN shares weights among all agents, it is easy to scale, well suited in large-scale MARL. DGN abstracts the mutual interplay between agents by relation kernels, extracts latent

[1]Peking University [2]Macalester College. Correspondence to: Zongqing Lu <zongqing.lu@pku.edu.cn>.

features by convolution, and induces consistent cooperation by temporal relation regularization. We empirically show the learning effectiveness of DGN in jungle and battle games and routing in packet switching networks. We demonstrate that DGN agents are able to develop cooperative and sophisticated strategies and the performance of DGN agents surpasses existing methods in a large margin.

By ablation studies, we demonstrate/verify that: (*i*) Graph convolution greatly enhances the cooperation of agents. Unlike other parameter-sharing methods, graph convolution allows the gradient of one agent to flow to neighbors according to their contribution, promoting the mutual help. (*ii*) Relation kernels help create a policy in MARL that can better generalize to the environment with much more agents, which is very important for real-world applications, since we can train with much fewer agents and directly use in large-scale environments. (*iii*) Temporal regularization, which minimizes the KL divergence of relation representations in successive timesteps, boosts the cooperation, helping the agent to form a long-term and consistent policy in the highly dynamic environment with a lot of moving agents.

To the best of our knowledge, we are the first to propose graph convolutional reinforcement learning in MARL.

2. 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, while networked critics that are updated via communication are considered in (Zhang et al., 2018). However, all these three models have to train an independent policy network for each agent, which tends to learn a policy specializing specific tasks, easily overfits to the number of agents, and does not scale.

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 et al., 2016) uses continuous communication for full cooperative 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 reccurent neural network (RNN) as the communication channel to connect each individual agent's policy and value networks. ATOC (Jiang & Lu, 2018) and TarMAC (Das et al., 2018) enable agents to learn when to communicate and who to send messages to, respectively, using attention mechanism. These communication models prove that communication does help for cooperation. However, full communication is costly and inefficient, while restrained communication limits the range of cooperation.

When the number of agents increases, learning becomes difficult due to the curse of dimensionality and exponential growth of agent interactions. Instead of considering the different effects of other individuals on each agent, Mean-Field (Yang et al., 2018b) 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 could help cooperative decision making. In (Jaques et al., 2018), agents are rewarded for having causal influence over the actions of other agents, where causal influence is assessed using counterfactual reasoning. However, causal influence is not directly related to the reward of environment and thus cannot effectively encourage cooperation.

None of existing work in MARL studies the underlying graph of the multi-agent environment. However, we argue that the underlying graph could greatly promote multi-agent cooperation as it does for human cooperation (Ohtsuki et al., 2006; Apicella et al., 2012).

3. Background

3.1. Graph Convolutional Networks

Many important real-world applications come in the form of graphs, such as social networks (Kipf & Welling, 2017), protein-interaction networks (Duvenaud et al., 2015), and 3D point cloud (Charles et al., 2017). In the last couple of years, several frameworks (Henaff et al., 2015; Niepert et al., 2016; Kipf & 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 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 to produce feature maps.

3.2. 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.

3.3. Relational Reinforcement Learning

The 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 pairwise interaction representation.

4. Method

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 the influence of all agents into consideration, because receiving a large amount of information requires high bandwidth and incurs high computational complexity, and agents cannot differentiate valuable information from globally shared information (Jiang & 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 able to adapt the dynamics of the graph and learn as the multi-agent environment evolves.

4.1. Graph Convolution

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

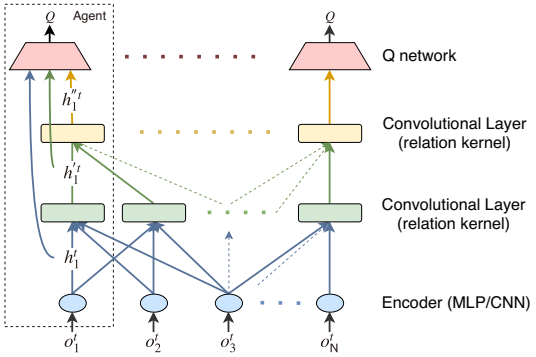


Figure 1. DGN consists of three modules: encoder, convolutional layer, and Q network. All agents share weights and gradients are accumulated to update the weights.

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 one convolutional layer, node i can directly acquire the latent feature vectors from the encoders of 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-hop. However, more convolutional layers will not increase the local region of node i, *i.e.*, node i still only communicates with its K neighbors. This salient characteristic is very important as we consider decentralized execution. Details of the convolution kernel will be discussed in next subsection.

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 $\mathsf{N} \times \mathsf{L}$ 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 $(\mathsf{K}+1) \times \mathsf{N}$ for agent i, where the first row is the one-hot representation of the index of node i, and the $j\mathtt{th}$ row, $j = 2, \ldots, \mathsf{K}+1$, is the one-hot representation of the index of the $(j-1)\mathtt{th}$ 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 concatenated 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 ϵ. 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 maxi-

mizing 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 nearby agents, which makes the environment more stable from the perspective of individual agent.

In DGN, all agents share weights. However, this does not prevent the emergence of sophisticated 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 $\mathcal{O} = \{o_1, \cdots, o_{\mathsf{N}}\}$, $\mathcal{A} = \{a_1, \cdots, a_{\mathsf{N}}\}$, $\mathcal{O}' = \{o'_1, \cdots, o'_{\mathsf{N}}\}$, $\mathcal{R} = \{r_1, \cdots, r_{\mathsf{N}}\}$, and $\mathcal{C} = \{C_1, \cdots, C_{\mathsf{N}}\}$. 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

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

where $y_i = r_i + \gamma \max_{a'} Q\left(O'_i, a'_i; \theta'\right)$, $O_i \subseteq \mathcal{O}$ denotes the set of observations of all the agents in i's receptive fields, γ is the discount factor, and the model is parameterized by θ. 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, 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.

4.2. Relation Kernel

Convolution kernels integrate the information in the receptive field to extract the latent feature. One of the most important properties is that the kernel should be independent from the order of the input feature vectors. Mean operation as in CommNet meets this requirement, but it leads to only marginal performance gain. BiCNet uses the learnable kernel, *i.e.*, RNN. However, the input order of feature vectors severely impacts the performance, though the affect is alleviated by bi-direction mechanism. Further, convolution kernels should be able to learn how to abstract the relation between agents so as to integrate their input features.

Adopting the idea from RRL, we use multi-head dot-product attention as the convolutional 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. The

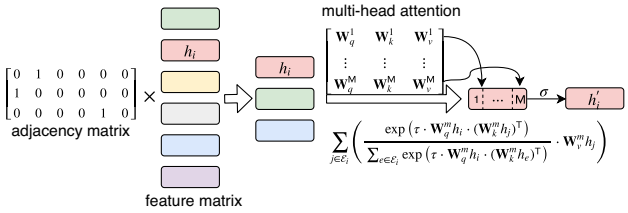


Figure 2. Illustration of computation of the convolutional layer with relation kernel of multi-head attention for an agent with two neighbors.

input feature of each entity is projected to query, key and value representation by each independent attention head. For attention head m, the relation between i and $j \in \mathcal{E}_i$ is computed as

$$\alpha_{ij}^m = \frac{\exp\left(\tau \cdot \mathbf{W}_q^m h_i \cdot (\mathbf{W}_k^m h_j)^{\mathsf{T}}\right)}{\sum_{e \in \mathcal{E}_i} \exp\left(\tau \cdot \mathbf{W}_q^m h_i \cdot (\mathbf{W}_k^m h_e)^{\mathsf{T}}\right)}, \qquad (2)$$

where τ is a scaling factor. For each attention head, the value representations of all the input features are weighed by the relation and summed together. Then, the outputs of M attention heads for agent i are concatenated and then fed into function σ, *i.e.*, one-layer MLP with ReLU non-linearities, to produce the output of the convolutional layer,

$$h_i^{'} = \sigma(\text{Concat}[\sum_{j \in \mathcal{E}_i} \alpha_{ij}^m \mathbf{W}_v^m h_j, \forall m \in \mathsf{M}]). \qquad (3)$$

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

Multi-head attention makes the kernel independent from the order of input feature vectors, and allows the kernel to jointly attend to different representation subspaces. More attention heads give more relation representations and make the training more stable empirically (Vaswani et al., 2017). Moreover, with multiple convolutional layers, higher order relation representations can be extracted, which effectively capture the interplay between agents and greatly help to make cooperative decision.

4.3. 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

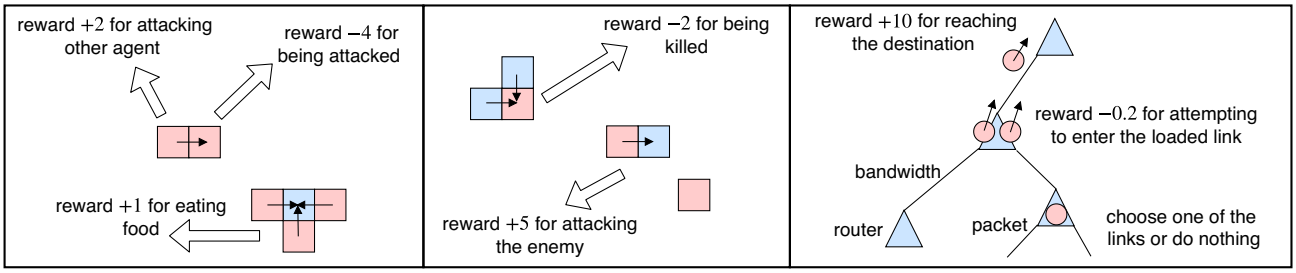


Figure 3. Illustration of experimental scenarios: *jungle* (left), where an agent gets reward by eating good, but gets higher reward by attacking other agents; *battle* (mid), where agents cooperate to fight against more powerful enemies; *routing* (right), where agents try to optimize the mean delay of data packets by determining only the next hop of a packet at a router.

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 the 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 κ for agent i. Then, with temporal relation regularization, the loss is modified as below

$$\mathcal{L}(\theta) = \frac{1}{\mathsf{S}}\sum_{\mathsf{S}}\frac{1}{\mathsf{N}}\sum_{i=1}^{\mathsf{N}}\left(\left(y_i - Q\left(O_i, a_i;\theta\right)\right)^2 + \lambda\, D_{\mathrm{KL}}(\mathcal{G}^{\kappa}(O_i;\theta)||z_i)\right), \tag{4}$$

where $z_i = \mathcal{G}^{\kappa}(O_i';\theta)$ and λ 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.

5. Experiments

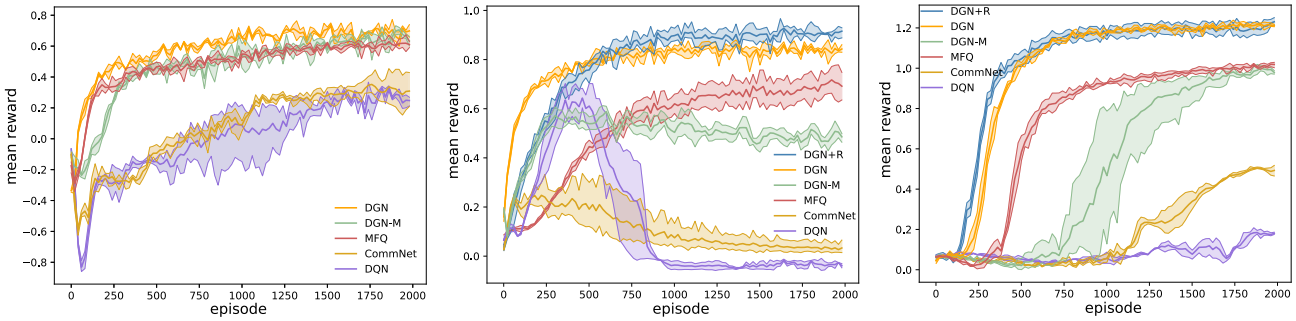
For the experiments, we adopt a grid-world platform MAgent (Zheng et al., 2017). In the environment, each agent corresponds to one grid and has a local observation that contains a square view with 11×11 grids centered at the agent and its own coordinates. The discrete actions are moving or attacking. 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. In the experiments[1], we compare DGN with independent DQN (Mnih et al., 2015), CommNet (Sukhbaatar et al., 2016), and MeanField Q-learning (MFQ) (Yang et al., 2018b). The hyperparameters of DGN and baselines are summarized in Appendix. The video in Appendix provides more details about the experiments. The codes of DGN are also available in Appendix.

5.1. Jungle

This scenario is a moral dilemma. There are N agents and L foods in the field, where foods are stationary. 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 in the setting of $\mathsf{N} = 20$ and $\mathsf{L} = 12$ for 2000 episodes. Figure 4a shows their learning curves, where DGN-M is graph convolution with mean kernel instead of relation kernel, and each model is with three training runs. Table 1 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.

[1]In the experiments, we consider open systems, *e.g.*, battle and routing, where agents come and go, and thus it is infeasible to train a model for every agent. Therefore, we do not consider the methods, which have to train an independent policy network for each agent, as baselines such as MADDPG (Lowe et al., 2017) and COMA (Foerster et al., 2018).



(a) jungle (b) battle (c) routing

Figure 4. Learning curves in terms of mean reward in jungle, battle, and routing, where DGN is graph convolution with relation kernel, DGN-M is graph convolution with mean kernel, and DGN+R is DGN with temporal relation regularization. The effectiveness of graph convolution, relation kernel, and temporal relation regularization can be illustrated respectively by the differences between CommNet and DGN-M, between DGN-M and DGN, and between DGN and DGN+R. For all the methods, the shadowed area is enclosed by the min and max value of different training runs, and the solid line in middle is the mean value.

Table 1. Jungle

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

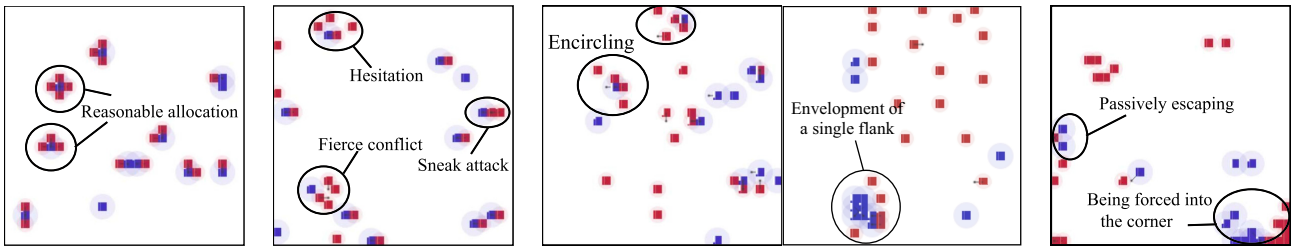
DGN outperforms all the baselines during training and test in terms of mean reward and number of attacks between agents. It is observed that DGN agents can properly select the close food and seldom hurt each other, and the food can be allocated rationally by the surrounding agents, as shown in Figure 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 substantially outperforms CommNet. *This is attributed to the graph convolution that can effectively extract latent features from gradually increased receptive field.* 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 $\mathsf{N} = 20$ and $\mathsf{L} = 12$ to the scenario of $\mathsf{N} = 50$ and $\mathsf{L} = 12$. 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 DGN agents can still rationally share foods even when food is not enough to supply each agent. However, agents of MFQ, CommNet and DQN attack each other more often when there are more agents sharing food.

5.2. 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 moving or attacking range of the agent 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 killed by six attacks). The reward is $+5$ for attacking the enemy, -2 for being killed, and -0.01 for attacking a blank grid. After the death of an agent/enemy, the balance will be easily lost and hence we will add a new agent/enemy at a random location to maintain the balance. By that, we can make fair comparison among different methods in terms of kills, deaths and kill-death ratio besides reward for given timesteps. 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 $\mathsf{N} = 20$ and $\mathsf{L} = 12$ 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 pol-



(a) DGN in jungle (b) DQN in jungle (c) DGN in battle (d) DQN in battle

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

icy, 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 2 shows the mean reward, kills, deaths, and kill-death ratio.

DGN agents learn a series of tactical maneuvers, such as 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 attacks 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 because there is no gradient backpropagation among agents to reinforce the cooperation.

In DGN, relation kernels can extract high order relations between 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 2. In the experiment, it is observed that DGN+R agents indeed behave more consistently 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.*

Table 2. Battle

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

5.3. 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 $\mathsf{N} = 20$ and $\mathsf{L} = 20$ 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.

Table 3. Routing

(N, L)		Floyd	Floyd with BL	DGN+R	DGN	DGN-M	MFQ	CommNet	DQN
(20, 20)	mean reward			**1.23**	**1.21**	0.99	1.02	0.49	0.18
	delay	6.3	8.7	**8.0**	**8.1**	9.8	9.4	18.6	46.7
	throughput	3.17	2.30	**2.50**	**2.47**	2.04	2.13	1.08	0.43
(40, 20)	mean reward			**0.86**	**0.83**	0.70	0.78	0.39	0.12
	delay	6.3	13.7	**9.8**	**10.0**	12.7	11.8	23.5	83.6
	throughput	6.34	2.91	**4.08**	**4.00**	3.15	3.39	1.70	0.49
(40, 20) retrained	mean reward			**0.94**	**0.90**	0.78	0.76	0.35	0.05
	delay	6.3	13.7	**10.2**	**10.5**	12.2	12.8	21.2	112.2
	throughput	6.34	2.91	**3.92**	**3.81**	3.27	3.12	1.86	0.35

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 10 test games, each game unrolled with 300 timesteps. Table 3 shows the mean reward, mean delay of data packets, and throughput, where the delay of a packet is measured by the timesteps taken from source to destination and the throughput is the number of delivered packets per timestep. 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 3). 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 3), which can be considered as the practical solution.

As shown in Table 3, the performance of DGN-M, MFQ, CommNet, and DQN are worse than Floyd with BL. However, the delay and throughput 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 heavier data traffic, *i.e.*, $\mathsf{N} = 40$ and $\mathsf{L} = 20$, where all the models are directly applied to the setting without retraining (their performance with $\mathsf{N} = 60$ and $\mathsf{L} = 20$ is available in Appendix). From Table 3, we can see that DGN+R and DGN still 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. Although the traffic is $2\times$ heavier than before, the delay of DGN+R and DGN only increases about 20%, which makes the throughput $1.6\times$ higher than before. We also retrain all the models in this setting. Interestingly, as show in Table 3, DGN+R and DGN with retraining have slightly higher reward, but longer delay and lower throughput. The reason is that agents trained in heavy traffic pay more attention to avoiding congestion (reducing the penalty), which may induce agents to take a longer path. However, when traffic is light, congestion is less likely and agents mainly focus on finding the shortest path. By the experiments, we can see that our model trained with fewer agents can well generalize to the setting with more agents, which demonstrates that *the policy that takes as input the integrated features from neighboring agents based on their relations scales well with the number of agents.*

6. Conclusions

We have proposed a graph convolutional model for multi-agent cooperation. DGN adapts to the dynamics of the underlying graph of multi-agent environment and exploits convolution with relation kernels to extract latent features from gradually increased receptive fields for learning cooperative strategies. The gradient of an agent not only back-propagates 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 significantly outperforms existing methods in a variety of cooperative multi-agent environments.

References

Apicella, C. L., Marlowe, F. W., Fowler, J. H., and Christakis, N. A. Social networks and cooperation in hunter-gatherers. *Nature*, 481(7382):497, 2012.

Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., et al. Interaction networks for learning about objects, relations and physics. In *NeurIPS'16*, pp. 4502–4510, 2016.

Charles, R. Q., Su, H., Kaichun, M., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR'17*, pp. 77–85, 2017.

Das, A., Gervet, T., Romoff, J., Batra, D., Parikh, D., Rabbat, M., and Pineau, J. Tarmac: Targeted multi-agent communication. *arXiv preprint arXiv:1810.11187*, 2018.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS'15*, pp. 2224–2232, 2015.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *AAAI'18*, 2018.

Henaff, M., Bruna, J., and LeCun, Y. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Hoshen, Y. Vain: Attentional multi-agent predictive modeling. In *NeurIPS'17*, pp. 2701–2711, 2017.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *CVPR'17*, pp. 2261–2269, 2017.

Jaques, N., Lazaridou, A., Hughes, E., Gulcehre, C., Ortega, P. A., Strouse, D., Leibo, J. Z., and de Freitas, N. Intrinsic social motivation via causal influence in multi-agent rl. *arXiv preprint arXiv:1810.08647*, 2018.

Jiang, J. and Lu, Z. Learning attentional communication for multi-agent cooperation. *NeurIPS'18*, 2018.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR'17*, 2017.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NeurIPS'17*, pp. 6379–6390, 2017.

Matignon, L., Jeanpierre, L., Mouaddib, A.-I., et al. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *AAAI'12*, 2012.

Melis, A. P. and Semmann, D. How is human cooperation different? *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 365(1553): 2663–2674, 2010.

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. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *ICML'16*, pp. 2014–2023, 2016.

Ohtsuki, H., Hauert, C., Lieberman, E., and Nowak, M. A. A simple rule for the evolution of cooperation on graphs and social networks. *Nature*, 441(7092):502, 2006.

Peng, P., Wen, Y., Yang, Y., Yuan, Q., Tang, Z., Long, H., and Wang, J. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

Shalev-Shwartz, S., Shammah, S., and Shashua, A. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

Sukhbaatar, S., Fergus, R., et al. Learning multiagent communication with backpropagation. In *NeurIPS'16*, pp. 2244–2252, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NeurIPS'17*, pp. 5998–6008, 2017.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Watters, N., Tacchetti, A., Weber, T., Pascanu, R., Battaglia, P., and Zoran, D. Visual interaction networks. *arXiv preprint arXiv:1706.01433*, 2017.

Wiering, M. Multi-agent reinforcement learning for traffic light control. In *ICML'00*, pp. 1151–1158, 2000.

Yang, Y., Hao, J., Sun, M., Wang, Z., Fan, C., and Strbac, G. Recurrent deep multiagent q-learning for autonomous brokers in smart grid. In *IJCAI'18*, pp. 569–575, 2018a.

Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., and Wang, J. Mean field multi-agent reinforcement learning. In *ICML'18*, pp. 5571–5580, 2018b.

Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

Table 4. Hyperparameters

Hyperparameter	DGN	CommNet	MFQ	DQN
discount (γ)		$0.96, 0.96, 0.98$		
batch size		10		
buffer capacity		2×10^5		
β		0.01		
ϵ and decay		$0.6/0.996$		
optimizer		Adam		
learning rate		10^{-4}		
# neighbors (K)	3	–	3	–
# convolutional layers	2		–	
# attention heads	8		–	
τ	0.25		–	
λ	0.03		–	
κ	2		–	
# encoder MLP layers	2	2	–	–
# encoder MLP units	$(512, 128)$	$(512, 128)$	–	–
Q network	affine transformation	affine transformation	$(1024, 256)$	$(1024, 256)$
MLP activation		ReLU		
initializer		random normal		

Table 5. Routing in the setting of $N = 60$ and $L = 20$ using the models trained in $N = 20$ and $L = 20$

(N, L)		Floyd	Floyd with BL	DGN+R	DGN	DGN-M	MFQ	CommNet	DQN
$(60, 20)$	mean reward			**0.73**	**0.69**	0.56	0.59	0.31	0.06
	delay	6.3	14.7	**12.6**	**13.4**	16.2	15.5	27.0	132.0
	throughput	9.52	4.08	**4.76**	**4.48**	3.70	3.87	2.22	0.45

Zhang, K., Yang, Z., Liu, H., Zhang, T., and Başar, T. Fully decentralized multi-agent reinforcement learning with networked agents. In *ICML'18*, 2018.

Zheng, L., Yang, J., Cai, H., Zhang, W., Wang, J., and Yu, Y. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. *arXiv preprint arXiv:1712.00600*, 2017.

A. Hyperparameters

Table 4 summarizes the hyperparameters used by DGN and the baselines in the experiments.

B. Additional Results

Table 5 gives the experimental result of routing when applying the models trained in the setting of $N = 20$ and $L = 20$ to $N = 60$ and $L = 20$. We can see that DGN+R and DGN still substantially outperform the baselines and Floyd with BL.

C. Video and Codes

The video of the experiments is given by this link https://goo.gl/AFV9qi. The video shows the games in jungle, battle, and routing. The video aims to highlight different behaviors of trained agents. The codes of DGN are available at https://github.com/PKU-AI-Edge/GraphConv4MARL.git/.