Graph Convolutional Reinforcement Learning

Sunoh Kim

Seoul National University

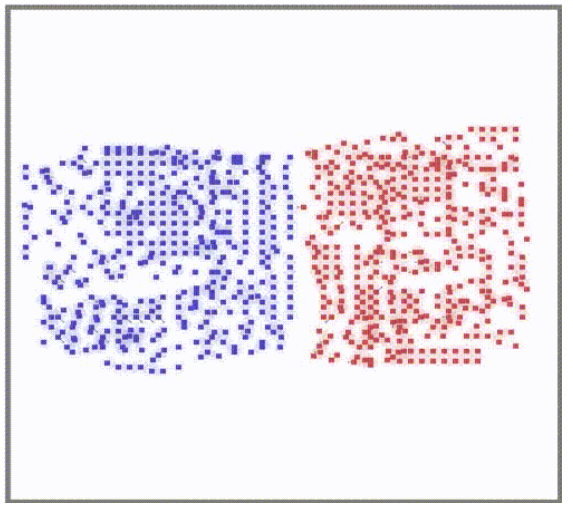
[1] Jiang, Jiechuan, et al. "Graph Convolutional Reinforcement Learning.", In *ICLR,* 2020.

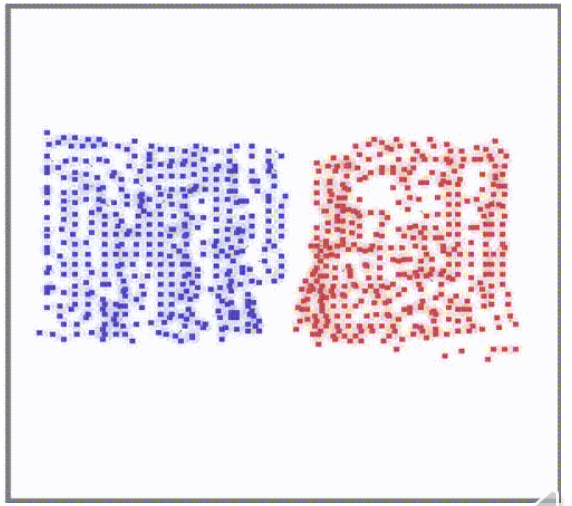
Learning to Cooperate

- In multi-agent environments, learning to cooperate is important.

- The key is to understand the mutual interplay between agents. But, it is hard to learn abstract representations of mutual interplay between agents.

Battle

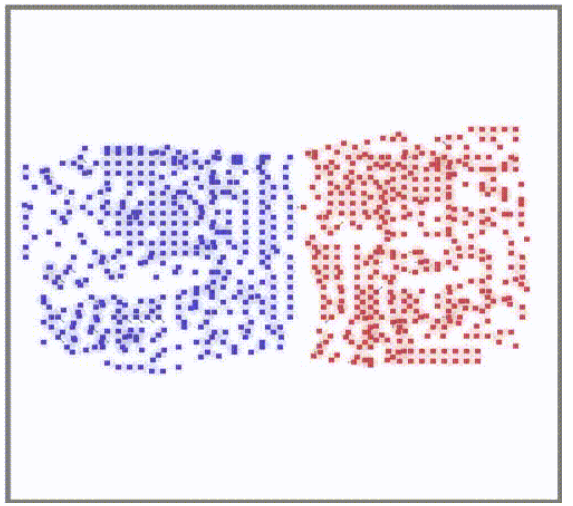


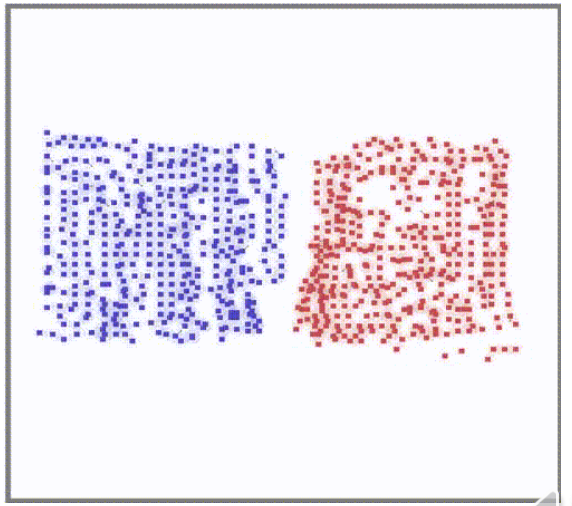


Learning to Cooperate

- In multi-agent environments, learning to cooperate is important.
- The key is to understand the mutual interplay between agents. But, it is hard to learn abstract representations of mutual interplay between agents.
- Graph convolutional reinforcement learning (DGN) is proposed to promote agent' cooperation by considering the underlying graph of agents.

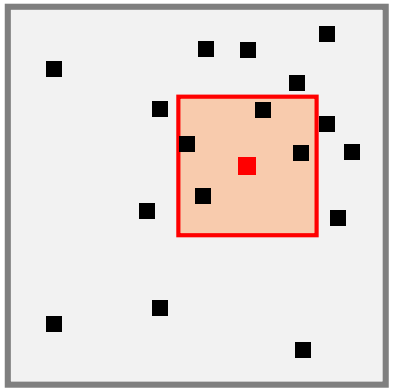
Battle



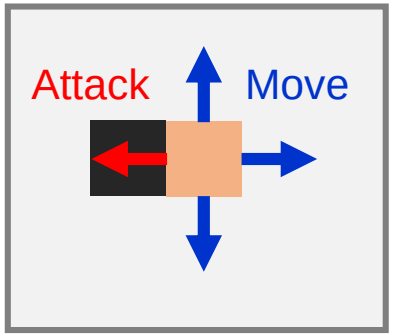


A Grid-World Platform: MAgent [2]

- Environment: A grid-world with 30 × 30 grids
- Each agent: One colored grid
- A local observation of the agent: A square view with 11 × 11 grids centered at the agent
- Actions of the agent: moving or attacking (two discrete actions)
- Scenarios: Battle and Jungle



A Grid-World



Actions

[2] Zheng, Lianmin, et al. "MAgent: A many-agent reinforcement learning platform for artificial collective intelligence.“, in *AAAI,* 2018.

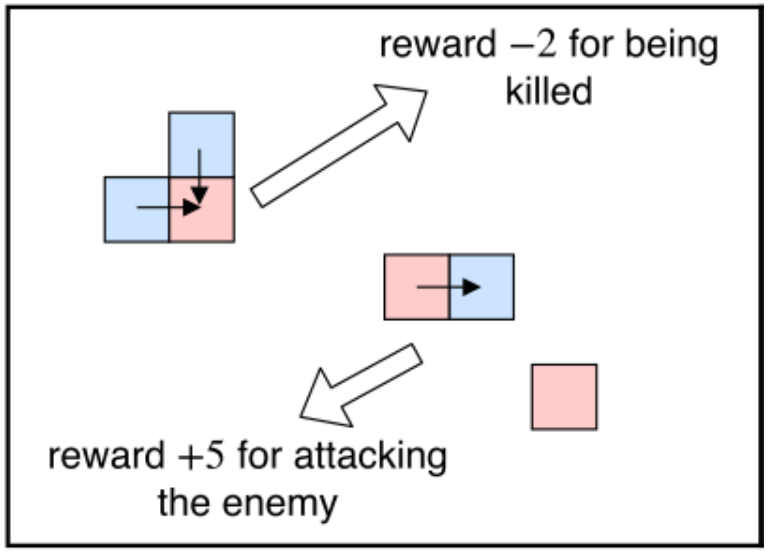
A Grid-World Platform: MAgent [2]

- Battle:

  N Agents learn to fight against L enemies.

  Each agent/enemy has six hit points.

  (i.e., being killed by six attacks).

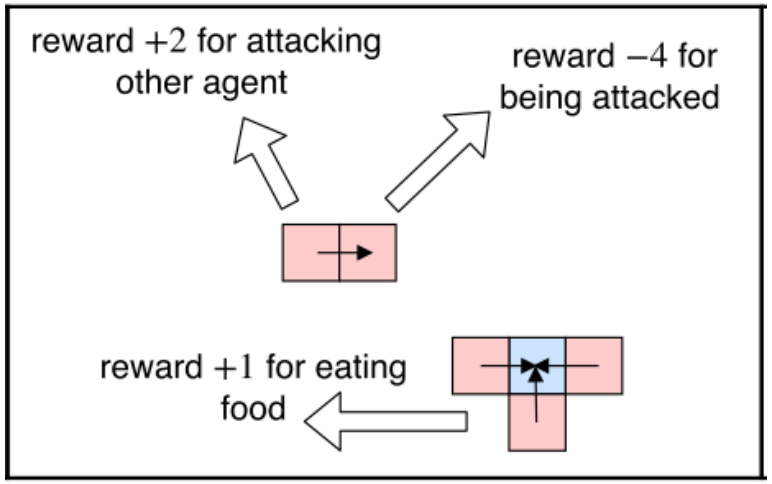


- Jungle:

  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.

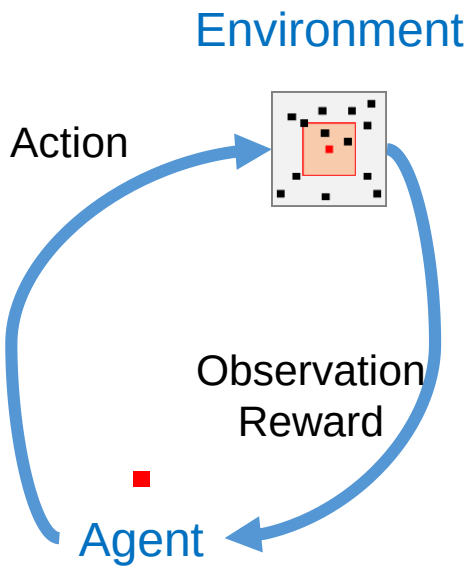


[2] Zheng, Lianmin, et al. "MAgent: A many-agent reinforcement learning platform for artificial collective intelligence.“, in *AAAI,* 2018.

A Grid-World Platform: MAgent [2]

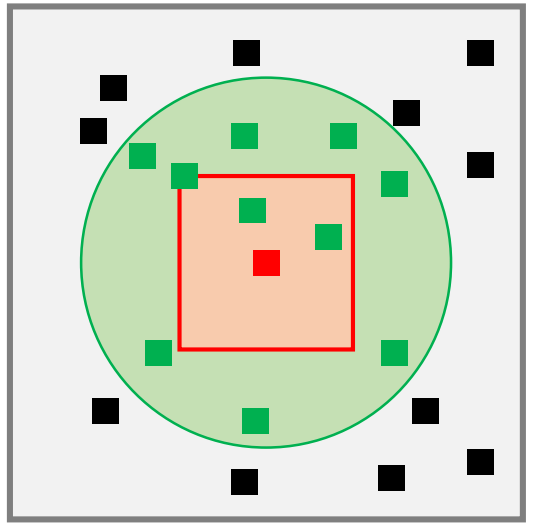
- The problem is formulated as Markov Decision Process. At each timestep, each agent receives a local observation, takes an action, and gets an individual reward.

- The objective is to maximize the sum of all agents' expected sum of rewards.



[2] Zheng, Lianmin, et al. "MAgent: A many-agent reinforcement learning platform for artificial collective intelligence.", in *AAAI,* 2018.

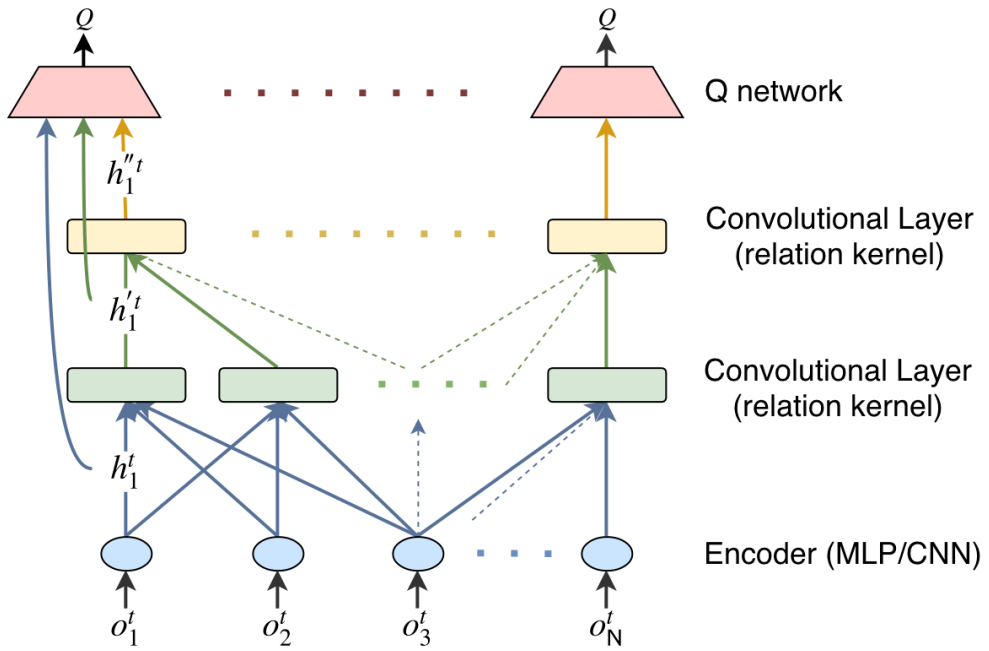
Graph Construction



- The multi-agent environment is constructed as a graph.
- Agents in the environment are represented by the nodes of the graph.
- The feature of node is the encoding of local observation of agent.
- Each node i has a set of neighbors $\mathbb{B}_i$ which is determined by distance or other metrics, depending on the environment, and varies over time.
- Unlike the static graph considered in GCNs, the graph of multi-agent environment is dynamic and continuously changing over time as agents move or enter/leave the environment.

Graph Convolutional Reinforcement Learning

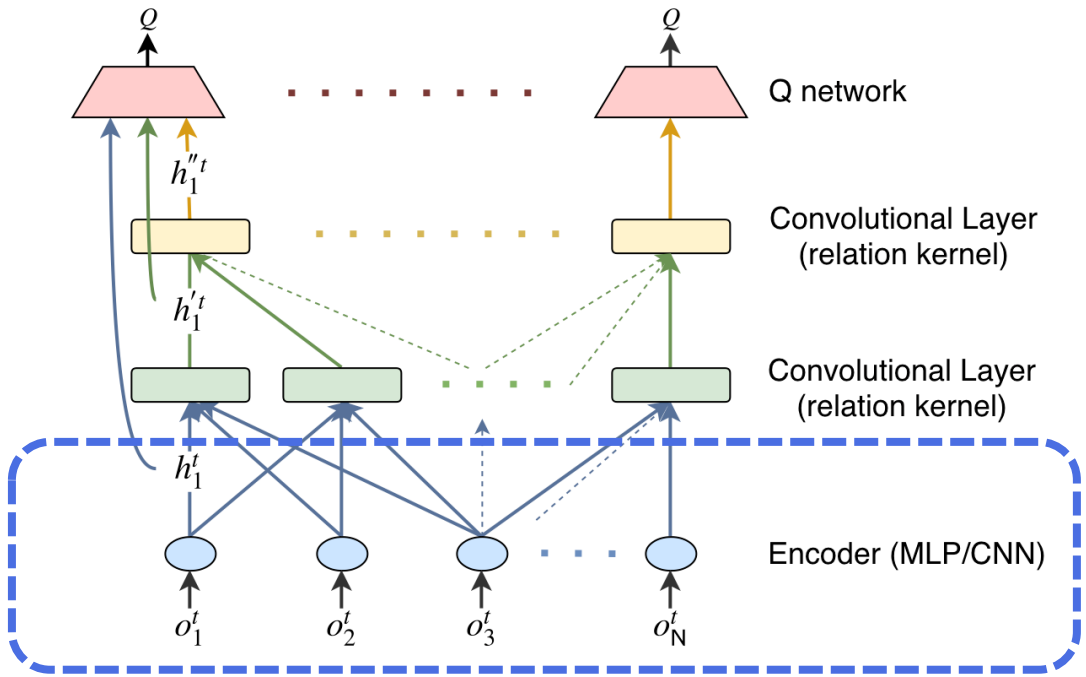
- DGN consists of three types of modules:



Graph Convolutional Reinforcement Learning

- DGN consists of three types of modules:

1. Observation encoder: encodes the local observation into a feature vector using MLP (Multi Layer Perception).

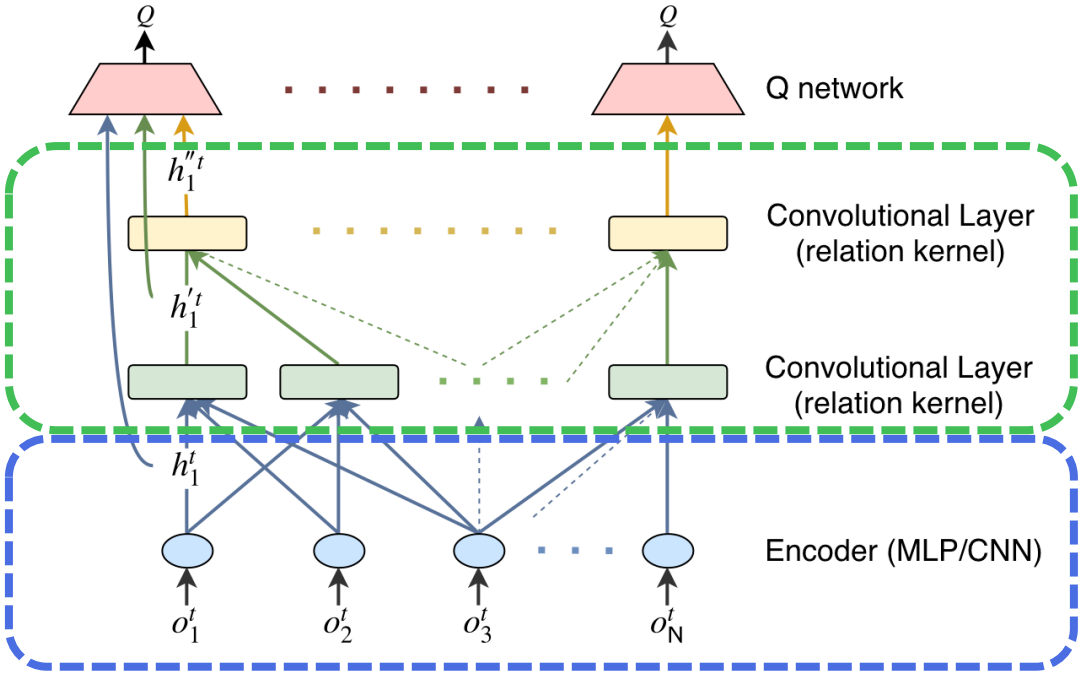


Graph Convolutional Reinforcement Learning

- DGN consists of three types of modules:

1. Observation encoder: encodes the local observation into a feature vector using MLP (Multi Layer Perception).
2. Convolutional layer: integrates the feature vectors in the local region and generate the latent feature vector.

   By stacking more convolutional layers, the receptive field of an agent gradually grows.

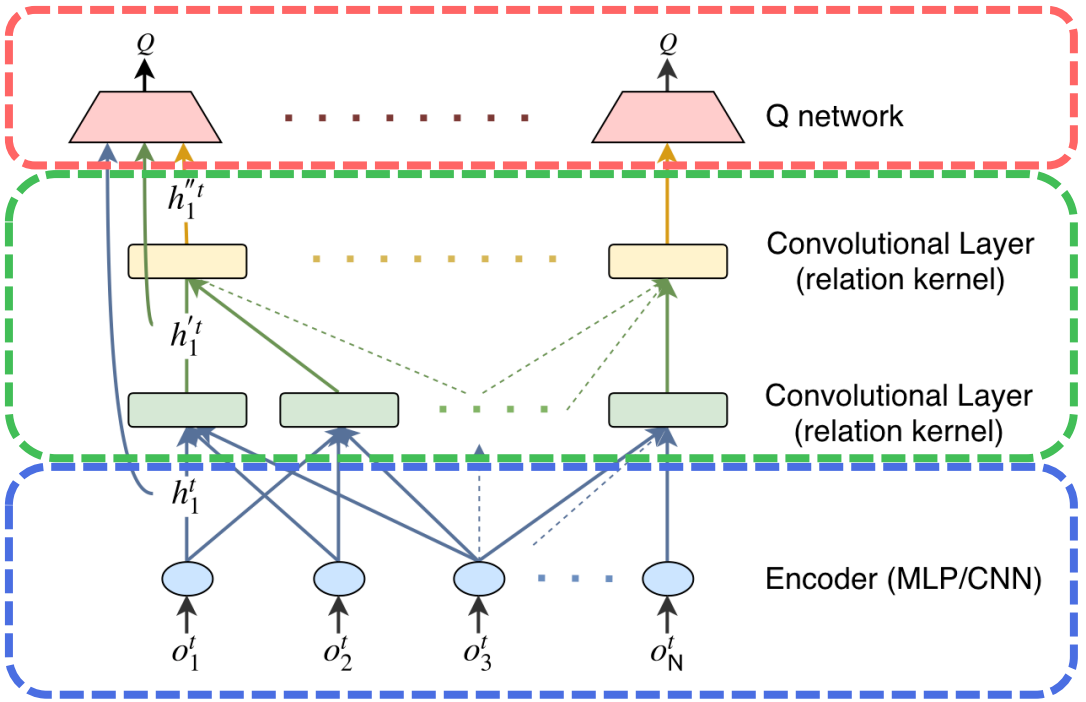


Graph Convolutional Reinforcement Learning

- DGN consists of three types of modules:

1. Observation encoder: encodes the local observation into a feature vector using MLP (Multi Layer Perception).
2. Convolutional layer: integrates the feature vectors in the local region and generate the latent feature vector.

   By stacking more convolutional layers, the receptive field of an agent gradually grows.



3. Q-network: 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.

Graph Convolutional Reinforcement Learning

1. Observation encoder: encodes the local observation into a feature vector using MLP (Multi Layer Perception).

i : agent index

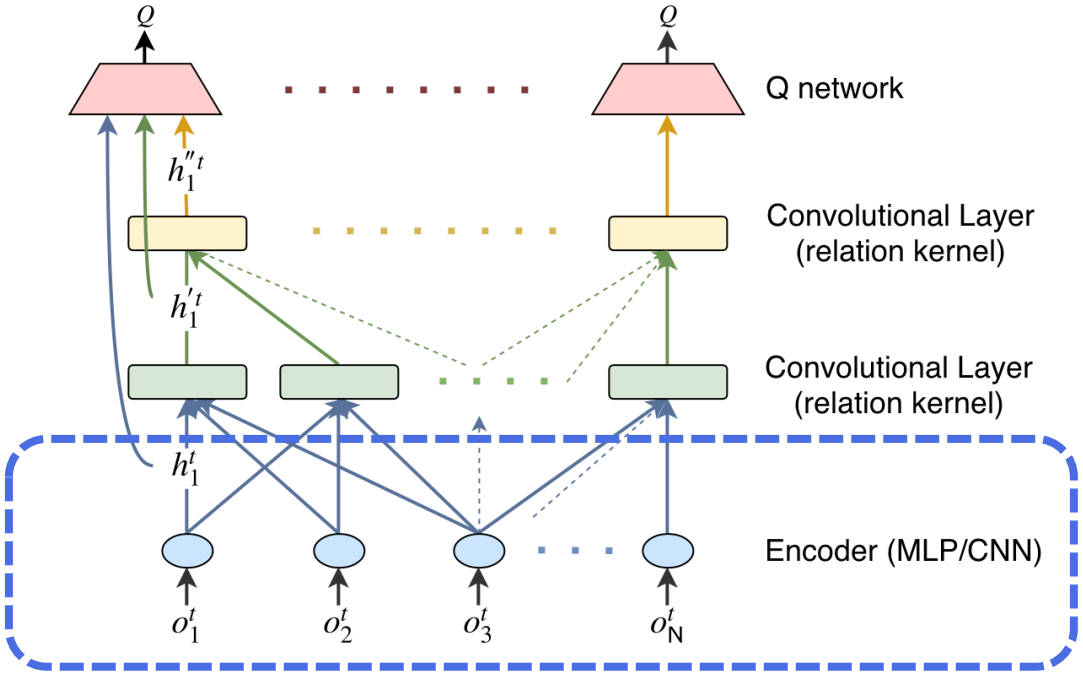
t : time step

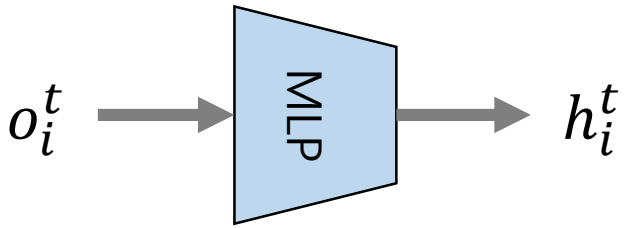
$\mathbb{B}_i$: a set of neighbors

o_i^t : local observation

a_i^t : action

r_i^t : reward

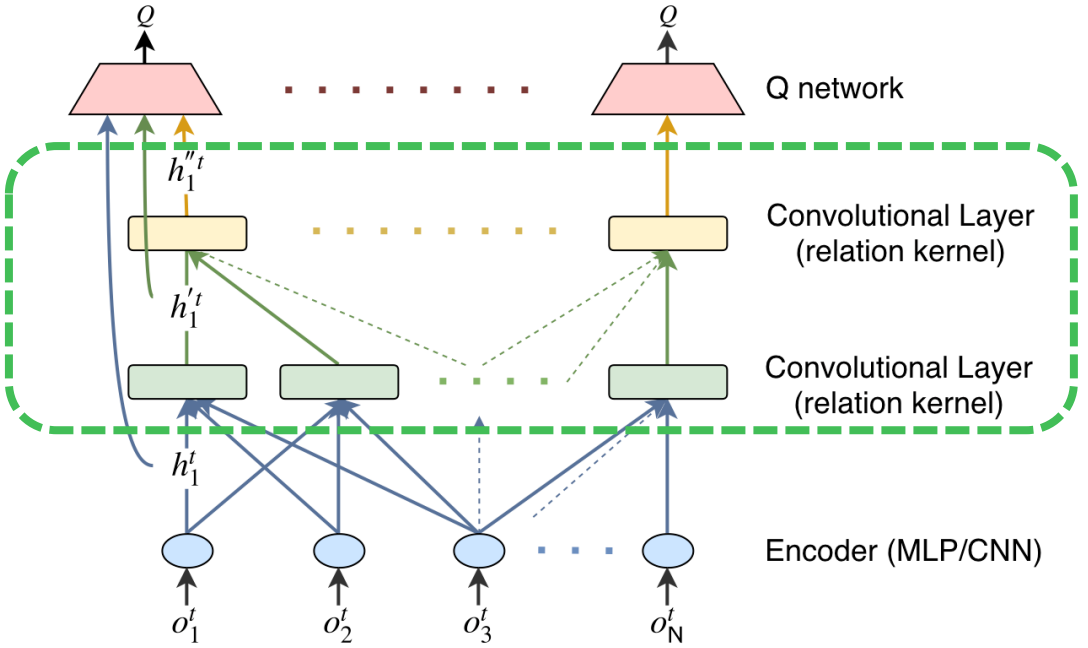




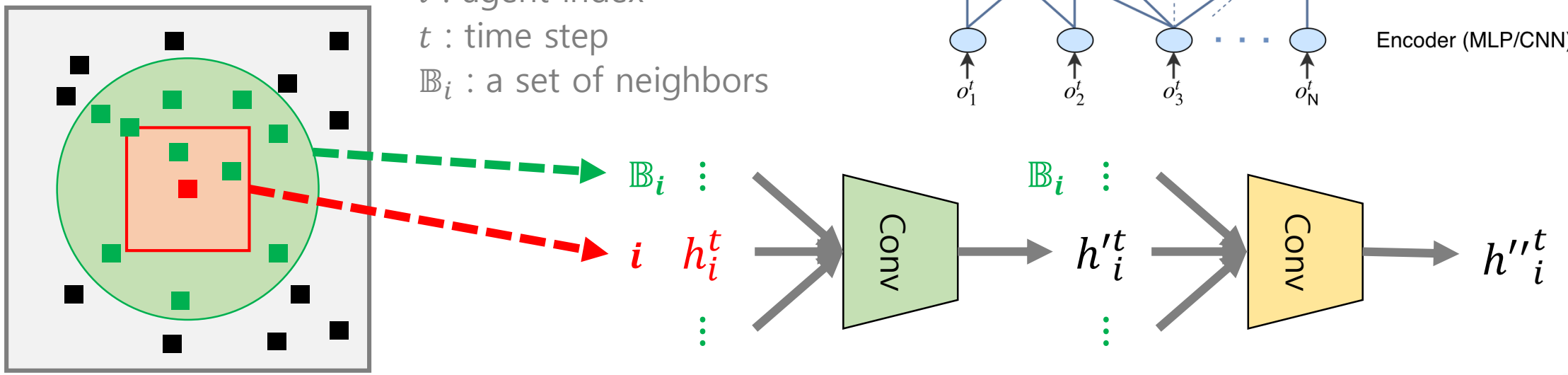
Graph Convolutional Reinforcement Learning

2. Convolutional layer: integrates the feature vectors in the local region and generate the latent feature vector.

   By stacking more convolutional layers, the receptive field of an agent gradually grows.



i : agent index
t : time step
$\mathbb{B}_i$: a set of neighbors



Graph Convolutional Reinforcement Learning

2. Convolutional layer:

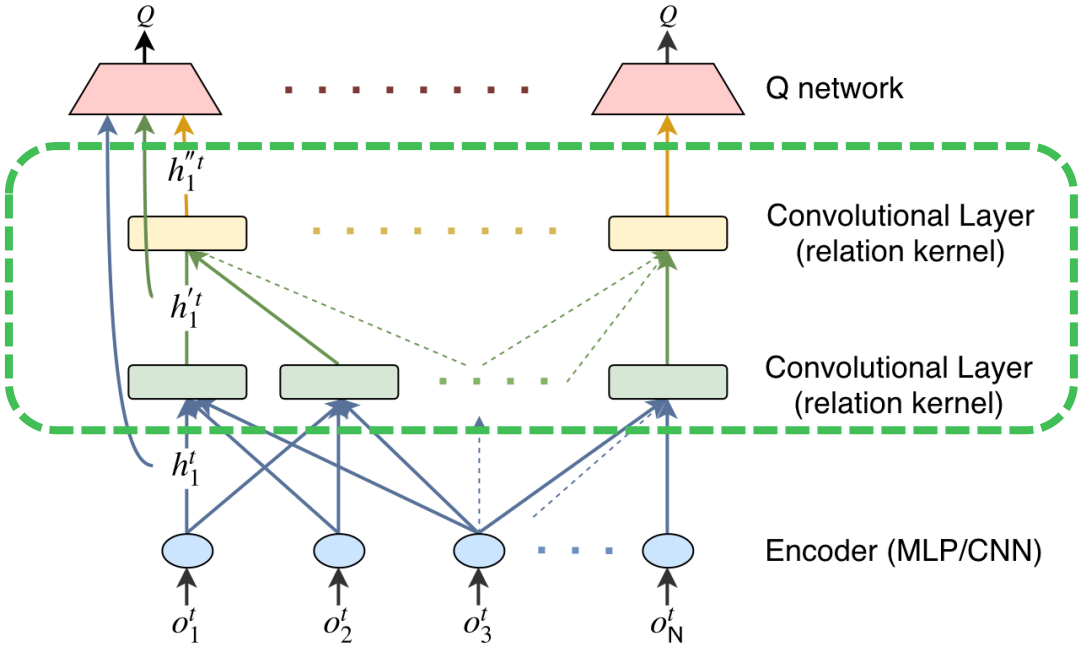
i : agent index

t : time step

$\mathbb{B}_i$: a set of neighbors

N : the number of agents

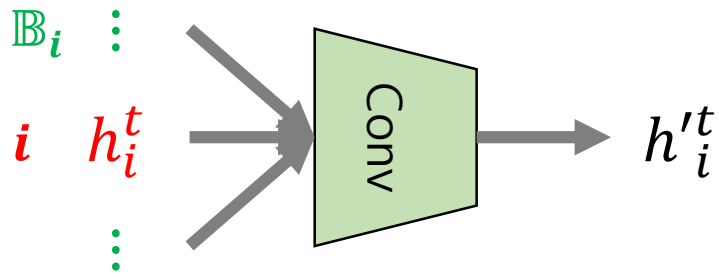
L : the length of feature vector



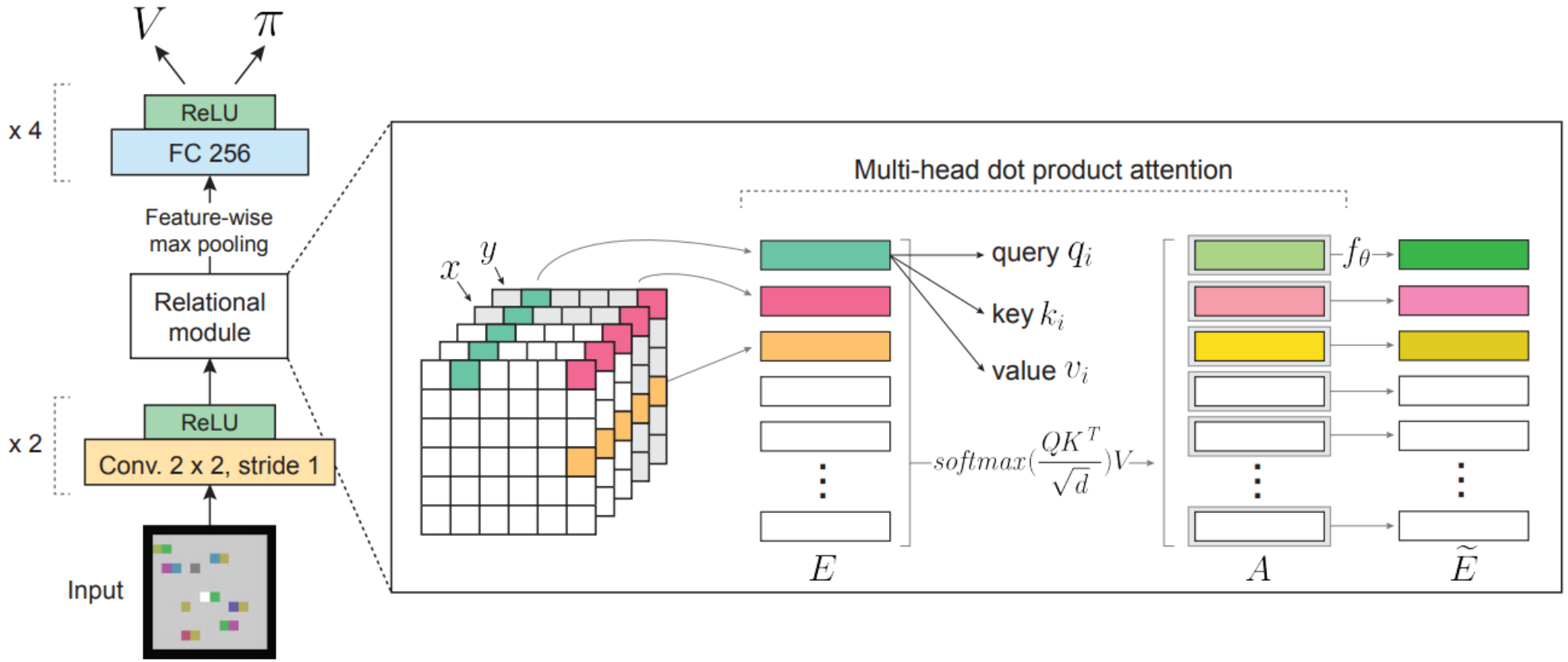
$F^t \in \mathbb{R}^{N \times L}$: feature matrix

$C_i^t \in \mathbb{R}^{(|\mathbb{B}_i|+1) \times N}$: adjacency matrix

$C_i^t \times F^t \in \mathbb{R}^{(|\mathbb{B}_i|+1) \times L}$: feature vectors in the local region



Relational Kernel [3]



[3] Zambaldi, Vinicius, et al. "Relational deep reinforcement learning." arXiv preprint arXiv:1806.01830, 2018.

Graph Convolutional Reinforcement Learning

3. Q-network: 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.

i : agent index

t : time step

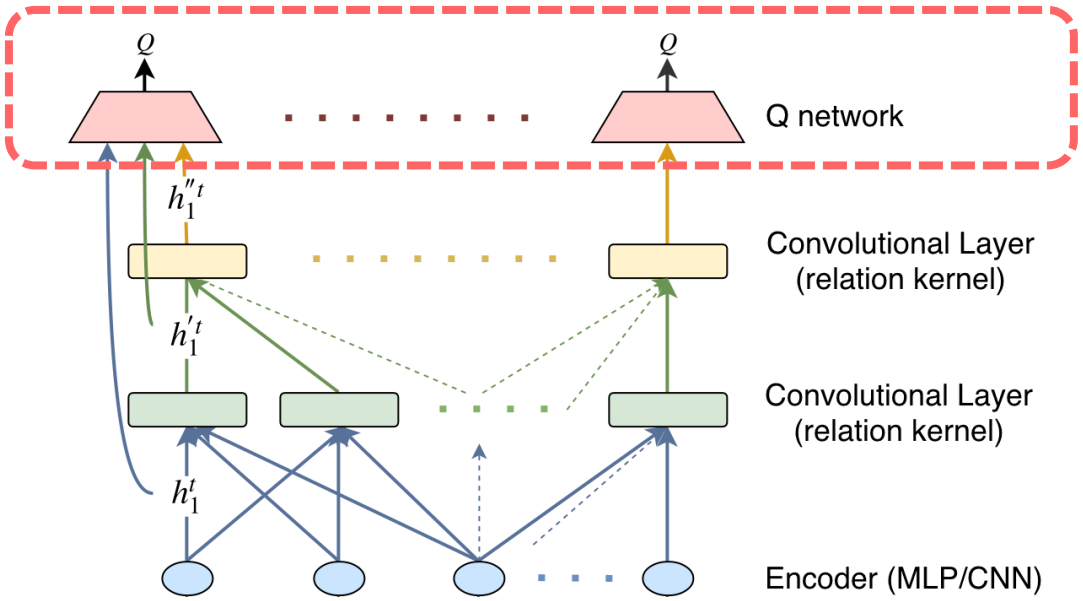
$\mathbb{B}_i$: a set of neighbors

N : the number of agents

o_i^t : local observation

a_i^t : action

r_i^t : reward



- Store the tuple in the replay buffer

$$(\mathcal{O}, \mathcal{A}, \mathcal{O}', \mathcal{R}, \mathcal{C})$$

$\mathcal{O} = \{o_1, \cdots, o_N\}$: a set of observations

$\mathcal{A} = \{a_1, \cdots, a_N\}$: a set of actions

$\mathcal{O}' = \{o_1', \cdots, o_N'\}$: a set of next obs.

$\mathcal{R} = \{r_1, \cdots, r_N\}$: a set of rewards

$\mathcal{C} = \{C_1, \cdots, C_N\}$: a set of adjacency mat.

Graph Convolutional Reinforcement Learning

3. Q-network:

$\mathcal{O} = \{o_1, \cdots, o_N\}$: a set of observations
$\mathcal{A} = \{a_1, \cdots, a_N\}$: a set of actions
$\mathcal{O}' = \{o'_1, \cdots, o'_N\}$: a set of next obs.
$\mathcal{R} = \{r_1, \cdots, r_N\}$: a set of rewards
$\mathcal{C} = \{C_1, \cdots, C_N\}$: a set of adjacency mat.

- Store the tuple in the replay buffer

$$(\mathcal{O}, \mathcal{A}, \mathcal{O}', \mathcal{R}, \mathcal{C})$$

- Sample a random minibatch of size S from the replay buffer and minimize the loss

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

Q function parameterized by θ

$$y_i = r_i + \gamma \max_{a'} Q\left(O'_{i,\mathcal{C}}, a'_i; \theta'\right)$$: the target

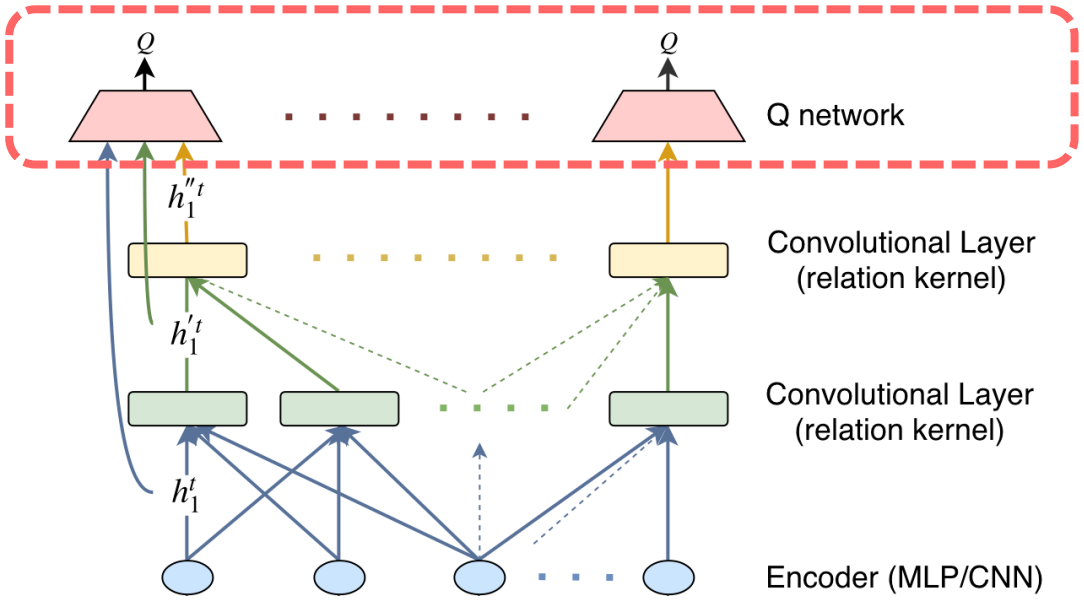
Reinforcement Learning [4]

- Q value function: $Q(o^t, a^t) = E[\sum_{k=0}^{\infty} r^{t+1+k} | o^t, a^t]$
- Optimal action: $a^* = \text{argmax}_a\, Q_\pi(o^t, a^t)]$

[4] Sutton, Richard S., and Andrew G. Barto. “Introduction to reinforcement learning.”, Vol. 135. Cambridge: MIT press, 1998.

Graph Convolutional Reinforcement Learning

3. Q-network:



- Sample a random minibatch of size S from the replay buffer and minimize the loss

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

Q function parameterized by θ

$$y_i = r_i + \gamma \max_{a'} Q\left(O'_{i,\mathcal{C}}, a'_i; \theta'\right)$$: the target

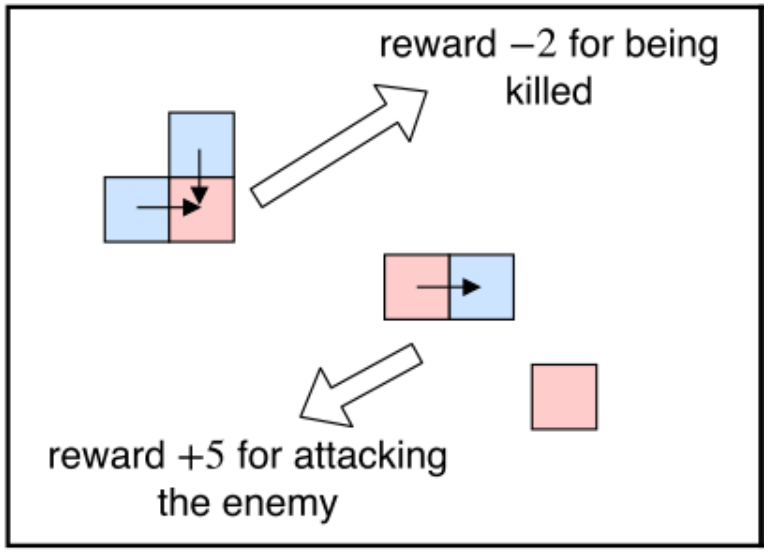
Experiments

- Battle:

  N Agents learn to fight against L enemies.

  Each agent/enemy has six hit points.

  (i.e., being killed by six attacks).

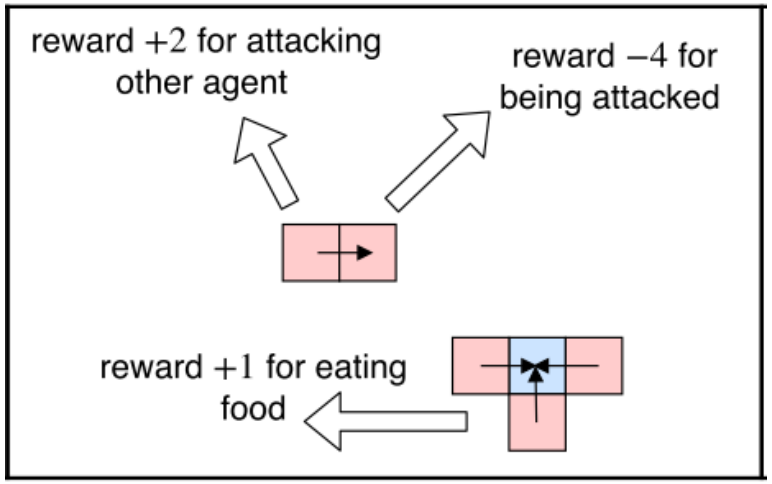


- Jungle:

  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.



[2] Zheng, Lianmin, et al. "MAgent: A many-agent reinforcement learning platform for artificial collective intelligence.“, in *AAAI,* 2018.

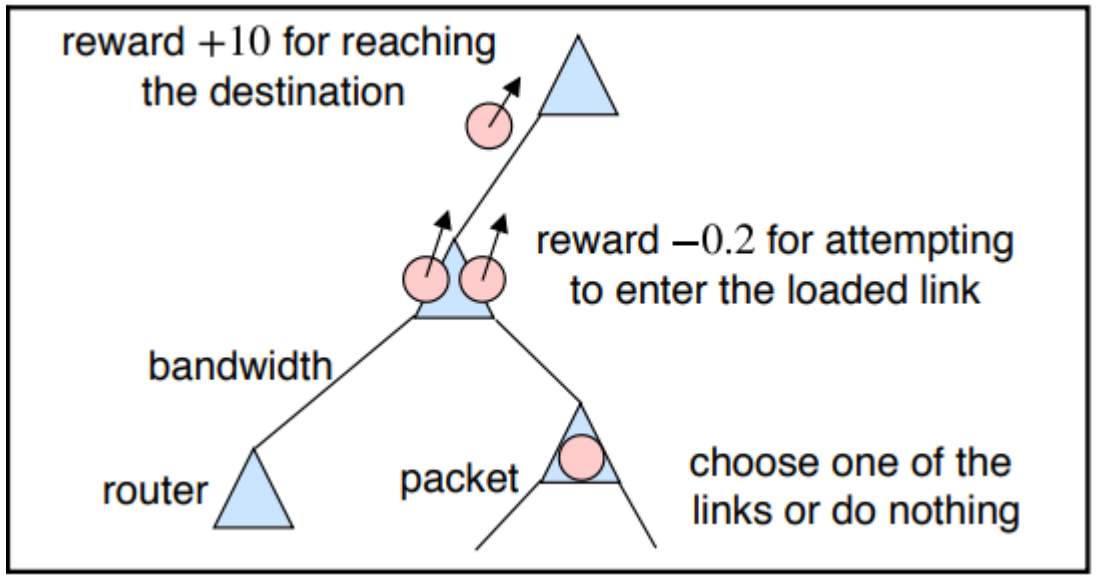
Experiments

- Routing:

The network consists of L routers.

Each router is randomly connected to a constant number of routers.

There are N data packets (agents) with a random size, and each packet is randomly assigned a source and destination router.



In the experiment, agents aim to quickly reach the destination while avoiding congestion.

Experiments

- Battle:

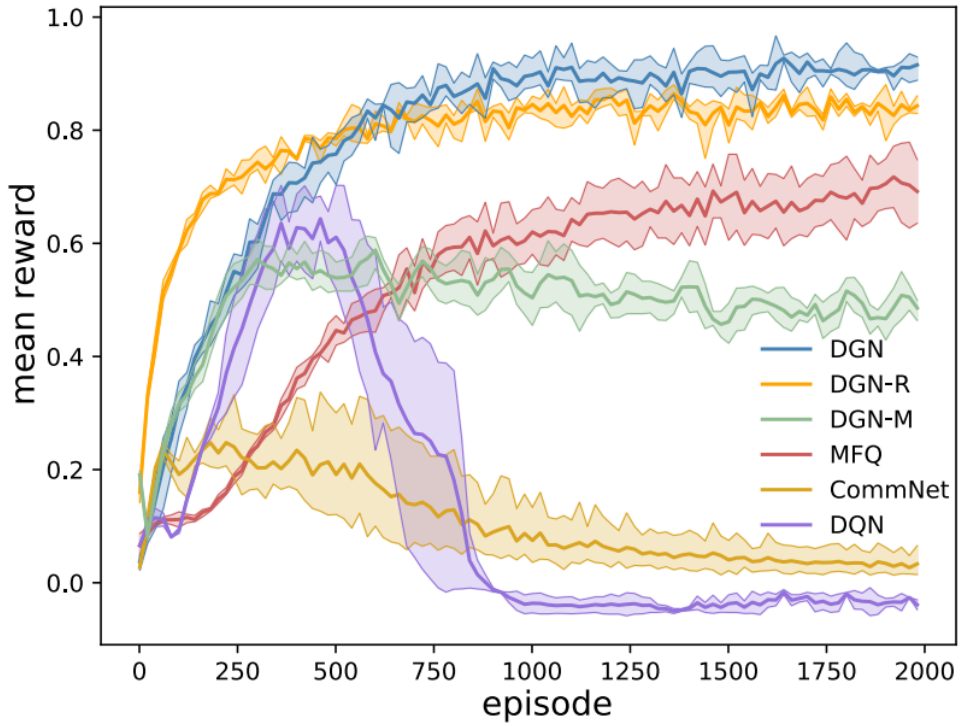


Table 1: Battle

	DGN	DGN-R	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

- Jungle:

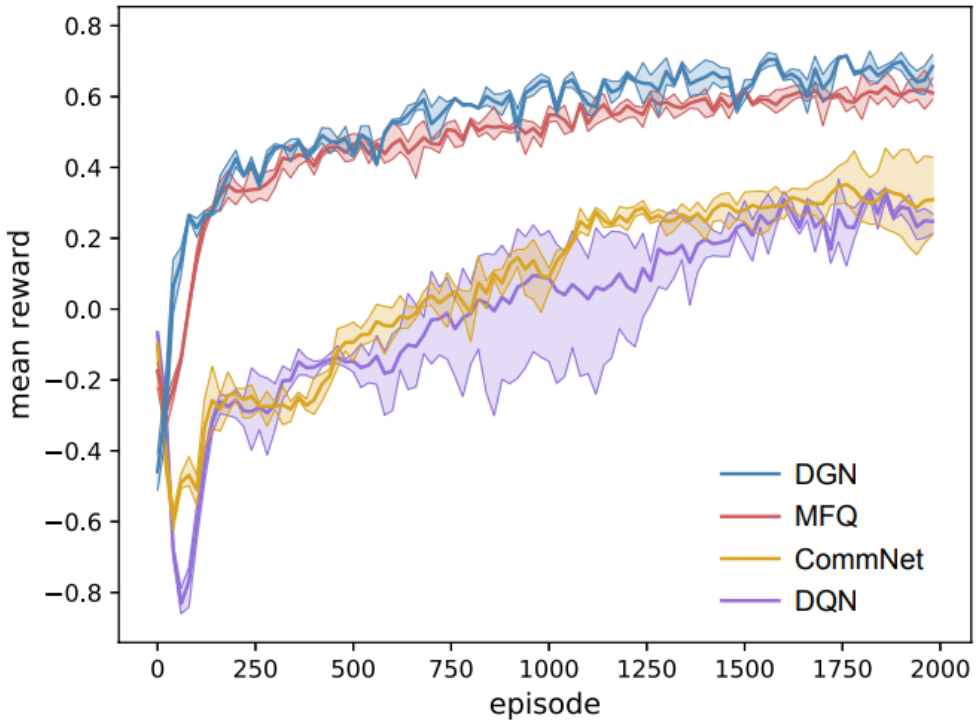


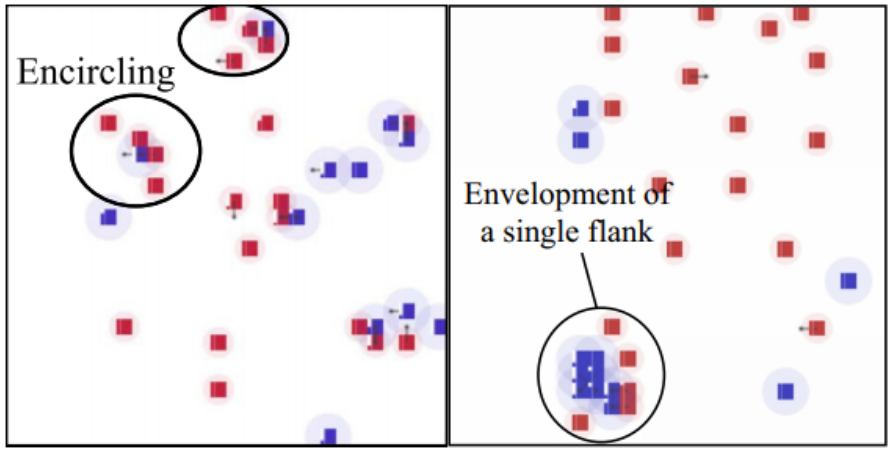
Table 2: Jungle

	DGN	MFQ	CommNet	DQN
mean reward	**0.66**	0.62	0.30	0.24
# attacks	**1.14**	2.74	5.44	7.35

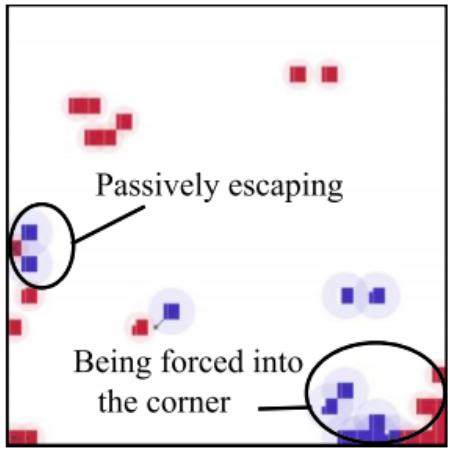
Experiments

- Battle:

Red : Our agents
Blue : Enemy



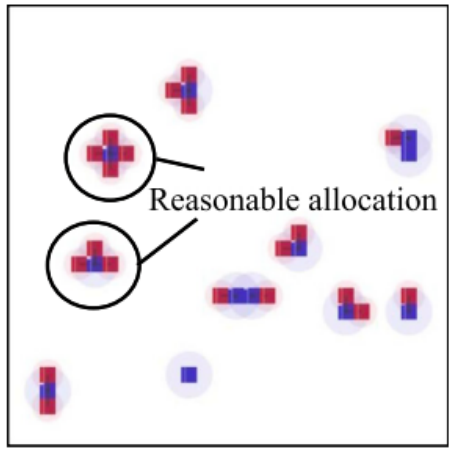
(a) DGN in battle



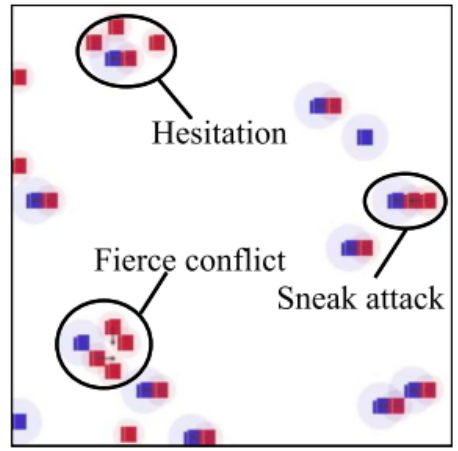
(b) DQN in battle

- Jungle:

Red : Our agents
Blue : Food



(c) DGN in jungle



(d) DQN in jungle

Table 1: Battle

	DGN	DGN-R	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

Table 2: Jungle

	DGN	MFQ	CommNet	DQN
mean reward	**0.66**	0.62	0.30	0.24
# attacks	**1.14**	2.74	5.44	7.35

Experiments

- Routing:

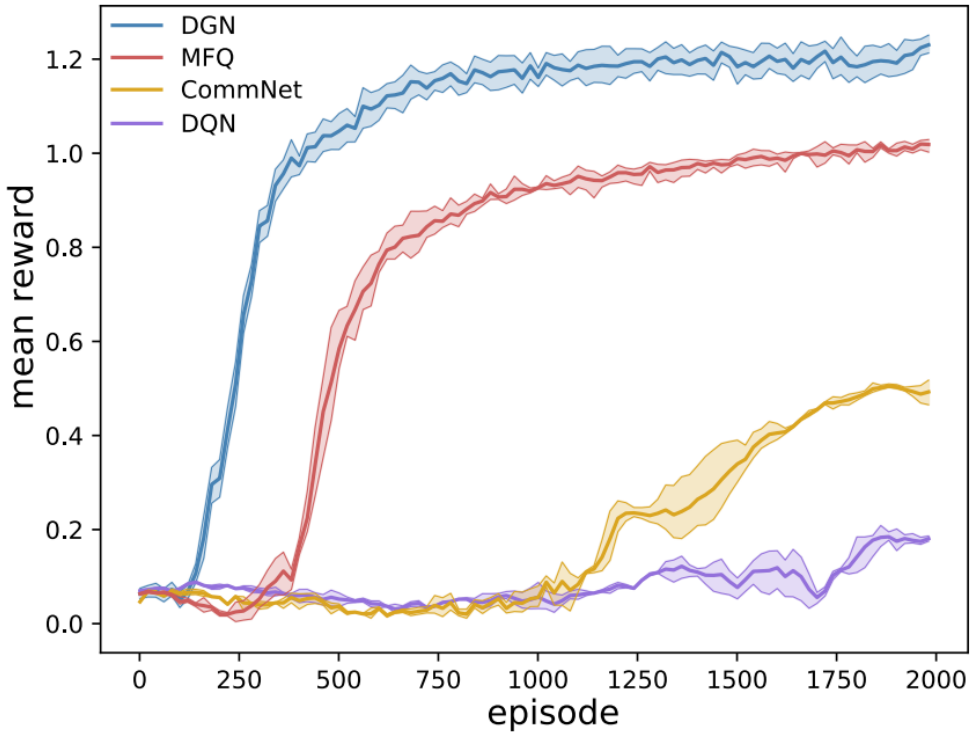


Table 3: Routing

(N, L)		Floyd	Floyd w/ BL	DGN	MFQ	CommNet	DQN
(20, 20)	mean reward			**1.23**	1.02	0.49	0.18
	delay	6.3	8.7	**8.0**	9.4	18.6	46.7
	throughput	3.17	2.30	**2.50**	2.13	1.08	0.43
(40, 20)	mean reward			**0.86**	0.78	0.39	0.12
	delay	6.3	13.7	**9.8**	11.8	23.5	83.6
	throughput	6.34	2.91	**4.08**	3.39	1.70	0.49
(60, 20)	mean reward			**0.73**	0.59	0.31	0.06
	delay	6.3	14.7	**12.6**	15.5	27.0	132.0
	throughput	9.52	4.08	**4.76**	3.87	2.22	0.45

Conclusion

- We have proposed graph convolutional reinforcement learning.
- Our proposed network (DGN) adapts to the dynamics of the underlying graph of the multi-agent environment and exploits convolution from gradually increased receptive fields for learning cooperative strategies.
- Empirically, DGN significantly outperforms existing methods in a variety of cooperative multi-agent scenarios.

Thank you

Final Report

0. Requirement

   Ubuntu 16.04

   Anaconda 3

   Python 3.6

   Cuda 9.2

   Cudnn 7.6.3

Final Report

1. Setting RL Environment

   First, we have to install MAgent [2] which is a research platform for multi-agent reinforcement learning. MAgent aims at supporting reinforcement learning research that scales up from hundreds to millions of agents.

   Link : https://github.com/geek-ai/MAgent

   - Installing MAgent on Linux:

     git clone https://github.com/geek-ai/MAgent.git

     cd MAgent

     sudo apt-get install cmake libboost-system-dev libjsoncpp-dev libwebsocketpp-dev

     bash build.sh

     export PYTHONPATH=$(pwd)/python:$PYTHONPATH

     (You may need to add this to the bashrc by typing “sudo vim ~/.bashrc”)

[2] Zheng, Lianmin, et al. "MAgent: A many-agent reinforcement learning platform for artificial collective intelligence.“, in *AAAI,* 2018.

Final Report

2. Installing Dependencies

   Before implementing DGN [1], we need to create a new conda environment (If you want) and install dependencies using pip.

   Link : https://github.com/suno8386/DGN-reproduced

   - Creating new conda environment:

     conda create -n <Your Environment Name> python=3.6

     conda activate <Your Environment Name>

   - Cloning DGN codes and installing dependencies:

     git clone https://github.com/suno8386/DGN-reproduced.git

     cd DGN-reproduced

     pip install -r requirements.txt

   (Actually the authors used tensorflow 2.1.0 and Keras 2.3.1. But some functions are deprecated in them. So, I used tensorflow 1.14.0 and Keras 2.0.8 and modified the codes accordingly.)

[1] Jiang, Jiechuan, et al. "Graph Convolutional Reinforcement Learning.", In *ICLR,* 2020.

Final Report

3. Implementing DGN

   Now, we will reproduce the result of the DGN [1] in the Routing scenario.

   - Reproducing DGN:

     cd Routing

     ```
     # without regularization
     python routers.py

     # with regularization
     python routers_regularization.py


     # The result will be saved in the log file.
     # e.g. /home/pil-kso/DGN-reproduced/log_router_gqn.txt
     ```

Final Report

- Reproduced Results

  The number of agents (data packets): $N = 20$

  The number of routers: $L = 20$

  Delay: the average timesteps taken by packets from source to destination (The smaller, the better)

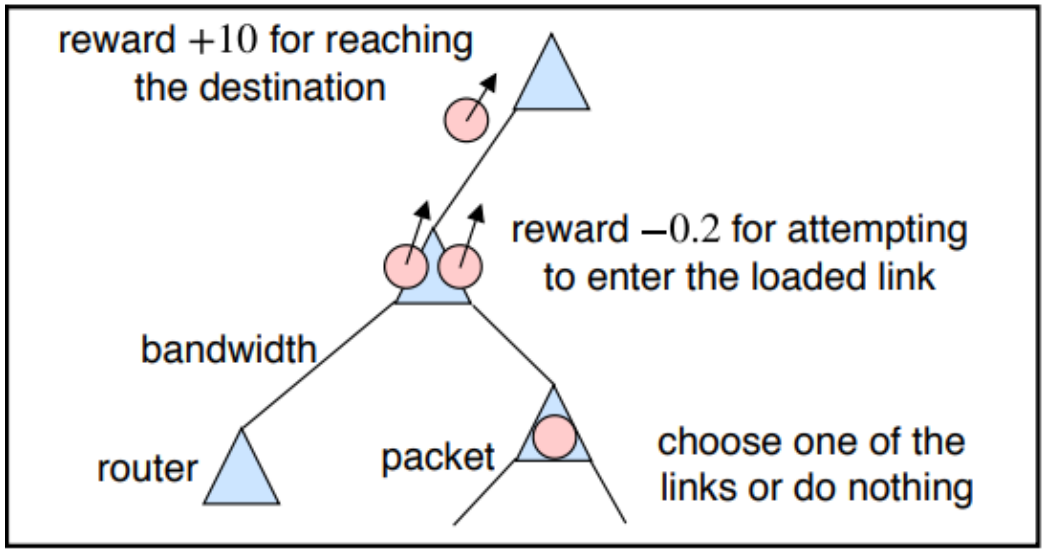
  Throughput: the number of delivered packets per time step (The larger, the better)

Table. evaluation on the Routing scenario

	DGN (in the paper)	DGN (reproduced w/o regularization)	DGN (reproduced)
Mean reward	**1.23**	0.56	**1.23**
Delay	**8.0**	18.74	8.23
Throughput	**2.50**	1.14	2.49

Final Report

- Code Outline
  - class **Router**: define an object ‘Router’ that is randomly connected to a constant number of other routers and become destination for the data packet (agent) to reach goal.
  - class **Edge**: define an object ‘Edge’ where the Routers are randomly connected .
  - class **Data**: define an object ‘Data’ (agent). There are N data packets with a random size, and each packet is randomly assigned a source and destination router. If there are multiple packets with the sum size larger than the bandwidth of a link, they cannot go through the link simultaneously. They aim to quickly reach the destination while avoiding congestion.



Final Report

- Code Outline

  - def **observation**: define a method which makes observation of the data packet. At each time step, 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).
  - def **set_action**: define a method which sets action of the data packet. It takes some time steps for the 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. Once the data packet arrives at the destination, it leaves the system and another data packet enters the system with random initialization.
  - def **Adjacency**: define a method which makes adjacency matrix of the graph.
  - def **MLP**: define a method which models a multi-layer perception.
  - def **MultiHeadsAttModel**: define a method which models a multi heads attention proposed in [3].
  - def **Q_Net**: define a method which models a Q-network. You can find further details in [4].

  ❖ The main code is composed of 1) build the graph, 2) build the model, 3) playing, 4) training.

[3] Zambaldi, Vinicius, et al. "Relational deep reinforcement learning." arXiv preprint arXiv:1806.01830, 2018.

[4] Sutton, Richard S., and Andrew G. Barto. “Introduction to reinforcement learning.”, Vol. 135. Cambridge: MIT press, 1998.