2019 International Symposium on Nonlinear Theory and Its Applications
NOLTA2019, Kuala Lumpur, Malaysia, December 2-6, 2019.

New Rules for Choosing Values of Consensus Weights in Distributed Training of Neural Networks

Kazuaki Harada, Tsuyoshi Migita and Norikazu Takahashi

Graduate School of Natural Science and Technology, Okayama University
3–1–1 Tsushima-naka, Kita-ku, Okayama 700–8530, Japan
Email: harada@momo.cs.okayama-u.ac.jp, migita@cs.okayama-u.ac.jp, takahashi@cs.okayama-u.ac.jp

Abstract—A framework for distributed training of neural networks was recently proposed. In this framework, each neural network computes its solution based on the partial knowledge about the training data and then updates the solution by communicating with each other using some hyper parameters known as the consensus weights. In this paper, we propose new rules for choosing the values of the consensus weights, and show experimentally that the proposed rules can achieve faster convergence.

1. Introduction

Distributed optimization over multiagent networks has attracted a great deal of attention in the past two decades [1–4]. A typical situation considered in the literature of distributed optimization is that an objective function expressed as the sum of finite functions is given to agents, but each agent knows only one of them. In this situation, agents have to find an optimal (or a locally optimal) solution cooperatively through local optimization for their own functions and local interaction between neighboring agents to achieve a consensus. Many algorithms for distributed optimization have been proposed so far, and proven to converge under certain assumptions.

Recently, Scardapane and Di Lorenzo [5] proposed a framework for distributed training of multiple neural networks (NNs) by extending the NEXT algorithm [2] developed for the distributed optimization. In this framework, multiple NNs with the same structure try to find a common set of parameter values that minimizes the loss for a given training dataset, but the dataset is divided into nonoverlapping subsets and each subset is given to only one NN. Thus NNs cooperatively solve the problem through local optimization and local interaction. Scardapane and Di Lorenzo [5] proved the convergence of their algorithms under some assumptions on the loss function, the graph representing the interaction among NNs, the *consensus weights* used in local interaction, and so on, and showed some experimental results. However, their assumptions on the consensus weights seem too strong. If these assumptions can be relaxed, we can choose the values of the consensus weights more flexibly to increase the speed of convergence.

In this paper, we propose new rules for choosing the values of the consensus weights. The basic idea behind these rules is that larger weights should be assigned to those NNs communicating with many NNs. We show experimentally that the proposed rules not only are convergent like the conventional rules but also achieve a faster convergence speed.

2. Distributed Training of NNs

In this section, we review the distributed training of NNs proposed by Scardapane and Di Lorenzo [5].

2.1. Problem Formulation

Suppose that a given set $\mathcal{S}$ of training data is divided into I nonoverlapping subsets, and each subset is assigned to one of I NNs that communicate with each other. The i-th subset of $\mathcal{S}$ is denoted by $\mathcal{S}_i = \{(\boldsymbol{x}_{i,m}, d_{i,m})\}_{m=1}^{N_i}$ where $\boldsymbol{x}_{i,m} \in \mathbb{R}^d$ and $d_{i,m} \in \mathbb{R}$ for $m = 1, 2, \ldots, N_i$. Note that the i-th NN only knows $\mathcal{S}_i$ and cannot access $\mathcal{S}_j$ $(j \neq i)$. We assume that all NNs have the same structure, that is, they have the same number of layers, the same number of neurons in each layer, only one neuron in the output layer, the same activation functions, and so on. Then the output of the i-th NN for the input vector $\boldsymbol{x} \in \mathbb{R}^d$ can be expressed as $f(\boldsymbol{w}_i; \boldsymbol{x})$ where $\boldsymbol{w}_i \in \mathbb{R}^Q$ is the vector composed of all adjustable parameters.

Assumption 1 The NN model $f(\boldsymbol{w}; \boldsymbol{x})$ satisfies the following conditions: i) f is continuously differentiable with respect to $\boldsymbol{w}$, and ii) f is Lipschitz continuous with respect to $\boldsymbol{w}$, that is, there exists a Lipschitz constant $L \geq 0$ such that $\|\nabla_{\boldsymbol{w}} f(\tilde{\boldsymbol{w}}; \boldsymbol{x}) - \nabla_{\boldsymbol{w}} f(\hat{\boldsymbol{w}}; \boldsymbol{x})\|_2 \leq L\|\tilde{\boldsymbol{w}} - \hat{\boldsymbol{w}}\|_2$ where $\|\cdot\|_2$ represents the Euclidean norm.

The distributed training of NNs is formulated as the following optimization problem:

$$\begin{array}{ll} \text{minimize} & U(\boldsymbol{w}) = \frac{1}{I}\sum_{i=1}^{I}\left(\sum_{j=1}^{I} g_j(\boldsymbol{w}_i) + r(\boldsymbol{w}_i)\right) \\ \text{subject to} & \boldsymbol{w}_1 = \boldsymbol{w}_2 = \cdots = \boldsymbol{w}_I \end{array} \tag{1}$$

where $\boldsymbol{w} = (\boldsymbol{w}_1^{\mathrm{T}}, \boldsymbol{w}_2^{\mathrm{T}}, \ldots, \boldsymbol{w}_I^{\mathrm{T}})^{\mathrm{T}} \in \mathbb{R}^{QI}$,

$$g_j(\boldsymbol{w}_i) = \sum_{m=1}^{N_j} \ell\left(d_{j,m}, f(\boldsymbol{w}_i; \boldsymbol{x}_{j,m})\right)$$

with ℓ being a loss function, and $r(\boldsymbol{w}_i)$ is a regularization term. Note that the i-th NN cannot compute $g_j(\boldsymbol{w}_i)$ $(j \neq i)$

because $g_j(\cdot)$ is determined by $\mathcal{S}_j$ and the i-th NN cannot access it.

Assumption 2 The optimization problem (1) satisfies the following conditions: i) ℓ is convex, continuously differentiable, and Lipschitz continuous, ii) r is either a) convex, continuously differentiable and Lipschitz continuous, or b) nondifferentiable convex function with bounded subgradients, and iii) U is coercive, that is, $\lim_{\|\boldsymbol{w}\|\to\infty} U(\boldsymbol{w}) = \infty$.

2.2. Communication among NNs

Throughout this paper, we assume for simplicity that the communication among NNs is time-invariant, though the time-varying case is considered in [5]. It is thus expressed as a time-invariant directed graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{1, 2, \ldots, I\}$ is the set of vertices, each of which corresponds to an NN, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of directed edges. $\mathcal{E}$ contains (j, i) if and only if $i \neq j$ and the i-th NN can receive information from the j-th NN. The in-neighborhood and out-neighborhood of vertex $i \in \mathcal{V}$ are defined as $\mathcal{N}_i^{\text{in}} = \{j \,|\, (j, i) \in \mathcal{E}\} \cup \{i\}$ and $\mathcal{N}_i^{\text{out}} = \{j \,|\, (i, j) \in \mathcal{E}\} \cup \{i\}$, respectively.

2.3. Algorithm

In the distributed training of NNs [5], each NN iterates alternating two operations: local optimization and consensus update. Let $\boldsymbol{w}_i[n]$ be the solution obtained by the i-th NN in the n-th iteration, and $\tilde{\boldsymbol{\pi}}_i[n]$ be an estimate of $\sum_{j=1, j\neq i}^{I} \nabla_{\boldsymbol{w}_i} g_j(\boldsymbol{w}_i[n])$ (details will be explained later). In the operation of local optimization, each NN first solves the optimization problem:

$$\text{minimize} \;\; \tilde{g}_i(\boldsymbol{w}_i; \boldsymbol{w}_i[n]) + \tilde{\boldsymbol{\pi}}_i[n]^{\mathrm{T}}(\boldsymbol{w}_i - \boldsymbol{w}_i[n]) + r(\boldsymbol{w}_i) \quad (2)$$

where $\tilde{g}_i(\boldsymbol{w}_i; \boldsymbol{w}_i[n])$ is a strongly convex surrogate function of $g_i(\boldsymbol{w}_i)$ at $\boldsymbol{w}_i = \boldsymbol{w}_i[n]$. In this paper, we use a simple surrogate function obtained from g_i by linearizing it around $\boldsymbol{w}_i = \boldsymbol{w}_i[n]$ and adding a proximal regularization term:

$$\tilde{g}_i(\boldsymbol{w}_i; \boldsymbol{w}_i[n]) = g_i(\boldsymbol{w}_i[n]) + \nabla g_i(\boldsymbol{w}_i[n])^{\mathrm{T}}(\boldsymbol{w}_i - \boldsymbol{w}_i[n]) + \frac{\tau}{2}\|\boldsymbol{w}_i - \boldsymbol{w}_i[n]\|_2^2 \quad (3)$$

where τ is a positive constant. Let $\tilde{\boldsymbol{w}}[n]$ be the solution of (2). Each NN then computes $\boldsymbol{z}_i[n]$ by

$$\boldsymbol{z}_i[n] = \boldsymbol{w}_i[n] + \alpha[n](\tilde{\boldsymbol{w}}_i[n] - \boldsymbol{w}_i[n]) \quad (4)$$

where $\{\alpha[n]\}_{n=0}^{\infty}$ is a sequence determined by $\alpha[n] = \alpha[n-1](1 - \epsilon\alpha[n-1])$ with $\alpha[0], \epsilon \in (0, 1]$. In the operation of consensus update, each NN computes $\boldsymbol{w}_i[n+1]$, $\boldsymbol{y}_i[n]$ and $\tilde{\boldsymbol{\pi}}_i[n+1]$ by

$$\boldsymbol{w}_i[n+1] = \sum_{j\in\mathcal{N}_i^{\text{in}}} c_{ij}\boldsymbol{z}_j[n], \quad (5)$$

$$\boldsymbol{y}_i[n+1] = \sum_{j\in\mathcal{N}_i^{\text{in}}} c_{ij}\boldsymbol{y}_j[n] + \nabla g_i(\boldsymbol{w}_i[n+1]) - \nabla g_i(\boldsymbol{w}_i[n]), \quad (6)$$

$$\tilde{\boldsymbol{\pi}}_i[n+1] = I\boldsymbol{y}_i[n+1] - \nabla g_i[n+1], \quad (7)$$

Algorithm 1 Distributed Optimization for Problem (1)

Input: $g_i, r : \mathbb{R}^Q \to \mathbb{R}$, G, $\{c_{ij}\}_{j\in\mathcal{N}_i^{\text{in}}}$, τ, $\alpha[0]$, ϵ

Output: $\boldsymbol{w}_i \in \mathbb{R}^Q$

1: Set $\boldsymbol{w}_i[0]$ to a randomly chosen vector. Set $\boldsymbol{y}_i[0] \leftarrow \nabla g_i(\boldsymbol{w}_i[0])$, $\tilde{\boldsymbol{\pi}}_i[0] \leftarrow I\boldsymbol{y}_i[0] - \nabla g_i(\boldsymbol{w}_i[0])$ and $n \leftarrow 0$.
2: If the stopping condition is satisfied then return $\boldsymbol{w}_i[n]$ and stop. Otherwise go to Step 3.
3: Set $\tilde{\boldsymbol{w}}_i[n]$ to the optimal solution of (2) with (3) and compute $\boldsymbol{z}_i[n]$ by (4).
4: Compute $\boldsymbol{w}_i[n+1]$, $\boldsymbol{y}_i[n+1]$ and $\tilde{\boldsymbol{\pi}}_i[n+1]$ by (5), (6) and (7), respectively.
5: Set $n \leftarrow n+1$ and go to Step 2.

respectively, where c_{ij} are nonnegative constants which we call the consensus weights in this paper. We should note that $\boldsymbol{y}_i[n]$ and $\tilde{\boldsymbol{\pi}}_i[n]$ are estimates of $\frac{1}{I}\sum_{j=1}^{I} \nabla_{\boldsymbol{w}_i} g_j(\boldsymbol{w}_i[n])$ and $\sum_{j=1, j\neq i}^{I} \nabla_{\boldsymbol{w}_i} g_j(\boldsymbol{w}_i[n])$, respectively (see [5] for details).

A pseudocode of the algorithm is shown in Algorithm 1.

Proposition 1 (Simplified version of Proposition 2 in [5]) Suppose that i) Assumptions 1 and 2 hold, ii) $G = (\mathcal{V}, \mathcal{E})$ is strongly connected, iii) the consensus weights c_{ij} are positive and satisfy the following conditions:

$$\sum_{j\in\mathcal{N}_i^{\text{in}}} c_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, I\}, \quad (8)$$

$$\sum_{j\in\mathcal{N}_i^{\text{out}}} c_{ji} = 1, \quad \forall i \in \{1, 2, \ldots, I\}, \quad (9)$$

iv) $\{\alpha[n]\}_{n=0}^{\infty}$ is chosen so that $\alpha[n] \in (0, 1]$ for all n and $\sum_{n=0}^{\infty} \alpha[n] = \infty$. Then all limit points of the sequence $\{(\boldsymbol{w}_1[n]^{\mathrm{T}}, \boldsymbol{w}_2[n]^{\mathrm{T}}, \ldots, \boldsymbol{w}_I[n]^{\mathrm{T}})^{\mathrm{T}}\}_{n=0}^{\infty}$ generated by Algorithm 1 are stationary solutions of (1).

3. How to Choose Values of Consensus Weights

In this section, we focus our attention on how to choose the values of consensus weights c_{ij}. We hereafter assume for simplicity that the communication among NNs is symmetric, that is, $(i, j) \in \mathcal{E}$ if and only if $(j, i) \in \mathcal{E}$.

3.1. Conventional Rules

Simple rules often used in consensus algorithms [1] are given by

$$c_{ij} = \begin{cases} 1/(\delta_i + s), & \text{if } j \in \mathcal{N}_i^{\text{in}} \setminus \{i\}, \\ s/(\delta_i + s), & \text{if } j = i, \end{cases} \quad (10)$$

where δ_i is the degree of vertex i in G and $s \in \{0, 1\}$. Note that we have two rules here depending on the value of s. Eq.(10) clearly satisfies (8) in both cases where $s = 0$ and $s = 1$, and hence the right-hand side of (5) represents the arithmetic mean of $\{\boldsymbol{z}_j[n]\}_{j\in\mathcal{N}_i^{\text{in}}}$ when $s = 1$ and that of

$\{\boldsymbol{z}_j[n]\}_{j\in\mathcal{N}_i^{\text{in}}\setminus\{i\}}$ when $s = 0$. However, Eq.(10) does not satisfy (9) in general. Therefore we cannot use Proposition 1 to show the convergence.

Scardapane and Di Lorenzo [5] did not use (10) but the Metropolis-Hasting rule:

$$c_{ij} = \begin{cases} 1/(\max\{\delta_i, \delta_j\} + 1), & \text{if } j \in \mathcal{N}_i^{\text{in}} \setminus \{i\}, \\ 1 - \sum_{j\in\mathcal{N}_i^{\text{in}}} 1/(\max\{\delta_i, \delta_j\} + 1), & \text{if } j = i, \end{cases} \tag{11}$$

in their experiments. It is easy to see that (11) satisfies $c_{ij} = c_{ji}$, (8) and (9). Therefore, by Proposition 1, all limit points of the sequence of $\{(\boldsymbol{w}_1[n]^{\mathrm{T}}, \boldsymbol{w}_2[n]^{\mathrm{T}}, \ldots, \boldsymbol{w}_I[n]^{\mathrm{T}})^{\mathrm{T}}\}_{n=0}^{\infty}$ generated by Algorithm 1 are stationary solutions of (1).

Sun *et al*. [3] claimed that (8) is not needed to prove the convergence and proposed the following rules:

$$c_{ij} = \begin{cases} 1/(\delta_j + s), & \text{if } j \in \mathcal{N}_i^{\text{in}} \setminus \{i\}, \\ s/(\delta_j + s), & \text{if } j = i, \end{cases} \tag{12}$$

where $s \in \{0, 1\}$[1]. It is easy to see that (12) satisfies (9) in both cases where $s = 0$ and $s = 1$. In particular, when $s = 0$, the values of c_{ij} for $j \in \mathcal{N}_i^{\text{in}}$ are inversely proportional to the degrees of vertices j.

3.2. Proposed Rules

Among various conditions on the consensus weights in Proposition 1, (8) is reasonable because it makes the right-hand side of (5) a convex combination of $\{\boldsymbol{z}_j[n]\}_{j\in\mathcal{N}_i^{\text{in}}}$. On the other hand, the role of (9) is not clear, though it is needed to the proof [5] of Proposition 1. In fact, it has been proven [4, 6] that some distributed optimization algorithms converge even if (9) is violated. Taking this fact into account, we propose the following rules:

$$c_{ij} = \begin{cases} \delta_j/(\sum_{j\in\mathcal{N}_i^{\text{in}}\setminus\{i\}} \delta_j + s), & \text{if } j \in \mathcal{N}_i^{\text{in}} \setminus \{i\}, \\ s/(\sum_{j\in\mathcal{N}_i^{\text{in}}\setminus\{i\}} \delta_j + s), & \text{if } j = i, \end{cases} \tag{13}$$

where $s \in \{0, 1\}$. Like (10), Eq.(13) satisfies (8) but neither $c_{ij} = c_{ji}$ nor (9) in general. The difference between (10) and (13) is that c_{ij} for $j \in \mathcal{N}_i^{\text{in}} \setminus \{i\}$ take the same value in the former while the values of c_{ij} for $j \in \mathcal{N}_i^{\text{in}} \setminus \{i\}$ are proportional to the degrees of vertices j in the latter. The idea behind (13) is that vertices with higher degrees may have more information than those with lower degrees. It is thus expected that NNs reach a consensus faster by assigning larger weights to NNs with high degrees.

4. Numerical Experiments

In order to clarify the relationship between the rule for choosing the values of the consensus weights and the convergence property of the distributed training, we perform experiments using some benchmark datasets.

[1] Strictly speaking, they proposed only the rule (12) with $s = 0$.

Table 1: Datasets used in experiments.

Dataset	Type	d	$\sum_{j=1}^{I} N_j$
Boston	Regression	13	506
Wisconsin	Classification	9	689

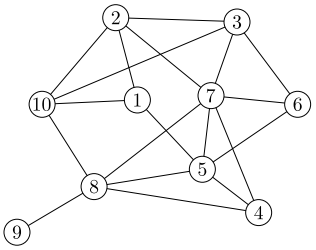


Figure 1: Communication among 10 NNs.

4.1. Setup

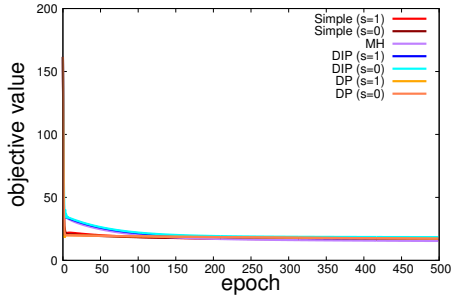
We use two datasets called Boston (or Boston House Prices) and Wisconsin (or Breast Cancer Wisconsin)[2]; the former is a regression problem and the latter a classification problem. Their characteristics are summarized in Table 1. In all experiments, the original data are normalized so that both inputs and outputs lie in the interval $[0, 1]$. Also, each dataset is divided into 10 nonoverlapping subsets and they are assigned to 10 NNs that can communicate with each other according to the undirected graph G shown in Fig. 1.

Each NN has three layers: the input layer with d neurons, the hidden layer with 10 neurons and the output layer with a single neuron. The output neurons of NNs for Boston dataset have the sigmoid activation function and all other neurons have the hyperbolic tangent activation function. We use the same loss functions and the regularization term as [5], that is, the squared loss function $\ell(a, b) = (a - b)^2$ for Boston dataset, the cross-entropy loss function $\ell(a, b) = -a\log(b) - (1-a)\log(1-b)$ for Wisconsin dataset, and the squared regularization term $r(\boldsymbol{w}) = \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$ for both datasets. The parameters of NNs are initialized by random numbers drawn from the standard normal distribution. The hyper parameters are set as $\tau = 10^{1.5}$, $\lambda = 10^{-1}$, $\alpha[0] = 0.001$ and $\epsilon = 0.01$ for Boston dataset, and $\tau = 10$, $\lambda = 10^{-0.5}$, $\alpha[0] = 0.0001$ and $\epsilon = 0.01$ for Wisconsin dataset. These values are determined based on preliminary experiments. All algorithms are implemented in Python 3.6.6 and executed on a PC with Intel Core i5-4590 and 8GB RAM.

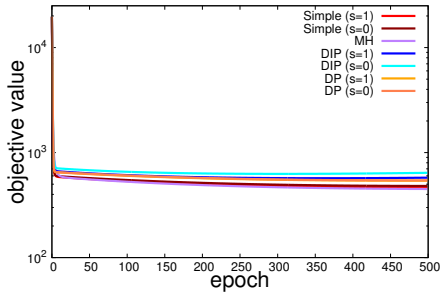
4.2. Results

We first compare the seven rules presented in the previous section in terms of the convergence speed of the objective value of (1). The time evolution of the objective value is shown in Fig. 2, where the rules (10), (11), (12) and (13)

[2] Both datasets are available at UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.html).

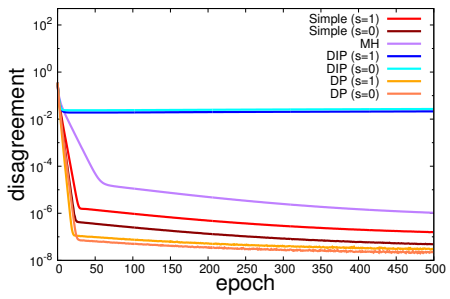


(a)

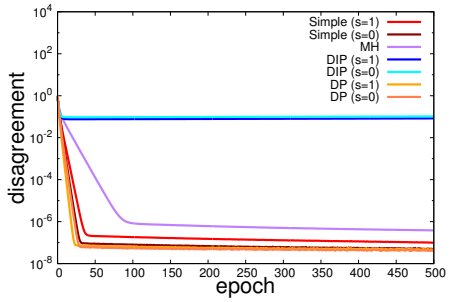


(b)

Figure 2: Time evolution of the objective value of (1) for (a) Boston dataset and (b) Wisconsin dataset.



(a)



(b)

Figure 3: Time evolution of the average disagreement for (a) Boston dataset and (b) Wisconsin dataset.

are denoted by Simple, MH (Metropolis-Hastings), DIP (Degree Inversely Proportional) and DP (Degree Proportional), respectively. It is easy to see that all rules decrease the objective value in a similar way for both datasets and there is no significant difference among seven rules.

We then compare the seven rules in terms of the convergence speed of the average disagreement: $D[n] = \frac{1}{I}\sum_{i=1}^{I}\left\|\boldsymbol{w}_i[n] - \frac{1}{I}\sum_{j=1}^{I}\boldsymbol{w}_j[n]\right\|_2$ where $I = 10$. The time evolution of the average disagreement is shown in Fig. 3. As one can see, DP is the fastest, Simple is the second, MH is the third, and DIP is much slower than these three rules. This indicates that the proposed rules allow NNs to quickly reach a consensus. It is also important to mention that DP with $s = 0$ is faster than that with $s = 1$, and the same holds true for Simple. This indicates that each NN should exclude its own solution when computing the weighted average of the solutions in its neighborhood.

5. Conclusions

We have proposed new rules for choosing values of consensus weights in the distributed training of neural networks, and shown experimentally that the proposed rules can achieve consensus faster than conventional rules. Evaluating the performance of the proposed rules on various training datasets, and theoretical analysis of the convergence property are future problems.

References

[1] V. D. Blondel, J. M. Hendrickx, A. Olshevsky and J. N. Tsitsiklis, "Convergence in multiagent coordination, consensus, and flocking," *Proceedings of the 44th IEEE Conference on Decision and Control*, pp.2996–3000, December 2005.

[2] P. Di Lorenzo and G. Scutari, "NEXT: In-network nonconvex optimization," *IEEE Transactions on Signal and Information Processing over Networks*, vol.2. no.2, pp.120–136, June 2016.

[3] Y. Sun, G. Scutari and D. Palomar, "Distributed nonconvex multiagent optimization over time-varying networks," *Proceedings of the 50th Annual Asilomar Conference on Signals, Systems, and Computers*, 2016.

[4] A. Nedić and J. Liu, "On convergence rate of weighted-averaging dynamics for consensus problems," *IEEE Transactions on Automatic Control*, vol.62, no.2, pp.766–781, February 2017.

[5] S. Scardapane and P. Di Lorenzo, "A framework for parallel and distributed training of neural networks," *Neural Networks*, vol.91, pp.42–54, July 2017.

[6] N. Takahashi and K. Kawashima, "A simple sufficient condition for convergence of projected consensus algorithm," *IEEE Control Systems Letters*, vol.2, no.3, pp.537–542, July 2018.