Autonomous Distributed System using Graph Convolutional Network

Koji Fukuda
Hitachi Kyoto University Laboratory,
Center for Exploratory Research, Hitachi Ltd.,
Kyoto, Japan
koji.fukuda.jf@hitachi.com

ABSTRACT

Realizing cooperation is a major goal in the study of a multi- agent system. In this paper we propose a distributed graph convolutional network (D-GCN) that estimates the macroscopic state. It Utilizes uniform consensus algorithms and enables not only the estimation (execution) but also learning to be executed asynchronously and in a distributed manner. In addition, we propose the SwarmCNN, which is a D-GCN consisting of a group of agents moving on a plane. When a SwarmCNN is used, collective animal behaviors such as herd formation arise from only the goal of reducing predation risk without any pre-given fundamental behavior rules.

KEYWORDS

Graph convolutional neural network, Multi-agent reinforcement learning, Distributed learning, Collective animal behavior

1 Introduction

Realizing cooperation is a major goal in study of a multi-agent system consisting of a large number of autonomous agents, such as cooperative control of multiple vehicles, a smart warehouse with cooperation of multiple handling robots, control of traffic signals, smart grid control, *etc.* [1–4]. One method for this is centralized gathering of information of all agents in one place (central server), deriving the optimum behavior of all agents, and notifying each agent of the result [5]. However, as the number of agents increases, gathering information from all agents to the central server and deriving the optimal behavior of all agents become problematic in terms of communication quantity and computational complexity. Moreover, gathering information on all agents in one place may raise privacy and/or security risk depending on the content of the information [6].

Another method is distributed operation that instead of gathering information on all agents in one place has each agent act autonomously while exchanging information between neighboring agents. In recent years there have been some efforts estimate the macroscopic state of the entire system or to optimize the entire behavior of the system in a distributed manner [7–9]. However, none of those efforts has satisfied two requirements important for practical use. First, each agent should operate completely asynchronously without any synchronization mechanism. Second, not only estimation (execution) but also learning should be done in a distributed manner.

In this paper, we propose a distributed graph convolutional network (D-GCN), in which each autonomous agent estimates the macroscopic state of the whole system by asynchronously communicating with neighboring agents. A D- GCN is based on the graph convolutional neural network upon the agent network which represents communicable relation among agents. Utilizing uniform consensus algorithms, a D-GCN can perform not only execution (forward propagation) but also learning (error backpropagation) in a distributed and asynchronous manner.

In addition to that, we propose a SwarmCNN which is a D- GCN applied to a group of agents moving in space. It will be shown that a herd can be formed by a SwarmCNN controlling the movement of agents.

2 Problem Statement

Let us consider a multi-agent system consisting of many agents. There are no central controllers nor synchronization mechanism, and each agent operates in a distributed and asynchronous manner. Each agent receives a localized input from environment, *i.e.*, the input is different for each agent. In addition, each agent has its own internal state.

Assume that each agent can communicate with specific agents. Here, communication means that two agents perform the following two operations:

1. each sending its own internal to the other, and
2. each updating its internal state based on the internal state of the other.

These two processes are assumed to be executed instantaneously. Although the frequency and timing of communication may differ for each edge (pair of agents that can communicates with each other) on the agent network, the communication frequency of every edge is assumed to not be zero.

Let us consider the following two tasks that the entire system should accomplish:

1. Regression (identification) of system evaluation function.
   Each agent ought to estimate (regress) a system evaluation function that depends on all the input describing the local environments of the agents. Each agent performs estimation independently, but the estimated values of all agents must converge to the correct system evaluation function. There is also a variational task that assuming the entire system have several discrete states, each agent ought to identify the current state of the entire system.
2. Optimization of system evaluation function (multi-agent reinforcement learning, MARL).

Consider that each agent takes some actions based on its local environmental input and internal state. It is assumed that the combination of actions taken by all agents affects the future input from the environment. Each agent ought to find the optimum action that maximizes the future value of the system evaluation function. This task is one of the typical forms of multi-agent reinforcement learning (MARL).

3 Related Work

3.1 Graph Convolutional Network (GCN)

In recent years, the graph convolutional network (GCN) has attracted attention as a method of deep learning for graph structures [10–11]. It consists of stacking of multiple graph convolutional layers (*i.e.*, summing up the values of the adjacent nodes multiplied by different weights for each edge type) and then applying a nonlinear activation function such as a ReLU.

Applying GCN to the agent network of a multi agent system is a natural idea. For example, [9] applied GCN to MARL of task 2 and confirmed the effectiveness. Although [9] adopted a paradigm of centralized training and distributed execution, distributed training is desirable for practical use.

3.2 Geometric Convolutional Neural Network

Conventional CNN for the image (grid network) uses, for example, 3×3 kernel filter, where different weights are applied to the pixels in the eight directions of the horizontal and vertical diagonal directions of the center pixel. For a graph convolutional network upon the geometric graph (i.e., graphs drawn in the n-dimensional space), it is a natural idea to change the weight of the convolution filter of the GCN according to the relative geometric relationship between the nodes. In [12], the weight is determined by the gaussian mixture model (GMM) based on the relative position (coordinates) of each node.

3.3 Uniform Consensus Algorithm

In task 1, the estimated value of the system evaluation functions of every agent, which operates in distributed manner, needs to be the same for all agents. Similarly, in order to realize distributed learning in MARL with a GCN, it is necessary for all agents to agree (converge to a uniform consensus) on learning parameters such as shared weights of the GCN.

Let us focus on the following two algorithms for the uniform consensus problem.

3.3.1 Average Consensus Algorithm. A well-known method of forming a uniform consensus is the average consensus algorithm [13–14]. Although there are many variants of the average consensus algorithm, the basic idea is to distribute x_i among agents while keeping the sum of values x_i held in each agent constant. Since the sum of x_i is always constant, the convergence value is the average value of the initial values of x_i. In the simplest average agreement algorithm, when communicating between agent i and agent j, x_i and x_j are changed as follows

$$x_i^{t+1} = x_j^{t+1} = \frac{1}{2}\left(x_i^t + x_j^t\right). \tag{1}$$

3.3.2 Average Consensus by Distributed ADMM. A distributed ADMM (D-ADMM), which executes the alternating direction method of multipliers (ADMM) algorithm in a distributed manner, has been proposed [15–16]. A D-ADMM can solve the optimization problem for all agents by considering the dual problem of the optimization problem. In particular, by using a D-ADMM to solve the following optimization problem, it is possible to calculate the average value $v = 1/N\left(\sum_{i=1}^{N} x_i\right)$ of the N values x_i in a distributed and asynchronous manner

$$v = \underset{\{v\}}{\operatorname{argmin}} \sum_{i=1}^{N} \frac{1}{2}(x_i - v)^2. \tag{2}$$

A concrete calculation is briefly described here. Let v_i be the estimated value of agent i and $J_i = \{j \mid e_{ij}\}$ be the set of agents adjacent to agent i. Agent i holds the primal variable z_j and the dual variable p_j in its internal state, where $j \in J_i$. The agent i receives the current estimated value v_j of the adjacent agent j by communication and updates its own estimated value v_i with the following recurrence formula

$$\begin{aligned} z_j &\leftarrow \frac{v_i + v_j}{2} \\ p_j &\leftarrow p_j + \frac{v_j - v_i}{2} \\ v_i &\leftarrow \frac{1}{1 + \rho\,|K|}\left[x_i + \sum_{k \in K}(p_k + \rho\, z_k)\right]. \end{aligned} \tag{3}$$

where, ρ is a hyper parameter representing the update step size. If the agent network is a connected graph, v_i converges to the correct average value $v = 1/N\left(\sum_{i=1}^{N} x_i\right)$ [16].

4 Method

4.1 Distributed Graph Convolutional Network (D-GCN)

In this section, we propose a distributed graph convolutional network (D-GCN) that carries out task 1, *i.e.*, regression (or identification) of the system evaluation function.

For each agent to estimate the information of the entire system from communication (information exchange) with only a small number of neighboring agents, it is a natural idea to apply the graph CNN on the agent network. We use a relational GCN (R-GCN), which can consider the edge types.

The differences between D-GCN and the graph convolutional reinforcement learning (GCRL) proposed in [9] are the following two points. First, in GCRL, all agents execute synchronous operation, that is, in each layer of graph convolution, all agents synchronously execute a convolution operation based on the output value of the former layer of the adjacent agents. On the other hand, a D-GCN does not have any synchronization mechanism, and each agent operates autonomously in an asynchronous manner. Second,

GCRL adopts a centralized-training-and-distributed-execution paradigm. That is, it operates in distributed manner at the time of execution, but at the time of learning it gathers the information of all agents in one place and performs centralized learning. On the other hand, a D-GCN performs not only execution but also learning in a distributed manner.

First, in order to operate asynchronously, each agent in a D-GCN stores the internal state values of the adjacent agents obtained by communication as part of its own internal state value. When the values of the adjacent agents are necessary, such as in convolution or error backpropagation, the stored values are used. Note that the stored values of the adjacent agents should include not only the output value of the hidden layer necessary for forward propagation but also the error signals required for learning (*i.e.*, error backpropagation). The errors due to the difference between the current value of the adjacent agents and the stored values obtained at the last communication is a penalty inevitable in a fully asynchronous multi-agent system.

Next, in order to execute distributed learning, a distributed algorithm to form consensus among all the agents of the system is necessary. For example, the graph CNN usually shares the weights and bias values of convolution filters over the entire graph. In order to simulate the shared parameters such as weights and bias values of the graph CNN by using a D-GCN, it is necessary to form uniform consensus on learned parameters held by all agents at learning time. Besides, in order to accomplish the regression (identification) task of the system evaluation function (task 1), a fully connected (FC) layer or a global average pooling (GAP) layer is required after convolution layers. To form a uniform consensus among agents it is necessary to realize these layers in a distributed manner.

To form a uniform consensus, we use the two different uniform consensus algorithms mentioned the previous section. First, to realize a FC layer or GAP layer in forward propagation, obtaining the average consensus using a D-ADMM is desirable. In a FC layer, each agent independently multiplies the input value by the weight and adds the bias, and then calculates the average value of all the agents by using a D-ADMM. It should be noted that calculating the average instead of the summation in a FC layer is not a problem for the FC layer because the learned parameters (the weight and the bias) become multiplied by the number of the agents. In a GAP layer, each agent simply calculates the average value of all the agents by using a D-ADMM. In the D-ADMM algorithm, the input (the value of each agent) and the output (the average value of all the agents) are separate variables. Therefore, even if the value of each agent changes, a D-ADMM can continuously output the correct average value.

Next, let us consider how to learn (update) shared parameter values such as weight and bias of convolutional layers in a distributed manner. Parameter learning is performed by error backpropagation, *i.e.*, by subtracting the difference value determined by the error signal and the learning rate from the current value of the parameter. To realize error backpropagation in a distributed manner, each agent calculates the difference value from only the error signals from the upper layer that the agent is in charge of and independently updates the value of the parameter value stored in its own internal state. After that, the final value of the shared parameter is calculated as the average value of all the agents by a uniform agreement algorithm. This calculation is justified by the chain rule of differentiation—the fact that if a single parameter appears multiple times on a computation graph, the error signal for the parameter can be calculated by summation of the error signals calculated separately for each occurrence. Note that calculating the average instead of the summation in error backpropagation is not a problem because the same result will be obtained if the learning rate is multiplied by the number of agents in advance.

The average consensus algorithm is desirable for parameter learning. Since a parameter is updated by a difference in back propagation, if a D-ADMM is used the input value for each agent deviates as time goes on, continuous calculation is impossible. On the other hand, in the average consensus algorithm, the input (the value of each agent) and the output (the average value of all the agents) are the same variables. Since the parameter values held by each agent are simultaneously updated by back propagation and averaged by the average consensus algorithm, continuous learning can be performed.

Algorithm 1 shows, as an example, the calculation of a D-GCN consisting of one convolutional layer (Conv. layer) and one fully connected layer (FC layer). The internal state S_i of agent i and the calculation performed by the agent i when communicating with the adjacent agent j are shown, where ρ is the update step size of the D-ADMM and η is the learning rate. Note that the bias parameters of Conv. layer and FC layer are omitted for simplicity. Agent i can autonomously calculate the final output value $S_i.y_i$ from its local input x_i. It can also autonomously update the parameters $s.W_c^r$, $s.W_c^{self}$ and $s.W_f$ according to the error signal Δy which is common to all agents.

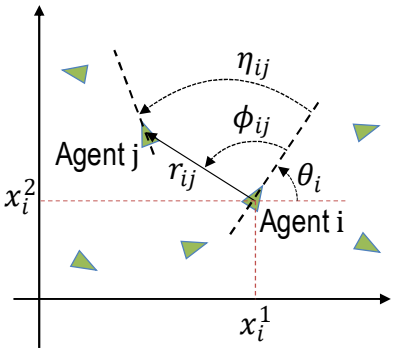


Figure 1: N agents moving on a 2-dimensional space.

Algorithm 1: D-GCN (Conv. Layer → FC Layer)

Struct S_i **contains** : agent internal state of agent i
Let $J = \{j \mid e_{ij}\}$: all adjacent agents of agent i
x : environmental input to agent i
h_c : output value of Conv. layer
y : final output value of FC layer
Δy : error signal (environmental input) for agent i (environmental input)
W_c^r : weight parameter of Conv. layer for edges with relation r
w_c^{self} : weight parameter of Conv. layer for self-loop edges
W_f : weight parameter of FC layer
$\{z_k\}_{k\in K}$: primal variable of D-ADMM for FC layer
$\{p_k\}_{k\in K}$: dual variable of D-ADMM for FC layer
$\{\mathrm{m_k}\}_{\mathrm{k\in K}}$: memorized internal states of adjacent agents
End Struct

Procedure *Communicate* (i, j)
Send S_i **to** Agent j : send internal state to adjacent agent
$S_i.m_j \leftarrow$ **Received** S_j **from** Agent j : memorize internal state of adjacent agent
$s.z_j \leftarrow \big(s.y + S_i.m_j.y\big)/2$: D-ADMM update of primal variable
$s.p_j \leftarrow s.p_j + (\rho/2)\big(S_i.m_j.y - s.y\big)$: D-ADMM update of dual variable
$s.W_c^r \leftarrow \big(s.W_c^r + S_i.m_j.W_c^r\big)/2$: average consensus update
$s.W_c^{self} \leftarrow \big(s.W_c^{self} + S_i.m_j.W_c^{self}\big)/2$: average consensus update
$s.W_f \leftarrow \big(s.W_f + S_i.m_j.W_f\big)/2$: average consensus update
$S_i.x \leftarrow x_i$: update $S_i.x$ to the latest value
$S_i.\nabla y \leftarrow \nabla y$: update $S_i.\nabla y$ to the latest value
ForwardPropagation(S_i) : execute forward propagation
BackwardPropagation(S_i) : execute backward propagation
End Procedure

Subroutine *ForwardPropagation*(S_i)
Let $J = \{j \mid e_{ij}\}$: all adjacent agents of agent i
$S_i.h_c \leftarrow \sigma\big[\sum_{k\in K} W_c^{r_{ik}}(S_i.m_k.x) + W_c^{self}(S_i.x)\big]$: forward propagation of Conv. layer
$h_f = \big(S_i.W_f\big)(S_i.h_c)$: forward propagation of FC layer
$S_i.y \leftarrow \big[h_f + \sum_{k\in K}(S_i.p_k + \rho\, S_i.z_k)\big]/(1+\rho|K|)$: D-ADMM for FC layer
End Subroutine

Subroutine *BackwardPropagation* $(\mathrm{S_i})$
Let $J = \{j \mid e_{ij}\}$: all adjacent agents of agent i
$\nabla h_f = S_i.\nabla y$: chain rule for ∇h_f
$\nabla W_f = (S_i.h_c)\,\nabla h_f$: chain rule for ∇W_f
$\nabla h_c = \big(S_i.W_f\big)\,\nabla h_f$: chain rule for ∇h_c
$\nabla W_c^r = \sum_{k\in K,\, r_{ik}=r} (S_i.m_k.x)\,\sigma'^{(S_i.h_c)}\nabla h_c$: chain rule for ∇W_c^r
$\nabla \mathrm{W_c^{self}} = (\mathrm{S_i.x})\,\sigma'(\mathrm{S_i.h_c})\,\nabla \mathrm{h_c}$: chain rule for ∇W_c^{self}
$s.W_c^r \leftarrow s.W_c^r - \eta\,\nabla W_c^r$: SGD update of parameter W_c^r
$s.W_c^{self} \leftarrow s.W_c^{self} - \eta\,\nabla W_c^{self}$: SGD update of parameter W_c^{self}
$s.W_f \leftarrow s.W_f - \eta\,\nabla W_f$: SGD update of parameter W_f
End Subroutine

4.2 Swarm Convolutional Neural Network (SwarmCNN)

Since in a D-GCN each agent can operate in a distributed and asynchronous manner, the agent network structure, *i.e.*, which agents each agent can communicate with, can change dynamically. For example, each agent moves in a geometric space, and at any point in time, the agent network is configured according to the proximity of the agents. That is, two agents in the vicinity can communicate with each other.

Let us consider a task of estimating a system evaluation function of moving agents by D-GCN. If the system evaluation function depends on the geometric relationship of the agents, it is natural idea to change the weight of the convolution filter of CNN according to the relative position between two agents, similarly to what is done in the geometric convolutional neural network [12]. We call a D-GCN consisting of a moving agent with geometric convolution a swarm convolutional network (SwarmCNN).

Figure 1 shows an example of a SwarmCNN consisting of N agents moving in a 2-dimensional space. Let $\boldsymbol{x} = \{(x_i^1, x_i^2)\}_{i=1,\dots,N}$ be the positions (coordinates) of the agents, and $\boldsymbol{\theta} = \{\theta_i\}_{i=1,\dots,N}$ be its head directions (moving directions). Agent i views agent j at the distance $r_{ij} = \sqrt{\big(x_j^1 - x_i^1\big)^2 + \big(x_j^2 - x_i^2\big)^2}$ and the relative angle $\phi_{ij} = \arctan\left(\frac{x_j^2 - x_i^2}{x_i^2 - x_i^1}\right) - \theta_i$. The relative head direction of agent j as viewed from agent i is $\eta_{ij} = \theta_j - \theta_i$. Let $\mathrm{J_i(r)} = \{\mathrm{j} \mid \mathrm{r_{ij}} \le \mathrm{r}\}$ denote the set of agents within distance r from agent i (excluding agent i itself).

Assuming that each agent can communicate with agents within a distance r_0, the output values $\{f_i\}_{i=1,\dots,N}$ of a convolutional layer are calculated from the input values $\{h_i\}_{i=1,\dots,N}$ as follows

$$f_i = W_{\mathrm{self}} \cdot h_i + \sum_{j\in J_i(r_0)} W\big(r_{ij}, \phi_{ij};\, W_{\mathrm{adj}}\big) \cdot h_{ij} \qquad (4)$$

where W_{self} and W_{adj} are learning parameters, which represent weights for itself and neighboring agents, respectively, and $W\big(r, \phi;\, W_{\mathrm{adj}}\big)$ represents a family of functions parameterized by

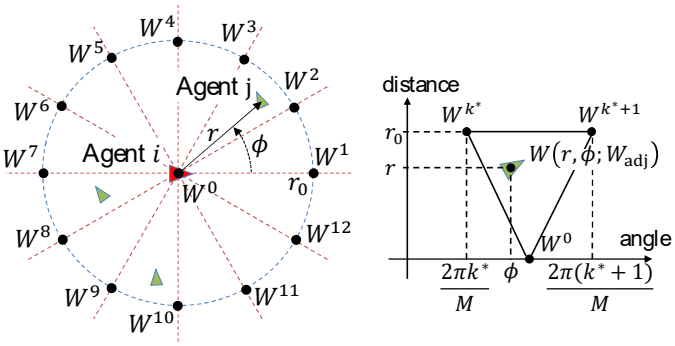


Figure 2: Interpolated convolutional weight based on relative position between two agents (for M = 12).

W^{adj}. By multiplying the input value of the agent j by the weight $W(r_{ji}, \phi_{ij}; W_{\text{adj}})$ depending on the relative position r_{ij} and ϕ_{ij} of agent j viewed from the agent i, the geometric relationship between agents is taken into account. h_{ij} is the value of agent j used for the convolutional operation with respect to agent i. To simply set $h_{ji} = h_j$ is one option. Another option is to set

$$h_{ij} = \text{concat}(h_j, r_{ij}, \phi_{ij}, \eta_{ij}), \quad (5)$$

that is, to reflet the relative position between the two agents as the input value h_{ij} itself.

An example of the function family $W(r, \phi; W_{\text{adj}})$ is the use of interpolation, as shown in Figure 2. Let the learning parameter W^{adj} consist of W^0, which is a weight used when agent j is in the exact same position as agent i, and W^k (where $k = 1,2,\dots,M$), which is used when agent j is at distance r_0 and relative angle $2\pi/\text{M}$ from agent i. Then $W(r, \phi; W^{adj})$ is determined by interpolation from the triangle in r-ϕ space

$$\begin{aligned} &W(r, \phi; W^{adj}) \\ &= \frac{r_0 - r}{r_0} W^0 + \frac{Mr}{2\pi r_0}\left[\left(\frac{2\pi(k^*+1)}{M} - \phi\right) W^{k^*}\right. \\ &\qquad \left. + \left(\phi - \frac{2\pi k^*}{M}\right) W^{k^*+1}\right] \end{aligned} \quad (6)$$

where k^* is an integer satisfying $2\pi k^*/\text{M} \le \phi < 2\pi(k^*+1)/\text{M}$.

4.3 MARL using SwarmCNN with action gradients

SwarmCNN consists of moving agents with geometric convolution. Furthermore, let us consider that the movements of the agents are controlled by SwarmCNN itself; that is, each agent moves in a way based on the output value of the SwarmCNN that it belongs to.

According to the standard framework of reinforcement learning, let SwarmCNN estimate the action value function $\text{Q}(s, a)$, where s is the state of all the agents and a is all the combinations of actions that the agents can take. Here we assume that the action a of the agents is not a discrete but a continuous value.

We adopt an on-policy reinforcement learning; that is, while learning SwarmCNN to estimate the action value function $\text{Q}(s, a)$, at the same time, the next action a_{t+1} of the agents is determined according to the current value $\text{Q}(s_t, a)$ of the state s_t of the agents at the time t. An on-policy reinforcement learning often uses the ε-greedy method to determine the next action. However, it is difficult to determine the greedy action $\text{argmax}_a\, \text{Q}(s_t, a)$ when the number of agents (*i.e.*, the dimension of action a) is large.

By taking advantage of the fact that the action a is a continuous value and SwarmCNN which represents $Q(s, a)$ is differentiable, we propose an action gradient method to determine the next action a_{t+1} of the agents. This method updates action a by the gradient ascent of the action value function $\text{Q}(s, a)$ at the time t

$$a_{t+1} = a_t + \lambda \left.\frac{\partial \text{Q}(s, a)}{\partial a}\right|_{s=s_t,\ a=a_t} \quad (7)$$

where λ is a hyper parameter representing the step size of the action update.

Table 1: Layer configuration of D-GCN.

	Layer	Structure	channels
0	input	28x28	1
1	Conv1+ReLU	28x28	10
2	MaxPool	28x28	10
3	Conv2+ReLU	28x28	5
4	MaxPool	28x28	5
5	FC (Linear)+SoftMax	10 (one-hot)	

5 Experiments

5.1 MNIST digit recognition with D-GCN

First, we verified the effectiveness of a D-GCN in an image recognition task with the MNIST handwritten digit dataset [17]. It consists of 50,000 images for training and 10,000 images for test (cross validation). Each image is a gray-scale image of 28×28 pixels on which one handwritten digit symbol is drawn.

The experiments were conducted on a D-GCN consisting of $28 \times 28 = 784$ agents, where each agent was responsible for one pixel. While the input of each agent was only the value of the pixel that it is responsible for, each agent had to identify the number drawn on the whole image of 28×28 pixels by exchanging information with the eight neighboring agents. This setting is, for example, analogous to the situation that people on the ground identify what is drawn on "Nazca's landscapes". The final answer must be agreed by all participants. Reaching a consensus is probably not easy when information is exchanged by people who cannot move and can see only the ground directly beneath them.

The agent network of the D-GCN consisted of 784 nodes (agents), and 2,970 edges (pairs of agents that can communicate with each other). Each edge was annotated with one of eight kinds of relations depending on its direction. The layer configuration used in the experiment is shown in Table 1.

The D-GCN was trained in an online setting with handwritten images switched one after another. The input to each agent arranged in a grid was the pixel value corresponding to the position on the input image. In addition, the correct identification result, which was the same for all agents, was presented to each agent as a teacher signal.

In a D-GCN, communication between adjacent agents (and updating of the internal state) can be executed asynchronously. To simulate the asynchronous operation, we prepared an array of length $2{,}970 \times 5 = 14{,}850$ which stored five sets of 2,970 edge IDs in random order. Then, communications between adjacent agents were performed in the order stored in this array. Once all the edges (communications between pairs of agents) in the array had been processed, the array was shuffled in order again. Hereinafter, this series of 14,850 operations will be referred to as one frame. That is, during one frame, the communication between all the 2,970 adjacent pairs of agents and following internal state update of the agents were performed five times respectively in random order. In the experiment, one handwritten digit image was input to the D-GCN for 30 frames and then switched to another, and so on.

Each agent independently calculated the cross-entropy loss from the teacher signal and its own output value at the time and updated the parameter to decrease the loss. Adaptive moment estimation (ADAM) with the learning rate lr=1e−6 was used for parameter updating.

Figure 3 shows the temporal decrease of the average cross-entropy loss for all the agents. The horizontal axis of the figure represents the number of images shown. As described above, one image corresponds to 30 frames. Since the training set of MNIST consists of 50,000 images, we learned for less than 6 epochs (30,000 images). Learning could not be done further due to limitation of calculation time. The cross-entropy loss jumped at the moment when the image was switched every 30 frames, and then it decreased as a new consensus was formed among the agents. In Figure 3, moving averages were taken with a window width of 3000 frames (100 images) to exclude the fluctuation every 30 frames.

The accuracy rate of identification using the learned parameters was 93.6% for the test set. It did not reach the expected rate of 97.3%, which is the accuracy rate with the optimum parameters. As is clear from Figure 3, this was mainly because parameter learning (optimization) had not yet converged. In addition, hyperparameters such as learning rate might not be the optimal values. First, we tried to set learning rate as 1e−4, which is a value often-used in ADAM, but learning hardly proceeded. In this experiment, since the parameters are updated for each communication, parameter updating is performed 150 times during one image (= 30 frames). The learning rate must be set to a considerably small value to learn parameters properly.

Figure 4 shows the experiment of handwritten digit recognition by D-GCN. The handwritten digits displayed in the background of the figure indicate each pixel value which is the only input to each agent. The small number displayed on each pixel is the recognition result of each agent at that point. The frame number and the correct answer of the handwritten digit image presented at that time are shown at the upper right of each figure.

At the beginning (frame 0), the output values from each agent did not match at all. As frame number increased, a consensus was formed among agents, and most agents outputted the correct answer **7**. Although the consensus was destroyed when the input image suddenly switched to **2** at frame 30, as time (frame number) elapsed, all the agents again outputted correct results.

Note that the consensus formation started in agents near the center of the image and spread around the peripheral part because the pixels near the center of the image had more information on the handwritten digit.

5.2 Collective behavior with SwarmCNN

As an application of a SwarmCNN, let us consider the collective animal behavior under predation risk. The “selfish herd” theory proposed by Hamilton is a pioneering study of this field [18]. He thought that the formation of herds is due to dilution of the predation risk by aggregation. That is, each individual acts to reduce its domain of danger (DOD), which is the area closest to the individual. Known concrete behavior rules of each individual to reduce DOD includes, for example, Nearest Neighbor (NN), Multiple NNs (3NN), Local Crowded Horizon (LCH), *etc.* [18–20]. In recent years, studies on what type of herd is formed under each behavioral rule are actively conducted [21–22].

On the other hand, in the field of computer vision, Raynolds proposed well-known BOIDs rules [23]. The behavior of various herds can be reproduced by changing the intensity of the three rules of separation, alignment and cohesion. Recently, attempts have been made to optimize the intensities of the three rules by genetic algorithm to form herds under predation risk [24]. In these approaches, some fundamental behavioral rules are given in advance and the optimal combination of these rules is discussed. In this section, we try using SwarmCNN to derive the behavior rules from only the goal of reducing DOD.

Let us consider a discrete-time system consisting of N agents (fishes) moving on a 2-dimensional plane as shown in Figure 1. The moving speed of all agents is fixed at v=0.003/timestep, and each agent can change the moving direction (head direction) only at each time step. Suppose that the agent i suffers a penalty $L_i(\boldsymbol{x},\boldsymbol{\theta})$ depending on the position $\boldsymbol{x}$ of the agents and the head directions $\boldsymbol{\theta}$. Specifically, the penalty $L_i(\boldsymbol{x},\boldsymbol{\theta})$ is assumed to be the sum two terms $L_i^P(\boldsymbol{x},\boldsymbol{\theta})$ and $L_i^G(\boldsymbol{x},\boldsymbol{\theta})$, depending on the relative positions of the neighboring agents. When the agents move to minimize these penalties, a herd is formed.

1. Proximity penalty.
   A collision risk arises if agents are too close together. If there exist other agents closer than a threshold distance r_1, agent i is penalized according to the distance to the other agents
   $$L_i^p(\boldsymbol{x},\boldsymbol{\theta}) = \sum_{j\in J_i(r_1)} \left(\frac{r_1}{r^\alpha} - 1\right) \tag{8}$$
   where $\alpha > 0$ is a given parameter.
2. Gap penalty.
   A predation risk arises if an agent (fish) is not surrounded by other agents. Arranging agents $j \in J_i(r_2)$ within distance r_2 from agent i in ascending order of the relative angle ϕ_{ij}, agent i is penalized according to the gap angle $d_{j_1} = \phi_{ij_2} - \phi_{ij_1}$ between adjacent agents j_1 and j_2
   $$L_i^G(\boldsymbol{x},\boldsymbol{\theta}) = \sum_{j\in J_i(r_2)} L_d(d_j) \tag{9}$$
   where
   $$L_d(d) = \begin{cases} 0 & \text{if } d \le d_0 \\ (d-d_0)^\beta & \text{if } d \ge d_0 \end{cases} \tag{10}$$
   is a gap penalty function with given parameters $d_0 > 0$ and $\beta > 0$. That is, if there exists another agent within the distance r_2, agent i need not worry about predators for the angle range of d_0 around the direction to that agent. This penalty, which corresponds to the concept of Hamilton's DOD, is convenient because it always has a finite value.

In the experiment, MARL using a SwarmCNN with the action gradient method was used to form a herd. A SwarmCNN consisting of all the agents was being learned to estimate the penalty value of each agent. At the same time, the head direction of each agent was changed by the action gradient method based on the SwarmCNN.

Actually, the SwarmCNN estimates the discounted sum of future penalty $\mathcal{L}_{\mathrm{i}}(\mathrm{t}) = \sum_{\tau=0}^{\infty} \gamma^{\tau} L_i(\boldsymbol{x_t}, \boldsymbol{\theta_t})$ of each agent, where $0 < \gamma < 1$ is a discount factor.

Figure 5 shows the architecture of SwarmCNN used in the experiment. It consists of three convolutional layers. The inputs to the SwarmCNN were a one-hot vector $\boldsymbol{k} = \{k_i\}$ representing the kind of each agent (if all agents are the same kind, $k_i = 1$ for all agents) and the relative position $\{r_{ij}\}$, $\{\phi_{ij}\}$ and the relative head direction $\{\eta_{ij}\}$ between the agents, while the output was the future discount sum of the penalty value of each agent. The input to the first convolutional layer was the concatenation of four values k_j, r_{ij}, ϕ_{ij} and η_{ij} as Equation (5), while the input of the second and the third convolutional layer was simply is the output value of the previous layer. In addition, since the task is to estimate not a single value but N penalty values for every agent, the final outputs were calculated using JK-Nets (concat) decoder, which concatenates the value of each layer and performs linear conversion [25].

The SwarmCNN was learned by the state-action-reward-state-action (SARSA) method with action gradient method. The learning was performed purely on-policy without the experiment replay. To stabilize learning, fixed target Q-network was used, where the target network was continuously updated by taking running average with momentum of 1e−3.

In parallel with learning of the SwarmCNN, each agent changed the head direction by action gradient of the SwarmCNN at that time. Actually, the head direction of each agent was changed to decrease the average of estimated future penalties of all agents. Note that all agents ware assumed to cooperate unlike selfish herd problem. Table 2 shows the list of parameter values used for the experiment.

Figure 6 shows the result of simulation with 100 fish agents on 1×1 sized area with periodic (torus) boundary. Initially, all fishes ware swimming randomly (Figure 6a). They got closer to reduce the gap penalty, but they suffer the proximity penalty because they got too close to each other (Figure 6b). Finally, they formed a herd by keeping an appropriate distance from each other (Figure 6c). Individual fish agents were moving complicatedly, such as moving in opposite directions in the core and peripheral part of the herd.

Figure 7 shows the results with surrounding walls instead of periodic boundaries. When an agent approaches the walls, it suffers a penalty of the same form as the proximity penalty. To make the walls recognizable by the fish agents (gray), a total of 156 fixed (non-moving) "buoy" agents (blue) were placed at intervals of 0.025 along the walls. The $\{k_i\}$ values of the buoy agents were different from the fish agents. The buoy agents could be recognized by the fish agents but did not participate in the SwarmCNN of the fish agents. That is, they existed only as the inputs of the SwarmCNN and did not communicate with other agents. Finally, the fish agents formed a herd while avoiding the walls.

Figure 8 shows the result if 10 predator agents (red) existed in addition to 100 fish agents (gray). The predator agents traveled in a straight line at constant speed 0.015 /timestep. The fish agent suffered a large proximity penalty as it approached the predator agents. Like the buoy agents, the predator agents were recognizable from the fish agent but did not participate in the SwarmCNN of the fish agents. In the figure, it is seen that the fish agent makes a herd while avoiding the predator agent.

6 Conclusion

We proposed a distributed graph convolutional network (D-GCN) for multi-agent systems, which estimates the macroscopic state of the entire system in a distributed and asynchronous manner. Although each agent operates autonomously while communicating with neighboring agents, the estimates of every agent coincide with the correct value. A D-GCN utilizes uniform consensus algorithms, not only the estimation (execution) but also learning can be executed asynchronously and in a distributed manner.

In addition to that, the SwarmCNN, which is a D-GCN consisting of a group of agents moving on a plane, was proposed. When a SwarmCNN is used, collective animal behaviors such as herd formation arises from only the goal of reducing predation risk, without any pre-given fundamental behavior rules.

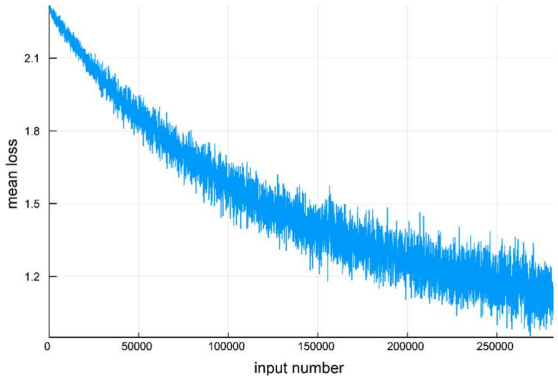


Figure 3: The average of the cross-entropy loss for all agents. (Applied moving average for 3000 frames)

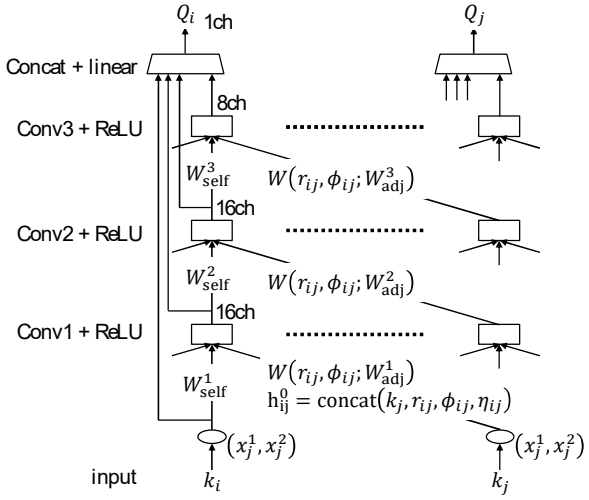


Figure 5: Architecture of SwarmCNN.

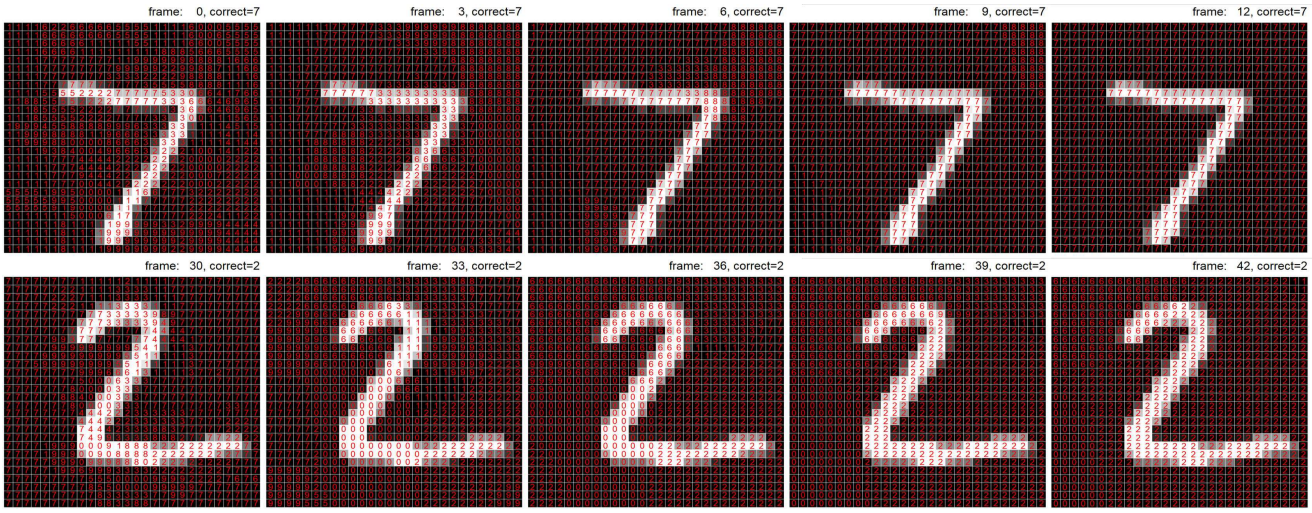
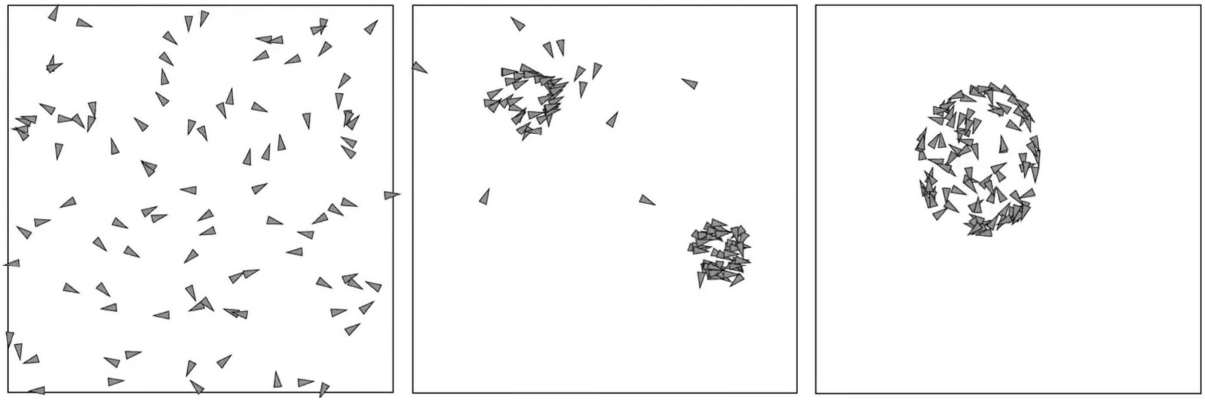


Figure 4: Digit recognition by D-GCN.



(a) Initial state **(b) Intermediate state** **(c) Final state**

Figure 6: Herd formation by SwarmCNN.

Table 2: Parameter values for SwarmCNN.

Parameter	value
M (Conv. layer)	12
r_0 (Conv. layer)	0.1
λ (action gradient updating)	5.0
r_1 (Proximity penalty)	0.01
α (Proximity penalty)	1.0
r_2 (Gap penalty)	0.1
d_0 (Gap penalty)	$\pi/6$
β (Gap penalty)	1.5
optimizer algorithm	ADAM
learning rate	1e-5
velocity/timestep (fish)	0.003
predetor's velocity/timestep	0.0015
r_1^P (Proximity penalty for predetor)	0.2
α^P (Proximity penalty for predetor)	1.0
r_1^W (Proximity penalty for wall)	1.5
α^W (Proximity penalty for wall)	1.5

Figure 7: Herd formation with surrounding walls.

grey: fish agents
blue: buoy agents

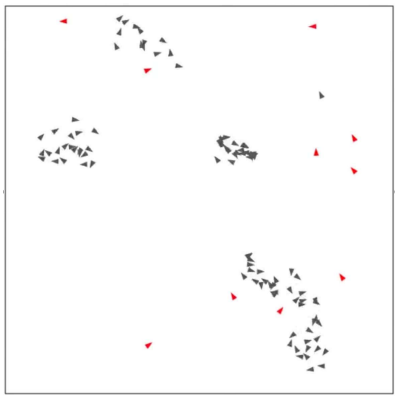


Figure 8: Herd formation with predators.

grey: fish agents
red: predator agents

REFERENCES

[1] W. Ren and R.W. Beard, (2008). *Distributed Consensus in Multi-vehicle Cooperative Control: Theory and Applications*. Springer-Verlag London. DOI: http://dx.doi.org/10.1007/978-1-84800-015-5.

[2] S. Traba, E. Bajic, A. Zouinkhi, M.N. Abdelkrima, H. Chekir, and R.H. Ltaief (2015). *Product Allocation Planning with Safety Compatibility Constraints in IoT-based Warehouse*. Procedia Computer Science, 73, 290–297.

[3] K.J. Prabuchandran, A.N. Hemanth Kumar, and S. Bhatnagar. (2014). *Multi-agent Reinforcement Learning for Traffic Signal Control. Proc*. of IEEE International Conference on Intelligent Transportation Systems (ITSC) 2014, 2529–2534.

[4] H. Kawashima, T. Kato, and T. Matsuyama. (2013). *Distributed Mode Scheduling for Coordinating Power Balancing.* Proc. of IEEE International Conference on Smart Grid Communications (SmartGridComm) 2013, 19–24.

[5] X. Yi, W. Lu, and T. Chen. (2016). *Centralized event-triggered control for linear multi-agent systems.* Proc. of Chinese Control and Decision Conference (CCDC) 2016, 225–230.

[6] A. Abdallah and X. Shen, 2018. *Security and Privacy in Smart Grid*. Springer International Publishing. DOI: http://dx.doi.org/10.1007/978-3-319-93677-2.

[7] Y. Cao, W. Yu, W. Ren, and G. Chen. (2017). *An Overview of Recent Progress in the Study of Distributed Multi-Agent Coordination*. (2019). IEEE Transactions on Industrial Informatics, 9(1), 427–438.

[8] S. Miura and J. Miyakoshi. (2017). *New interaction in autonomous decentralized control method for future social systems.* Proc. of International Symposium on Control Systems (SICE ISCS) 2017, 33–40.

[9] J. Jiang, C. Dun, and Z. Lu. (2019), *Graph Convolutional Reinforcement Learning for Multi-Agent Cooperation.* https://arxiv.org/abs/1810.09202.

[10] T.N. Kipf and M. Welling. (2017). *Semi-Supervised Classification with Graph Convolutional Networks*, International Conference on Learning Representation (ICLR) 2017, https://arxiv.org/abs/1609.02907.

[11] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Berg, I. Titov, and M. Welling. (2017), *Modeling Relational Data with Graph Convolutional Networks.* https://arxiv.org/abs/1703.06103.

[12] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. (2017). *Geometric deep learning on graphs and manifolds using mixture model CNNs.* In Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017, 5425–5434.

[13] S. S. Kia, B.V. Scoy, J. Cortes, R. A. Freeman, K. M. Lynch, and S. Martinez (2017), *Tutorial on dynamic average consensus: the problem, its applications, and the algorithms.* https://arxiv.org/abs/1803.04628.

[14] Dhuli, Sateeshkrishna and Singh, Yatindra. (2018). Analysis of Average Consensus Algorithm for Asymmetric Regular Networks. https://arxiv.org/abs/1806.03932.

[15] E. Wei and A. Ozdaglar. (2013). *On the O(1/k) Convergence of Asynchronous Distributed Alternating Direction Method of Multipliers*. Proc. of IEEE Global Conference on Signal and Information Processing (GlobalSIP), 2013, 551–554.

[16] E. Wei and A. Ozdaglar. (2013). *On the O(1/k) Convergence of Asynchronous Distributed Alternating Direction Method of Multipliers*. https://arxiv.org/abs/1307.8254.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. (1998). *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 86(11), 2278–2324.

[18] W.D. Hamilton. (1971). *Geometry for the Selfish Herd.* Journal of Theoretical Biology, 31(2), 295–311.

[19] T. L. Morton, J. W. Haefner, V. Nugala, R. D. Decino, and L. Mendes. (1994). *The selfish herd revisited: Do simple movement rules reduce relative predation risk?*. Journal of Theoretical Biology, 167(1), 73–79.

[20] S. V. Viscido and D. S. Wethey. (2001). *Quantitative analysis of fiddler crab flock movement: evidence for 'selfish herd' behaviour.* Animal Behaviour, 63(4), 735–741.

[21] L. J. Morrell, G.D. Ruxton, and R. James, (2011), *Spatial positioning in the selfish herd*, Behavioral Ecology, 22(1), 16–22.

[22] G.M. Rodgers, B. Downing, and L.J. Morrell, (2015), *Prey body size mediates the predation risk associated with being "odd".* Evolutionary Ecology, 26(1), 242–246.

[23] C.W. Reynolds. (1987). *Flocks, herds and schools: A distributed behavioral model.* In Proceedings of the 14th annual conference on Computer graphics and interactive techniques (SIGGRAPH) 1987. 25–34.

[24] M. Wagner, W. Cai, and M. H. Lees. (2013). *Emergence by strategy: flocking boids and their fitness in relation to model complexity.* In Proceedings of the 2013 Winter Simulation Conference. IEEE. 1479–1490.

[25] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. (2018). *Representation Learning on Graphs with Jumping Knowledge Networks.* Proceedings of the 35th International Conference on Machine Learning (ICML), PMLR 80:5453–5462.