Consensus-based Distributed Machine Learning: Theory, Algorithms, and Applications

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Science & Engineering

2021

Bo Liu
Department of Electrical and Electronic Engineering
School of Engineering

Contents

List of Tables

List of Figures

Symbols

η	The learning rate
$\lambda_i(A)$	The ith eigenvalue of matrix A
$\mathbb{E}[\cdot]$	The expectation of an element
$\mathbb{R}^{m\times n}$	The set of real matrices of size $m \times n$
$\mathcal{A}$	The adjacency matrix
$\mathcal{D}$	A dataset
$\mathcal{E}$	The set of edges
$\mathcal{G}$	A graph
$\mathcal{L}$	The laplacian matrix
$\mathcal{N}(k)$	The neighbors of agent k
$\mathcal{V}$	The set of agents
$\max\{\cdot\}$	The maximum element
$\min\{\cdot\}$	The minimum element
$\otimes$	The Kronecker product
$\tilde{\mathcal{L}}$	The normalized laplacian matrix
$\|\cdot\|$	The L_2 norm of a vector
I_n	The $n \times n$ identity matrix

W	The weighted connectivity matrix
W^{T}	The transpose of matrix W

Abbreviations

ADMM Alternating Direction Method of Multipliers.

DQN Deep Q-networks.

IID Independent and Identically Distributed.

IoT Internet of things.

MAPE Mean Absolute Percentage Error.

MAS Multi-Agent System.

RMSE Root Mean Squared Error.

SGD Stochastic Gradient Descent.

SUMO Simulation of Urban Mobility.

SVM Support Vector Machines.

SVRG Stochastic Variance Reduced Gradient.

The University of Manchester

Bo Liu
Doctor of Philosophy
Consensus-based Distributed Machine Learning: Theory, Algorithms, and Applications
2021

With the great development of big data and increasing attention on privacy protection, distributed machine learning has received significant research interests in the last decade for its great ability in large-scale and privacy-related machine learning problems. Compared to traditional machine learning, distributed machine learning allows all participants to train a combined model, while keeping their private data locally stored. This thesis proposes a consensus-based distributed machine learning framework based on a decentralized communication topology, which frees the central master and exhibits great robustness and expansibility. The main contribution includes distributed supervised learning and distributed reinforcement learning.

First, a distributed training method based on the consensus algorithm is proposed for neural networks connected over a decentralized topology, which only requires a single consensus step after every training step. It is proved that the distributed training allows all the agents over a decentralized topology to converge to the optimal model based on the convergence analysis on empirical risk and model parameter. Second, the distributed training method is promoted based on the heuristic adaptive consensus algorithm and stochastic variance reduced gradient for agents connected in switching communication topologies. Theoretical analysis shows that all agents in switching graphs can still converge to the optimum and the stochastic variance reduced gradient reduces the variance introduced by stochastic gradient with only a little extra computational cost. Third, the error-compensated compression method with bit-clipping is applied in distributed training to compress the model parameter before sharing, which significantly saves communication costs with little decrease in model accuracy and is suitable for both IID and non-IID datasets. In addition, the distributed training method is combined with the blockchain technology to further benefit the privacy protection, and the proposed blockchain empowered distributed adaptive learning algorithm is applied in vehicular network, which ensures communication security and is immune to attack from a malicious participant.

Furthermore, the distributed training framework is extended to reinforcement learning, where deep Q-network is taken as an example. The learning process of deep Q-network is changed into a two-phase update process, where the Q-network of each agent is locally updated based on its own experience first, and the Q-networks of all agents are then globally updated using the consensus algorithm. This allows all agents to learn from other agents' experiences without the sharing of experience samples. Lastly, the distributed deep reinforcement learning framework is applied in the intelligent traffic light control problem, where a group of traffic light agents are connected in a decentralized communication topology. The superiority of the proposed distributed deep Q-networks method for traffic light control is verified by the simulation in SUMO with homogeneous and heterogeneous traffic flow patterns on different intersections.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright Statement

Publications

Journal Papers:

1). **B.Liu**, Z.Ding, C.Lv. 'Distributed Training for Multi-layer Neural Networks by Consensus.' **IEEE Transactions on Neural Networks and Learning Systems**, 31(5):1771-1778, 2019.

2). **B.Liu**, Z.Ding. 'Distributed Heuristic Adaptive Neural Networks with Variance Reduction in Switching Graphs.', **IEEE Transactions on Cybernetics**, 2019. (Accepted)

3). **B.Liu**, Z.Ding. 'A Consensus-based Decentralized Training Algorithm for Deep Neural Networks with Communication Compression.' **Neurocomputing**, 2021. (Accepted)

4). Z.Liang, J.Zhao, **B.Liu**, Y.Wang, Z.Ding. 'Velocity-based Path Following Control for Autonomous Vehicles to Avoid Exceeding Road Friction Limits Using Sliding Mode Method.' **IEEE Transactions on Intelligent Transportation Systems**, 2020. (Accepted)

5). Z.Li, **B.Liu**, Z.Ding. 'Consensus-Based Cooperative Algorithms for Training Over Distributed Data Sets Using Stochastic Gradients.' **IEEE Transactions on Neural Networks and Learning Systems**, 2021. (Accepted)

6). **B.Liu**, Z.Ding. A Distributed Deep Reinforcement Learning Method for Traffic Light Control.' **Neurocomputing**. (Submitted)

Conference Papers:

1). **B.Liu**, Z.Ding, C.Lv. ‘Platoon Control of Connected Autonomous Vehicles: A Distributed Reinforcement Learning Method by Consensus.’ **IFAC-PapersOnLine**, 2020. (Accepted)

Acknowledgements

Firstly, I would like to express my sincere gratitude to my PhD supervisor Prof. Zhengtao Ding for his guidance and suggestions on my research work in the past four years. I benefited greatly from his profound knowledge and rigorous academic attitude, and learned a lot from him on how to behave and deal with issues. This will benefit me all my life.

I wish to thank The University of Manchester for the generous scholarship for supporting my study and life. I also want to thank all colleagues in Control System Center and all friends during my doctoral study. Thanks to them for their gratuitous help and support in my study and for bringing happiness to my life.

I would like to thank my parents for their accompany, support, and encouragement during my schooling over the past 20 years.

Chapter 1

Introduction

1.1 Background

Recent years have witnessed the rapid development of artificial intelligence on both theory and application [1], especially for machine learning [2], which has been widely applied in many real-life scenarios. Machine learning mainly includes supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning has been well developed with theoretical analysis and widely used in varieties of applications [3], such as image recognition [4], speech recognition [5], and text processing [6]. The primary task of supervised learning is to train a 'black-box' model from limited data samples, where the training process is generally in a single machine. However, this centralized training manner may not be suitable for those large-scale or privacy-concerned problems, such as big data applications [7] and recommendation systems [8], which could only be or better addressed in a distributed manner [9, 10].

Firstly, the entire dataset may be too large to be processed by a single machine because of the hardware or software limitations, for example, processors in the vehicle intelligent terminal [11]. Secondly, data samples are generated or collected by different machines, which are intrinsically distributed, such as wireless sensor networks [12].

Thirdly, the data samples cannot be collected centrally in a single machine or shared among different machines because of privacy or sensitivity issues, especially for those data about personal behavior [13] and medical use [14, 15].

However, we still expect that the model is trained based on the data from all agents, even though without the sharing of data samples. The problem that we are now facing is that the model or decision is better to be made based on all the data samples instead of training with only local samples, while every agent cannot reveal its local data to a central server or other agents. Therefore, parallel computation and distributed optimization methods are proposed to address this problem.

Reinforcement learning is about an agent exploring an unknown environment to learn an optimal policy for a given task by iteratively trial and learning, which shows great potential in robot control [16], video games [17], and sequential decision-making problems [18]. The combination of supervised learning and reinforcement learning considerably promotes the development of artificial intelligence, which has been widely used in different engineering fields [19, 20]. Generally, the performance of a deep reinforcement learning algorithm for a specific task largely depends on the size of the experience data on interacting with the environment. Limited by the computing power of a terminal device (agent), the learning process with large-scale data is often conducted on a cloud server [21], with the agent collecting experience data and uploading it to the server. This centralized computing strategy offers a more powerful computational resource, but also bring some practical problems.

First, the large-scale data makes it powerless for a single server to train the model, for example, the AlphaGo Zero [22] requires the cooperation of hundreds of computation nodes to train the model. Second, the server-based training manner stretches the distance between the learner and the actor, which not only increases the communication costs but also decreases the response speed [23]. This weakness limits its application to practical problems with high real-time requirements, such as autonomous vehicles [24], especially when the communication systems are with high latency and low bandwidth. Third, the growing concern on privacy protection also hinders the server from gathering all data centrally, such as the data related to personal habits [25], medical use [26], and commercial activity [27].

On the one hand, the exponentially rising computing capability makes it possible for deep reinforcement learning to deal with more complex problems, such as chess [28], video games [29], and autonomous driving [30]. On the other hand, a more complicated problem commonly requires more computational resources to train the model, which is incapable or inefficient for a single computation node. Therefore, there is a trend to explore the distributed training framework to accelerate the learning process by the cooperation of many computation nodes.

Different from the single-agent system, network-connected Multi-Agent System (MAS) is more complex and challenging, especially for the coordination of this system for a desired purpose. Inspired by the behavior of biological swarms, distributed control strategies for MASs have been widely researched in both theory and applications, where the information of connected agents are used for global objectives. Motivated by the distributed control of MASs using neighboring information or sub-system information, we are expected to explore the distributed machine learning in network-connected agents to achieve global optimal model by the cooperation or coordination of a group of learning-based agents.

1.2 Literature Review

1.2.1 Distributed Supervised Learning

Different methods and algorithms for distributed optimization and distributed training are proposed, including distributed support vector machines (SVM) [31, 32] and distributed neural networks [33, 34].

One significant advance in distributed optimization is the alternating direction method of multipliers (ADMM) method [35], which decomposes a large-scale optimization problem into many small sub-problems, and then iteratively optimizes these sub-problems in alternating direction. ADMM is widely used in parallel optimization problems [36], such as distributed support vector machines [31, 37]. Another method for achieving distributed SVM is cascade SVM, related theoretical and practical issues can be found in the literature [38, 39], the main idea of which is to share the support vectors among connected agents. Castillo *et al.* [40] and Kim *et al.* [41] proposed the distributed training algorithm for support vector machines, where the support vectors from the local dataset are exchanged with connected neighbors, which exhibits a simple communication mechanism and fast convergence rate. However, as the support vectors are exactly the useful data samples, the sharing of which makes it unsuitable for privacy-concerned problems.

Federated learning [42] allows many agents to cooperatively train a shared model without exchanging local data. During the training process, each agent downloads the latest model from the central server and then trains this model with the local data to obtain a local update. All agents' local updates are subsequently collected and averaged to optimize the shared model by the centralized server. A recurrent neural network federated learning framework for keyboard next-word prediction is developed in [43], which shows that the federated algorithm outperforms the server-trained method on the language model, and benefits the privacy protection. Zhao *et al.* [44] analyzed the accuracy reduction of federated learning on non-IID (independent and identically distributed) data [45] and proposed a method to alleviate this reduction, where a small sub-dataset is globally shared to warm up the local model. Simulations show that this data-sharing strategy greatly improves the performance of federated learning on non-IID data.

To reduce the communication costs and improve model accuracy in federated learning, a multi-objective evolutionary algorithm is used in [46] to optimize the neural network structure, and a scalable approach is used for encoding the connectivity of network to improve the neural network evolution efficiency. The effectiveness of the proposed method is verified by the simulation experiments, where it improves the performance of federated learning, while reducing communication costs. After theoretical analyzing the convergence bound of federated learning, Wang *et al.* [47] proposed an adaptive method to balance the local update and global aggregation in resource-constrained systems for minimizing the loss function, which shows optimal performance under different models and data distributions in experimental simulation.

Another important issue in distributed training systems is communication security, as there may be dishonest participants in practical cooperative learning cases [48] and it is even possible to extrapolate the original data from the shared gradient or model information [49]. Blockchain technology is a good solution for the communication security problems in distributed learning, and there has been a lot of work on the combination of distributed learning and blockchain in applications.

Kong *et al.* [50] presented a blockchain-based distributed learning framework to realized the collective intelligence of connected autonomous vehicles, where all vehicles train their local models and then upload them to the blockchain for aggregation using the federated learning algorithm, with blockchain protecting these distributed learned models. Lu *et al.* [51] designed a data-sharing framework for the Internet of Vehicles with the combination of federated learning and blockchain to promote the driving experience while ensuring privacy security. In [52], a federated learning-based architecture is proposed to reduce the communication cost and protect the privacy of data providers with a deep reinforcement learning algorithm selecting the participants in the vehicular network, while a hybrid blockchain architecture is developed to ensure the reliability and enhance the security of the shared model parameter. Chai *et al.* [53] proposed a hierarchical blockchain-based federated learning framework for knowledge sharing in vehicular networks, where each vehicle learns a local model from the environmental data and shares the learned model with other vehicles. This framework is suitable for large-scale vehicular networks with a distributed pattern while meeting the privacy requirement of the Internet of Vehicles. To avoid unreliable updates and data poisoning attacks in federated learning, Kang *et al.* [54] proposed a worker selection method based on the metric of reputation to find out reliable workers, and took advantage of blockchain to manage the reputation of works, which is unchangeable and tamper-proof.

Different from federated training, the distributed training does not require a central node to collect and integrate all agents' information. The decentralized random vector functional link networks were proposed using a consensus-based method and

ADMM in [55], which shows that these two methods exhibit comparable performance on the accuracy, while the consensus-based method has a faster convergence rate than ADMM. Scardapane *et al.* developed the distributed learning algorithms echo state networks [56], where the entire dataset is distributed evenly on all agents over a decentralized graph, and the consensus-based algorithm is taken to compute the global average of the parameters over the graph. Although simulations show great performance in effectiveness and efficiency, these distributed models may not be capable of dealing with complex problems, because they have a quite simple structure and only train parameters of the output layer using the least squares method. Georgopoulos and Hasler [57] proposed a distributed algorithm for machine learning in networks, where the entire dataset is divided on arbitrarily connected agents without a central agent, and a consensus method is used to transform the centralized iterative learning algorithm to a distributed manner. However, a large number of communications are needed to reach consensus, which could be quite computationally expensive.

Yuan *et al.* [58] analyzed the convergence rate of decentralized parallel stochastic gradient descent for convex functions with bounded gradient, which shows a linear convergence rate. In reference [59], Lian *et al.* theoretically analyzed the centralized and decentralized algorithms on their convergence rate, which shows that the decentralized method has a faster convergence rate than its centralized counterpart on high latency or low bandwidth system, and can bring asymptotically linear speedup when more agents are available. Tang *et al.* [60] proposed a variance reduction method for decentralized parallel training on the distributed stored data with large variance. Chen *et al.* [61] discussed the synchronization of multiple neural networks with time delay and proposed the event-triggered control strategy to guarantee global exponential synchronization under some convenient conditions. The criterion of synchronization is feasible and does not require a Laplacian coupling matrix. Wang *et al.* [62] studied the coupled neural networks connected over both directed and undirected graphs, and designed appropriate adaptive laws and controllers to achieve synchronization in a limited time.

Sattler *et al.* [63] designed the sparse ternary compression to compress both the upstream and downstream communications, which considerably reduce communication costs. Tang *et al.* [64] provided extrapolation compression and difference compression for decentralized training, which is proved to be able to match the convergence rate of full precision and show good performance on networks with high latency and low bandwidth. Two different methods are proposed in [65] to decrease communication costs on the uploading process, one of which is to learn a restricted update with a part of model parameters, while the other method computes a full model update and then compresses it. However, combining the error-compensated strategy is not explored to further improve its performance by decreasing the influence of the compression operation.

Existing distributed training algorithms are mostly based on a central server to coordinate the training process, such as federated learning. The main drawback of federated learning is the strong dependence on the center, which may cause communication jam and is not robust to single point of failure. Thus, distributed training algorithms based on decentralized communication topology are expected to be intensively studied, including the training scheme, consensus strategy, communication compression, and security, etc.

1.2.2 Distributed Reinforcement Learning

Combining with deep learning, deep reinforcement learning has made a great success in recent years and has been applied in many different fields, such as chess [66], video games [67, 68], robot control [69, 70], autonomous vehicles [71, 72]. Generally, it is needed a huge number of experience samples to train a reinforcement learning model for a given task, and more complex tasks commonly required more experience samples [73]. Besides, the diversity of the experience samples also greatly influences the performance of the reinforcement learning model [74]. Thus, it is expected to collect as many experience samples as possible in as many scenarios as possible for the modeling process of reinforcement learning models. This poses a great challenge to traditional single-agent reinforcement learning algorithm, as the collection of experience samples by a single agent is inefficient and time-consuming. Therefore, distributed and parallel training systems are considered, which allows many agents to train a combined model, while keeping their experience samples locally stored. This allows each agent to learn from other agents' experiences and avoid the direct sharing of experience samples.

Based on federated learning, Zhuo *et al.* [75] proposed federated reinforcement learning to allow each agent to build its Q-network with other agents' help without actually data sharing, and the Gaussian differentials are applied on the share information to promote privacy protection. In [76], Lim *et al.* designed a similar federated reinforcement learning architecture to make multiple agents learn a control policy with the same type but different dynamics, where the learning experience of each agent is shared to transfer mature policy to other agents. This federated scheme promoted the reinforcement learning process for multiple devices, which is positively related to the number of agents. Anwar *et al.* [77] analyzed some attack methods from adversary participants in multi-task federated reinforcement learning, where a number of agents collaboratively improve the sum of their reward. An adaptive attack approach is proposed to achieve a better attack effect, and the federated reinforcement learning algorithm is modified to eliminate the effect of attack by adversary participants.

Nair *et al.* [78] designed a distributed architecture to train the agents using the reinforcement learning method, where each agent accumulates the experience data samples to fill its replay memory by interacting with a copy of the environment. Each agent computes the gradient of the loss based on its own replay memory and sent it to the server asynchronously, while the server gathers all the gradients to update the model, which is then pushed to all agents at a certain number of steps. Mnih *et al.* [79] proposed a parallel framework for reinforcement learning methods, where all agents explore in the same environment to accumulate experience data and optimize a shared model asynchronously, which reduces the correlation of the experiences samples. This architecture is suitable for both on-policy and off-policy algorithms and shows better performance than the previous work [78, 80]. By decoupling acting from learning, Horgan *et al.* [81] provided a distributed framework for deep reinforcement learning, which consists of multiple actors and a single learner. During the learning process, these actors explore in their copies of the environment and share the same model and experience replay memory, while the learner optimizes the model with prioritized experience replay [82]. Barth *et al.* [83] proposed the distributed framework for the deep deterministic policy gradient algorithm to deal with the continuous control task, which uses a number of distributed workers to write to the same replay memory. The proposed algorithm shows state-of-the-art performance on many difficult tasks, combining the use of prioritized experience replay and the N-step returns.

Sartoretti *et al.* [84] extend the asynchronous advantage actor-critic (A3C) to a distributed manner, which allows a number of agents to learn homogeneous goals without explicitly interacting. The proposed algorithm is based on a centralized policy and a decentralized execution to accelerates the learning process by the sum of all agents' experience. The effectiveness of the method is demonstrated by a multi-robot construction problem and additional training is not required for different test structures. Lee *et al.* [85] proposed a primal-dual distributed gradient temporal difference to optimize the distributed reinforcement learning problem, where each agent obtains its local reward and communicates with its neighbors to update their global value function regarding the sum of local rewards. After reformulating the problem into a convex problem with consensus constraint, it is proved that all agents converge to the same set of stationary points.

Liang *et al.* [86] developed an open library, RLlib, for reinforcement learning, which allows parallel of different reinforcement learning algorithms to get state-of-the-art performance, which distributes components of reinforcement learning using top-down hierarchical control algorithm to encapsulate parallelism and resource in short-running compute tasks. Chen *et al.* [87] designed a communication-efficient policy gradient algorithm in distributed reinforcement learning with a central controller and a number of learners, which is suitable for both multi-agent reinforcement learning and parallel reinforcement learning. By skipping gradient communication adaptively, the proposed method reduces the communication costs in distributed reinforcement learning without a decrease in performance.

Currently, most distributed reinforcement learning algorithms in the literature require a central server for aggregating all agents' information, which suffers from the heavy communication costs on the server. It is expected to explore the distributed reinforcement learning process without a central server to reduce the communication burden and alleviate the strong dependence on the server.

1.3 Contribution and Organization

In traditional machine learning, all the data are collected in a single agent (computing node) to train a model, which is then used for classification, regression, or prediction. This has made a great success in both theories and applications, while it has two main problems. One is, for a large-scale problem, the entire dataset is too large to be processed by a single agent or computing node. The other one is, the data cannot be collected locally due to privacy issues or huge energy consumption. For this problem, Google proposed federated learning in 2017, which allows a number of agents to train a combined model, while keeping their private data locally stored.

In federated learning, it requires a central master to coordinate the whole training process. More specifically, each agent trains its model based on its local data, and then sent the model or gradient to the master, while the master aggregates all information and then sends it back to each agent. This framework is effective and efficient, but it also has two main issues. One is the strong dependence on the central master, as every agent needs to communicate with the master at each iteration, this may cause a communication jam on the center, especially for the system with low bandwidth and high latency. More seriously, if the center fails for some reason, the whole training process will stop.

To avoid the above problem, this thesis aims to design a fully decentralized training framework, where it does not require a central master, and can still allow all the agents to train a combined model without data sharing. Besides, the distributed training based on a decentralized communication shows nearly the same performance as federated learning on different applications (for example, classification and regression), and the decentralized communication topology has great robustness and expansibility.

The research work mainly includes distributed supervised learning and distributed reinforcement learning. For distributed supervised learning, we propose the distributed training framework and promote it with the heuristic adaptive consensus algorithm, communication compression strategy, and blockchain technology. In terms of distributed reinforcement learning, we extend the distributed training framework to reinforcement learning and apply the distributed reinforcement learning method in traffic light control problem.

The main contribution of this work can be summarized as follows:

1. A distributed training framework based on a fully decentralized communication topology is proposed, which allows all the agents in the communication topology to train a combined model, while keep their private data locally stored. Analysis on empirical risk and model parameter shows that the proposed distributed training neural networks can converge to the optimal model, based on the assumption of the convexity of the empirical risk function.

2. A heuristic adaptive consensus algorithm is proposed for distributed training, which adaptively adjusts the weighted connectivity matrix based on the performance of each agent over the communication graph. Combining with the stochastic variance reduced gradient to reduce computational costs, the distributed adaptive neural networks with variance reduction is proposed, which also shows great robustness in switching communication topologies.

3. The convergence of the distributed neural networks is proved without the assumption on convexity, and an error-compensated model compression method is considered to reduce the communication costs during the distributed training process, which shows comparable performance with distributed training and centralized training, while saving a lot of communication costs.

4. Blockchain technology is combined with distributed learning to further promote privacy protection, and an adaptive consensus strategy is designed based on the contribution of each participant. The blockchain empowered distributed adaptive learning method is applied in vehicular network, which encourages all participants to make more contributions to the distributed learning process and prevent attack from a malicious participant.

5. The distributed training framework is extended to reinforcement learning, which allows homogeneous agents to learn from other agent's experiences to accelerate the training process, without the actual sharing of experience data samples.

6. The distributed reinforcement learning method is applied in the traffic light control problem, where the traffic light in each intersection is taken as a reinforcement learning agent. After learning with its own experience data samples by interacting with the environment, each agent communication with its neighbors to globally update their model. This allows all agents to learn from other agents' experience, which is effective and efficient for both homogeneous and heterogeneous traffic scenarios.

There are nine chapters in this thesis. Chapters 1 and 2 introduce the background and preliminary of the research work in this thesis. Chapters 3 to 6 propose the distributed supervised learning, including distributed training, distributed adaptive training, distributed training with communication compression, and blockchain empowered distributed adaptive training. Chapter 7 and Chapter 8 are about distributed reinforcement learning, including the design, verification, and application of the distributed reinforcement learning algorithm. Finally, Chapter 9 discusses the limitations and future work of the work. The detailed descriptions of each chapter are as follows.

Chapter 1 first introduces the background and motivation of distributed machine learning, and then conducts a literature review on distributed supervised learning and distributed reinforcement learning.

Chapter 2 presents some related preliminaries of this work, including graph theory, consensus algorithm, neural network, and deep Q-network.

Chapter 3 first discusses the advantages and disadvantages of the master-slave and decentralized communication topologies for parallel computation, which shows that the decentralized communication topology could avoid the possible communication jam on the central agent but incur extra communication costs. Then, a consensus algorithm is designed to allow all agents over a decentralized graph to converge to each other, and the distributed neural networks with enough consensus steps could have nearly the same performance as the centralized training model. Through the analysis of convergence, it is proved that all agents over an undirected graph could converge to the same optimal model even with only a single consensus step after every learning iteration, and this can significantly reduce the communication cost. Simulation studies demonstrate that the proposed distributed training algorithm for multi-layer neural networks without data exchange could exhibit comparable or even better performance than the centralized training model. The work in this chapter is published in [88].

Chapter 4 proposes a distributed adaptive training method for neural networks in switching communication graphs to deal with the problems concerned with massive data or privacy-related data. First, the stochastic variance reduced gradient is used for the training of neural networks to reduce the variance introduced by the stochastic gradient. Then, the authors propose a heuristic adaptive consensus algorithm for distributed training, which adaptively adjusts the weighted connectivity matrix based on the performance of each agent over the communication graph. Furthermore, it is proved that the proposed distributed heuristic adaptive neural networks ensure the convergence of all the agents to the optimum with a single communication among connected neighbors after every training step, which is also suitable for switching graphs. This theorem is verified by the simulation, which gives the results that fewer iterations are required for all agents to reach the optimum using the proposed heuristic adaptive consensus algorithm, and the stochastic variance reduced gradient can greatly decrease the fluctuations caused by the stochastic gradient and improve its performance with only a little extra computational cost. The work in this chapter is published in [89].

Chapter 5 proposes a consensus-based distributed training method with communication compression. First, the distributed training method is designed based on the decentralized topology to reduce the communication burden on the busiest agent and avoid any agent revealing its locally stored data. The convergence of the distributed training algorithm is then analyzed, which demonstrates that the distributed trained model can reach the minimal empirical risk on the whole dataset, without the sharing of data samples. Furthermore, model compression combined with the error-compensated method is considered to reduce communication costs during the distributed training process. At last, the simulation study shows that the proposed distributed training with error-compensated communication compression is applicable for both IID and non-IID datasets, and exhibits much better performance than the local training method. Besides, the proposed algorithm with an appropriate compression rate shows comparable performance with distributed training and centralized training, while saving a lot of communication costs. The work in this chapter is published in [90].

Chapter 6 proposes a blockchain empowered distributed learning framework in vehicular networks for multi-vehicle intelligence, which consists of local learning, blockchain-based communication, and contribution-based adaptive global consensus. Firstly, the consensus-based distributed learning algorithm is presented based on the distributed communication topology, which is more suitable for vehicular networks because of the intermittent and unreliable communication among vehicles. Then an adaptive consensus strategy is designed based on the contribution of each vehicle in the vehicular network, including model accuracy and computing power. To protect the communication security, blockchain technology is introduced to encrypt the shard model information and record the model sharing events as transactions in the block, which is traceable and tamper-proof. Meanwhile, the proof of contribution protocol is proposed to substitute the traditional proof of work protocol in blockchain, which greatly saves computing costs, and urges vehicles to make more contributions to the distributed learning process. Lastly, the simulation study shows that the proposed blockchain empowered distributed adaptive learning algorithm shows comparable performance with the federated learning and distributed learning in a real-world traffic signal classification task, and can prevent the attack from the malicious vehicle in the vehicular network.

Chapter 7 proposes a distributed training framework for deep reinforcement learning algorithms to address large-scale problems with privacy protection. First, we designed the decentralized communication topology with a server to alleviate the heavy burden on the central server. By pushing the model updating and data storage to the edge side, it not only unleashes the computing potential of the terminal devices but also shortens the response time. Second, the consensus algorithm is applied for all agents over the decentralized topology to approach each other, where each agent only requires neighboring information to achieve consensus. Then, the distributed training framework for deep Q-networks (DQN) is designed based on the consensus algorithm, which consists of local learning and global consensus. Through the convergence analysis, it is proved that all the agents converge to the optimum using the distributed reinforcement learning algorithm. At last, the simulation demonstrates that the proposed distributed learning deep Q-networks shows better performance than its single learning counterpart.

Chapter 8 proposes a distributed deep reinforcement learning algorithm for the traffic light control problem, which consists of local learning and global consensus. Firstly, the reinforcement learning environment for the traffic light control problem is built by defining the three key elements of state, action, and reward. Then, the CNN-based deep Q-network is designed to process the quantized traffic state information to obtain the state-action values. After locally optimizing the deep Q-networks of multiple traffic light agents based on their private experience samples, the consensus algorithm is subsequently applied to globally update these agents that are connected over a decentralized communication topology. In this way, the distributed learning agents learn from their neighbors' experience to optimize the modeling process without actually sharing experience data samples. Lastly, homogeneous and heterogeneous traffic flow patterns on different intersections are simulated in SUMO to verify the superiority of the proposed distributed deep Q-networks, with the comparison to the fixed-time strategy, local learning, and centralized learning algorithms. The simulation study demonstrates that the distributed learning algorithm without a central server shows comparable performance with centralized learning, which is much better than fixed-time strategy and the local learning method in both homogeneous and heterogeneous traffic scenarios.

Chapter 9 summarizes the main research results of this thesis and discusses some possible research directions for future work.

Chapter 2

Preliminaries

2.1 Graph Theory

A decentralized graph topology is assumed as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$, with $\mathcal{V} = \{1, 2, ...K\}$, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ and $\mathcal{A} = [a_{ij}] \in \mathbb{R}^{K \times K}$ representing the set of agents, the set of edges and the adjacency matrix, respectively. The connectivity of a graph with K agents can be expressed by an adjacency matrix A, which is given by:

$$a_{ij} = \begin{cases} 1 & \text{if agent } i \text{ is connected to } j \\ 0 & \text{otherwise} \end{cases}. \tag{2.1}$$

An edge $(i, j) \in \mathcal{E}$ represents that the ith and jth agents can communicate with each other and $a_{ij} = a_{ji} = 1$. The connectivity of the graph with K agents is known in advance and can be formalized in the form of a $K \times K$ weighted connectivity matrix W, where $w_{ij} > 0$ if $(i, j) \in \mathcal{E}$ or $i = j$, otherwise $w_{ij} = 0$ [91]. More generally, the value of the element w_{ij} represents the strength of the connection between these two corresponding agents, and $w_{ij} = 0$ means agent i and agent j are disconnected.

2.2 Consensus Algorithm

The undirected graph is concerned in this thesis, and the weighted connectivity matrix $W = [w_{ij}] \in \mathbb{R}^{K\times K}$ of the graph should meet the following requirements to make all the agents achieve consensus,
(i) $w_{ij} \in [0, 1)$, $\forall(i, j)$,
(ii) $w_{ij} = w_{ji}$, $\forall(i, j)$,
(iii) $\sum_{j=1}^{K} w_{ij} = 1$, $\forall i$.

We suppose that each agent k in the graph has a parameter row vector denoted by $\theta_k \in \mathbb{R}^{1\times N}$, and the consensus algorithm can allow all agents to converge to their average $\overline{\theta} = \frac{1}{K}\sum_{k=1}^{K}\theta_k$ only with local communication by iteratively computing the mean value of directly connected neighbors. The update of an agent i using the consensus algorithm is given by:

$$\theta'_i = \sum_{j=1}^{K} w_{ij}\theta_j, \tag{2.2}$$

where θ'_i is the updated parameter of θ_i after a single consensus step.

The update of parameters of all the agents with a single consensus step can then be written as:

$$\boldsymbol{\theta}' = \mathcal{C}(\boldsymbol{\theta}, W) = W\boldsymbol{\theta}, \tag{2.3}$$

where the matrices $\boldsymbol{\theta} = [\theta_1, \theta_2 ... \theta_K]^T \in \mathbb{R}^{K\times N}$ and $\boldsymbol{\theta}'$ are defined as the concatenation of parameter vectors of all the agents before and after the consensus process $\mathcal{C}$, respectively, and the kth row of $\boldsymbol{\theta}$ is θ_k. Regardless of the initial status of each agent, this method would allow all the agents to converge to their global average by repetitively computing (2.3).

Different consensus strategies generate different weighted connectivity matrix W for a certain decentralized graph, which may influence the convergence rate of the consensus algorithm. Common consensus strategies include Max-Degree [92], Metropolis-Hastings [93] and Laplacian method [94].

2.2.1 Max-Degree

Max-degree weights matrix is a simple way to obtain the weighted connectivity matrix W to ensure all the agents convergence to the global average, which is also a common choice in real-world applications, and the matrix W is given by:

$$w_{ij} = \begin{cases} \frac{1}{d_{max}+1} & \text{if } \ i \ \text{ is connected to } \ j \\ 1 - \frac{d_i}{d_{max}+1} & \text{if } \ i = j \\ 0 & \text{otherwise} \end{cases}, \tag{2.4}$$

where d_i is the degree of agent i, which means the number of agents connected with agent i, and d_{max} is the maximum degree of the graph.

2.2.2 Metropolis-Hastings

Metropolis-Hastings is another way to obtain the weighted connectivity matrix W, which can also ensure the convergence. This method does not need the knowledge of global information of the graph topology, while each agent needs to know the degrees of all its neighbours. The weighted connectivity matrix W of Metropolis-Hastings is given by:

$$w_{ij} = \begin{cases} \frac{1}{max(d_i,d_j)+1} & \text{if } \ i \ \text{ is connected to } \ j \\ 1 - \sum_{j\in\mathcal{N}(i)} \frac{1}{\max(d_i,d_j)+1} & \text{if } \ i = j \\ 0 & \text{otherwise} \end{cases}, \tag{2.5}$$

where $\mathcal{N}(i)$ represents the set of agents which are directly connected to agent i, with d_i denoting the degree of agent i.

2.2.3 Laplacian Heuristic

Another efficient way to obtain the weighted connectivity matrix W is the Laplacian method [95]. The Laplacian matrix $\mathcal{L} = [l_{ij}] \in \mathbb{R}^{K\times K}$ of an undirected graph is described by

$$l_{ij} = \begin{cases} \sum_{j=1}^{K} a_{ij} & \text{if } \ i = j \in \mathcal{V} \\ -a_{ij} & \text{if } \ i \neq j \end{cases}. \tag{2.6}$$

The weighted connectivity matrix W is then obtained by

$$W = I_n - \alpha \mathcal{L}, \tag{2.7}$$

where $\alpha \in (0, 1/\max(l_{ij}))$ is a user-defined factor to adjust the influence of its neighbors on the agent.

A simple and efficient way to set α is given by:

$$\alpha = \frac{1}{d_{max} + 1}, \tag{2.8}$$

where d_{max} is the maximum degree of the graph.

2.3 Neural Network

The structure of an multi-layers neural network includes input layer, hidden layer and output layer. Input layer and output layer are both single layers, whose number of nodes depends on the numbers of inputs and outputs for a particular problem, while hidden layer often contains one or several layers, which represents the complexity of a neural network. Generally, more hidden layers means more complex network structure, which is also more likely to lead to over-fitting and decrease the model's generalization ability. Researchers found that single hidden layer neural network could have good enough performance with a sufficient number of nodes in hidden layer [96].

Back-propagation is a popular optimization method for training neural network to minimize a cost function, and the training process of a neural network can be described as two iterative steps. The empirical risk is computed at the first step, which can be the mean squared error between the actual and estimated output, and an update of the parameter θ is subsequently conducted based on the empirical risk. In the centralized learning case, the empirical risk is defined by:

$$E(\mathcal{D}, \theta) = \frac{1}{n} \sum_{i=1}^{n} l(x_i, y_i, \theta), \tag{2.9}$$

$$l(x_i, y_i, \theta) = \frac{1}{2} (\hat{y_i} - y_i)^2, \tag{2.10}$$

$$\hat{y_i} = f(x_i, \theta), \tag{2.11}$$

where E denotes the empirical risk over the entire training dataset $\mathcal{D}$ (n samples), $l(x_i, y_i, \theta)$ is the loss function for a single data sample (x_i, y_i) over parameter θ, $\hat{y}_i$ is the estimated value of y_i, and f represents a mapping from x_i to y_i with parameter θ.

The gradient of the empirical risk with respect to θ is given by:

$$\nabla E(\theta) = \frac{\partial}{\partial \theta} E(\mathcal{D}, \theta) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta} l(x_i, y_i, \theta), \tag{2.12}$$

where $\nabla E(\theta)$ is the partial derivative of E over θ.

Then, the update of model parameter θ using back-propagation is described as:

$$\theta' = \theta - \eta \nabla E(\theta), \tag{2.13}$$

where θ' is the updated model parameter of θ, with η being the learning rate.

2.4 Deep Q-network

We define an experience data sample of the agent at step t by a tuple $d_t = (s_t, a_t, r_{t+1}, s_{t+1})$ during the reinforcement learning process, where a single agent is interacting with an environment to learn the optimal policy in discrete time steps. In detail, a_t denotes the chosen action based on its policy π in the current state s_t of the environment, with r_{t+1} and s_{t+1} representing the reward from the environment and the next state, respectively.

The value of the state-action pair (s_t, a_t) is defined by the accumulated reward R_t, that is

$$R_t = \sum_{p=1}^{P} \gamma^k r_{t+p}, \tag{2.14}$$

where $\gamma \in (0, 1]$ is the discount factor, P denotes the end of an episode or ∞ for a task without a final state.

The main task of the reinforcement learning process is to estimate the values for all pairs (s, a) in the environment under the policy π, which is defined as

$$Q_\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a], \tag{2.15}$$

where $Q_\pi(s, a)$ is the value of action a in state s, $\mathbb{E}[\cdot]$ represents the expectation.

After that, we can obtain the optimal value $Q_*(s, a) = \max_\pi Q_\pi(s, a)$ of the action-state pair (s, a), and the optimal policy is to choose the action with highest value $Q_*(s, a)$ in all states of the environment. Q-learning records all the value $Q(s, a)$ for all action-state pairs in a table, and the table is updated based on the reward from the environment until the convergence.

However, most practical problems are with too many or even infinite state-action pairs, which makes it hard to calculate the values of all pairs of (s, a). Therefore, approximation methods are proposed to estimate the $Q_*(s, a)$, among which, the neural network is a good choice to represent a parameterized state-action value function $Q(s, a; \theta)$. Then the reinforcement learning problem is changed to the optimization of the neural network parameter θ to estimate the $Q_*(s, a)$ accurately, where the loss function l about θ is defined as

$$l(\theta_t, d_t) = [q_t - Q(s_t, a_t; \theta_t)]^2, \tag{2.16}$$

$$q_t = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_t), \tag{2.17}$$

where q_t is taken as the actual value of $Q(s_t, a_t; \theta_t)$ over the current parameter on the data sample d_t.

The key idea of deep Q-network [80] is the use of experience replay and the target parameter $\hat{\theta}_t$. Instead of updating the model parameter θ on a single data sample d_t at every step, the DQN randomly chooses a batch of samples from its replay memory $e = \{d_1, d_2, ...d_N\}$, which stores the agent's historical experience samples. Besides, the target value q_t is computed over an outdated model parameter, i.e., the target model parameter $\hat{\theta}_t$, rather than the current model parameter θ_t, which is modified as

$$q_t = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \hat{\theta}_t). \tag{2.18}$$

The experience replay memory offers more training samples in each step and decorrelates the samples sequence, while the use of target Q-network disrupts the correlation of the action value $Q(s_t, a_t; \theta_t)$ and its target q_t. Combining with replay memory and the target Q-network, the training process of DQN is more efficient and less likely to fall into overfitting, and correspondingly, shows better performance.

Chapter 3

Distributed Multi-layer Neural Networks by Consensus

3.1 Introduction

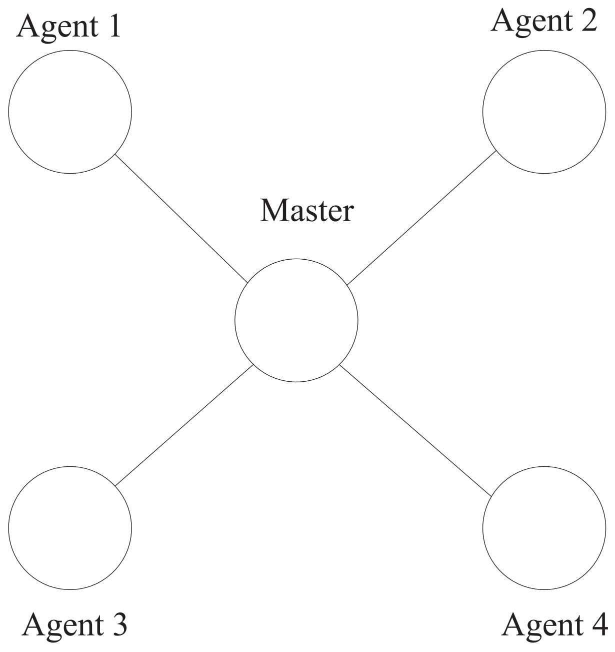
The affordable price and exponentially rising computing power of intelligent terminal devices, such as private computers and intelligent phones, makes it possible for the application of artificial intelligence in our daily life. This combination of information technology and artificial intelligence greatly enriches our life but also brings some practical problems.

First of all, a huge amount of data are generated and collected by different devices in our daily life, which makes it difficult and inefficient for a central server to gather and process all the data [97]. Besides, these devices are becoming increasingly multi-functional and intelligent, which are usually equipped with a variety of sensors to collect data, and even powerful hardware and software to process data. The advent and development of 5G and Internet of things(IoT) [98] allow intelligent agents to communicate with each other, not just the central server. This combination of intelligent devices with computational ability has promoted the progress of fog computing [99] and edge computing [100], which considerably reduces the computation and communication burden on the cloud server, and provides more efficient service. Furthermore, there is a growing concern on data privacy and security issues [101], which makes this central computing method unsuitable for processing private and sensitive data, such as medical data [102] and personal habit data [103]. Based on the above consideration, an increasing number of users prefer to store and process their data on the local agent rather than reveal it to a central server.

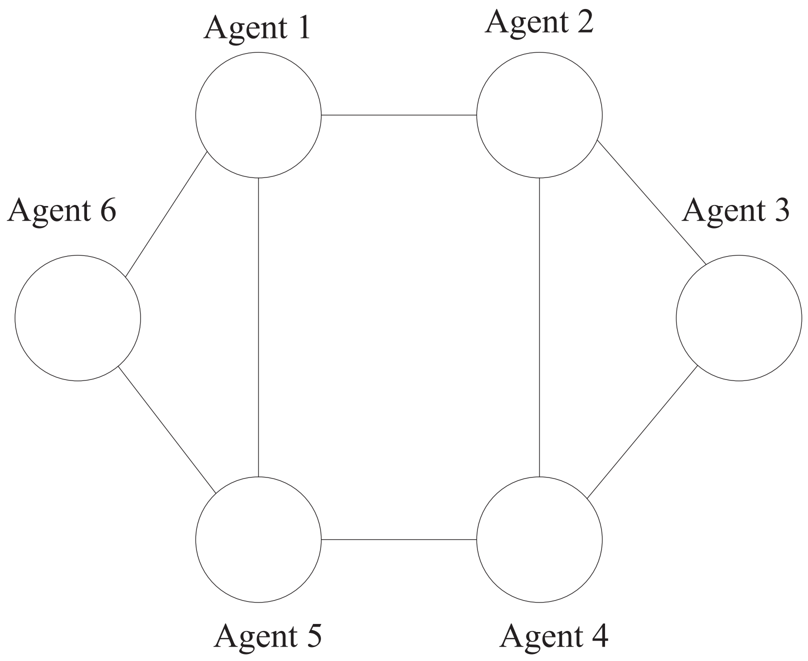
It is important that the local agent can still use the information of other agents for better modeling even though without actually sharing the original data. Therefore, we are facing the problem that how to deal with the massive data distributed stored on many connected intelligent devices with the limitation that each agent can not reveal its local data to a central node or any other agents.

Parallel computation is a leading method for distributed training to solve large-scale and privacy-concerned machine learning problems, such as deep learning [104], data mining with big data [105] and inference on wireless sensor network [106]. Exiting parallel algorithms are mostly designed for the centralized graph topology, such as the parameter server topology [107], where there is a master (or central) agent connected with multiple slave agents as shown in Figure 3.1(a). The central agent receives information (weights or gradients for neural networks) from all other slave agents and computes the sum or average of them, which is then fed back to the slave agents as the update of the model parameter. There are many popular distributed training algorithms designed for the parameter server topology, such as federated learning [108], and asynchronous parallel gradient descent [109]. The potential bottleneck of this master-slave graph topology is the possible communication traffic jam on the central agent for the reason that all other agents need to communicate with the central agent concurrently after each training iteration [110]. Too much pressure on the server may lead to communication blocking or even the failure of the server, especially for those communication systems with low bandwidth and high latency. In some extreme cases, the whole distributed training architecture would be out of operation, if the central node (server) fails. To avoid the communication jam on the central server, a decentralized topology as shown in Figure 3.1(b) is proposed, where there is no central agent, and all agents only need to communicate with its directly connected neighbors.

We consider the scenario that data samples are distributed stored on multiple agents over a graph, where the communication is only allowed among connected agents. However, data samples are unavailable to be exchanged among agents or collected centrally for some reason. First, the hardware or software limitation makes it infeasible for a



(a)



(b)

Figure 3.1: (a) Centralized graph topology (b) Decentralized graph topology.

single agent to process the massive data, for example, processors in mobile terminals [111]. Second, data samples are intrinsically stored on different devices or multi-sensors [112], where the data is generated or collected. Third, it is unavailable to share the data samples among agents or upload them to a centralized server out of privacy-related concerns, especially for those data about personal behavior [113] and commercial use [114, 115].

This chapter proposes a consensus-based distributed training method for multi-layer neural networks, which requires only a single communication among connected neighbors over a decentralized graph topology after each training iteration. The convergence analysis shows that the proposed distributed training algorithm for multi-layer neural networks can converge to the optimal model in union, which is verified by the simulation studies.

The remainder of this chapter is organized as follows. Section 3.2 compares the centralized graph topology and the decentralized graph topology for parallel computation and introduces the consensus algorithm for distributed training. Section 3.3 analyzes the convergence of distributed training for multi-layer neural networks over a decentralized graph using the consensus algorithm. Section 3.4 details the simulation and numerical results on four UCI datasets for binary classification, multi-labeled classification, and regression, which verify the effectiveness of the proposed distributed training algorithm. Section 3.5 concludes this chapter.

3.2 Distributed Neural Networks

In the distributed training case, supposing that the entire dataset is divided into K sub-datasets and distributed on K agents (or machines) with the same initialized neural networks, the parameters $(\theta_1, \theta_1 ... \theta_K)$ of all agents are averaged and then fed back to each agent as the updated parameter after each training iteration. In the master-slave graph, the central agent computes the mean value, while the consensus algorithm with enough consensus steps is used in the decentralized graph.

The distributed neural networks would give nearly the same result as the centralized training neural network based on the entire dataset when batch gradient descent is taken as the optimization method for minimizing the empirical risk. For distributed training, the entire empirical risk can be decomposed into local empirical risks,

$$E(\mathcal{D}_k, \theta_k) = \frac{1}{n_k} \sum_{i=1}^{n_k} l({}^k x_i, {}^k y_i, \theta_k), \tag{3.1}$$

$$\nabla E(\theta_k) = \frac{1}{n_k} \sum_{i=1}^{n_k} \frac{\partial}{\partial(\theta_k)} l({}^k x_i, {}^k y_i, \theta_k), \tag{3.2}$$

where $\mathcal{D}_k$, n_k, θ_k, and $({}^k x_i, {}^k y_i)$ represent the dataset, the number of data samples, model parameter and data sample i on agent k, respectively.

Substituting (3.1) and (3.2) into (2.9) and (2.12), respectively, the global empirical risk and parameter can be obtained by averaging these parameters over all agents, which leads to:

$$\begin{aligned} E(\mathcal{D}, \theta) &= \sum_{k=1}^{K} \frac{n_k}{n} \frac{1}{n_k} l({}^k x_i, {}^k y_i, \theta_k) \\ &= \sum_{k=1}^{K} \frac{n_k}{n} E(\mathcal{D}_k, \theta_k), \end{aligned} \tag{3.3}$$

$$\begin{aligned} \nabla E(\theta) &= \sum_{k=1}^{K} \frac{n_k}{n} \frac{1}{n_k} \frac{\partial}{\partial(\theta_k)} l({}^k x_i, {}^k y_i, \theta_k) \\ &= \sum_{k=1}^{K} \frac{n_k}{n} \nabla E(\theta_k), \end{aligned} \tag{3.4}$$

Equation (3.4) shows that the distributed neural networks can obtain the same gradient vector $\nabla E(\theta)$ as the centralized case by computing the average of gradient vectors $\nabla E(\theta_k)$ of all agents over the graph, and this can be easily extended to stochastic gradient descent and mini-batch stochastic gradient descent [59].

Therefore, the distributed neural networks in a decentralized topology can be realized by substituting the local gradient with the average of the gradients using the consensus algorithm with enough consensus steps. The update procedures are as follows:

$$\nabla E(\hat{\boldsymbol{\theta}}) = \mathcal{C}(\nabla E(\boldsymbol{\theta}), W), \tag{3.5}$$

$$\theta''_k = \theta_k - \eta \nabla E(\hat{\theta}_k), \tag{3.6}$$

where $\nabla E(\boldsymbol{\theta}) = [\nabla E(\theta_1), E(\theta_2), ...E(\theta_K)]^T$ and $\nabla E(\hat{\boldsymbol{\theta}})$ represent the concatenation of gradient vectors of all agents over the graph before and after the consensus process $\mathcal{C}$, and the kth row of $\nabla E(\boldsymbol{\theta})$ is $\nabla E(\theta_k)$. θ''_k is the updated parameter of θ_k after the training process, $\nabla E(\hat{\theta}_k)$ and η denote the gradient of agent k and the learning rate, respectively.

It is noteworthy that the order of (3.5) and (3.6) can be exchanged, which will not affect the theoretical analysis and can remove the requirement that all agents should have the same initialized parameters. That is, the local parameters θ_k are firstly updated with the local gradient vector $\nabla E(\hat{\theta}_k)$, and then the average of the parameters of all the agents are obtained using the consensus algorithm, the processes of which are as follows:

$$\theta'_k = \theta_k - \eta \nabla E(\theta_k), \tag{3.7}$$

$$\boldsymbol{\theta}'' = \mathcal{C}(\boldsymbol{\theta}', W), \tag{3.8}$$

where θ_k and θ'_k represent the parameter of agent k before and after the training process, respectively. The matrices $\boldsymbol{\theta}' = [\theta'_1, \theta'_2, ...\theta'_K]^T$ and $\boldsymbol{\theta}''$ are defined as the concatenation of parameter vectors of all agents before and after the consensus process, respectively. It is notable that repetitively computing similar to (2.3) is required for (3.5) and (3.8) to reach their mean value.

Based on the above analysis, we find that the distributed neural networks over a fixed and undirected graph using the consensus algorithm with enough consensus steps could exhibit nearly the same result as the centralized training model based on the entire dataset. However, the main drawback of this method is that repetitively computing is required to reach the approximate average after each training iteration, and this computation cost could be multiple times than training with the master-slave topology.

To tackle this problem, we propose the distributed training algorithm for multi-layer neural networks with only a single consensus step after each training iteration, which would significantly decrease the computation cost for consensus. And the proof, as well as simulation, will be given in the following section to verify that the distributed neural networks over an undirected graph allow all agents to converge to the globally optimal model.

The specific process of the distributed training algorithm for multi-layer neural networks is summarised in Table 3.1.

Table 3.1: The process of distributed training of neural networks

```
Algorithm 1: Distributed training for the multi-layer neural networks
Inputs: The structure of the neural networks, the number of agents $K$, the sub-datasets $(\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_K)$ and the corresponding weighted connectivity matrix $W$ of the graph.
Outputs: The optimal model parameter $\theta^*$.
1: Randomly select the model parameter $\theta_k$ for each agent over the graph.
2: Each agent $k$ solves the local training problem using back-propagation with corresponding sub-dataset $\mathcal{D}_k$ to obtain the locally updated $\theta'_k$.
3: Globally update the parameter $\theta'_k$ over the graph using the consensus algorithm with a single consensus step to obtain $\theta''_k$.
4: Back to step 2 with globally updated $\theta''_k$.
5: Check the termination criterion (such as a given number of iterations).
6: Return the optimal model parameter $\theta^*$.
```

3.3 Convergence Analysis

We will prove that the proposed distributed training methods with only a single consensus step after each training iteration would allow all the neural networks over an undirected graph to converge to the same optimal neural network model. To illustrate this point, we analyze the distributed neural networks from the perspectives of the decrease of empirical risks and the convergence of model parameters using two different methods as presented in 3.3.1 and 3.3.2, respectively.

The over-parameterized neural networks with enough hidden layer nodes are considered in this chapter, which has a linear convergence rate to the optimal solution based on the gradient descent method [116–118].

Assumption 1: The model parameter gradually approaches to the optimal solution with the increase of training steps, that is,

$$\|\theta_i(t+1) - \theta^*\| \leq \|\theta_i(t) - \theta^*\|, \; i = (1, 2, ..., K), \tag{3.9}$$

where θ^* and $\theta_i(t)$ denote the optimal model parameter and the model parameter of agent i at iteration t, respectively.

3.3.1 Analysis on Empirical Risk

Assumption 2: Empirical risk $E(\theta)$ is a quasiconvex function of θ, which satisfies $E(\theta_i) \leq E(\theta_j)$ if $\|\theta_i - \theta^*\| \leq \|\theta_j - \theta^*\|$.

The parameter matrix $\boldsymbol{\theta'} = [\theta'_1, \theta'_2, ...\theta'_K]^T$ is defined as the updated parameter matrix $\boldsymbol{\theta} = [\theta_1, \theta_2, ...\theta_K]^T$ after the training process $\mathcal{T}$, which satisfies $\|\theta'_i - \theta^*\| \leq \|\theta_i - \theta^*\|$ based on Assumption 1, thus $E(\theta'_i) = E(\mathcal{T}(\theta_i)) \leq E(\theta_i)$ based on Assumption 2. $E(\theta_i)$ and $E(\theta'_i)$ could be aliased as E_i and E'_i, respectively.

The consensus process $\mathcal{C}$ with a single step for the parameter matrix $\boldsymbol{\theta'} = [\theta'_1, \theta'_2, ...\theta'_K]^T$ of the empirical risk vector $\mathcal{F}' = (E'_1, E'_2, ...E'_K)$ over the weighted connectivity matrix W can be given as:

$$E(\theta''_i) = E(\sum_{j=1}^{K} w_{ij}\theta'_j),\ i = (1, 2, ..., K), \tag{3.10}$$

where $E(\theta''_i)$ denotes the updated $E(\theta'_i)$ after a single consensus step, which is aliased as E''_i.

Proposition 1: Given that all the entries of the weighted connectivity matrix $w_{ij} \in [0, 1)$ and $\sum_{j=1}^{K} w_{ij} = 1$, it can be obtained that,

$$\|(E''_1, E''_2, ...E''_K)\|_\infty \leq \|(E'_1, E'_2, ...E'_K)\|_\infty, \tag{3.11}$$

where $\|\cdot\|_\infty$ represents the max-norm of a vector, and the '=' holds only when all the E'_k are equal.

Proof:

$$\begin{aligned} \|(E''_1, E''_2, ...E''_K)\|_\infty &= \max_k \|E''_k\| \\ &= \max_k \|E(\sum_{j=1}^{K} w_{kj}\theta'_j)\| \\ &\leq \max_k \|E(\theta'_k)\| \\ &= \|(E'_1, E'_2, ...E'_K)\|_\infty. \end{aligned} \tag{3.12}$$

The update of $\boldsymbol{\theta}$ with the combination of local training process $\mathcal{T}$ and global consensus process $\mathcal{C}$ with a single step can then be given by:

$$\begin{aligned} \boldsymbol{\theta}'' &= \mathcal{C}([\theta'_1, \theta'_2, ...\theta'_K]^T, W) \\ &= \mathcal{C}([\mathcal{T}(\theta_1), \mathcal{T}(\theta_2), ...\mathcal{T}(\theta_K)]^T, W). \end{aligned} \tag{3.13}$$

Theorem 1: Each empirical risk E_k will converge to the minimal empirical risk E^* in spite of their initial statuses, under Assumptions 1 and 2.

Proof: $R(F)$ is defined as the maximum gap between empirical risks $\mathcal{F} = (E_1, E_2, ...E_K)$ and the optimal E^*, which can be given by:

$$\begin{aligned} R(F) &= \max_k(\|E_k\| - \|E^*\|) \\ &= \|(E_1, E_2, ...E_K)\|_\infty - \|E^*\|, \end{aligned} \tag{3.14}$$

where it always holds that $\|E^*\| \leq \|E_k\|$.

By Proposition 1 and (3.14),

$$\begin{aligned} R(\mathcal{F}'') &= \max_k(\|E''_k\| - \|E^*\|) \\ &= \|(E''_1, E''_2, ...E''_K)\|_\infty - \|E^*\| \\ &\leq \|(E'_1, E'_2, ...E'_K)\|_\infty - \|E^*\|. \\ &= R(\mathcal{F}') \end{aligned} \tag{3.15}$$

Under Assumption 1,

$$\begin{aligned} R(\mathcal{F}') &= \max_k(\|E'_k\| - \|E^*\|) \\ &\leq \|E_k\| - \|E^*\| \\ &\leq \|(E_1, E_2, ...E_K)\|_\infty - \|E^*\|. \\ &= R(\mathcal{F}) \end{aligned} \tag{3.16}$$

Therefore, we can get,

$$R(\mathcal{F}'') \leq R(\mathcal{F}') \leq R(\mathcal{F}). \tag{3.17}$$

Equation (3.17) shows that the maximum gap $R(\mathcal{F})$ has a downward trend, with the increase of training process combined with consensus process, which means that the empirical risks $(E_1, E_2, ...E_K)$ of all the agents over the graph would gradually converge to the optimal E^*. This completes the proof. ■

Remark 1: Assumption 1 describes a general idea of the training process for an over-parameterized neural network; that is, a better model should be obtained after a single training step. In the case that the empirical risk decreases after several training steps, we can take the several consensus steps during these training steps as a whole consensus process, which will not affect Proposition 1 and the proof still holds.

Remark 2: The convexity of empirical risk function is often used to prove the convergence of neural networks [119–121]. Assumption 2 ensures that the upper bound of the empirical risks of all the agents would decrease after a consensus step, which may also apply to non-convex optimization problems, as long as all agents are restricted in the same basin of attraction. We will verify it in the simulation experiments.

3.3.2 Analysis on Model Parameter

This section analyzes the proposed distributed training algorithm using the Lyapunov method from the perspective of the convergence of model parameters of all agents.

We define $\Delta\theta_i(t) = \theta_i(t) - \theta^*,\ i = (1, 2, ..., K)$ as the gap between $\theta_i(t)$ and θ^*, which would exhibit a downward trend under Assumption 1, that is,

$$\|\Delta\theta_i(t+1)\| \leq \|\Delta\theta_i(t)\|,\ i = (1, 2, ..., K). \tag{3.18}$$

Assumption 3: The parameter gap matrix of all agents satisfies the following equation, with A being neutrally stable [122].

$$\Delta\boldsymbol{\theta}(t+1) = A\Delta\boldsymbol{\theta}(t), \quad \lambda_{max}(A) \leq 1. \tag{3.19}$$

where the matrix $\Delta\boldsymbol{\theta}(t) = [\Delta\theta_1(t), \Delta\theta_2(t), ...\Delta\theta_K(t)]^T$ is defined as the concatenation of parameter gap vectors of all agents and the ith row of the matrix $\Delta\boldsymbol{\theta}(t)$ represents the $\Delta\theta_i(t)$, $\lambda_{max}(A)$ is the maximum eigenvalue of A.

Remark 3: Neutrally stability only requires that the gap between the model parameter and the optimal parameter keep decreasing as the training moving on, which is an extension of Assumption 1.

Theorem 2: The parameters $\theta_i(i = 1, 2...K)$ of all neural networks over a fixed and undirected graph would converge to an identical optimal parameter close to θ^* using the proposed distributed training method, under Assumption 3.

Proof: With the combination of the local training process and the global consensus process, the update of agent i can be decomposed into two procedures:

$$\theta_i(t + 1/2) = \theta^* + \Delta\theta_i(t + 1), \tag{3.20}$$

$$\theta_i(t + 1) = B\theta_i(t + 1/2) + Cu_i(t + 1), \tag{3.21}$$

where (3.20) and (3.21) describe the local training and the global consensus process, with $\theta_i(t + 1/2)$ and $\theta_i(t + 1)$ being the updated parameter of agent i by this two processes, respectively. For the neural network training problems, both B and C are the identity matrix I_n, with n being the number of parameters of θ_i.

Over the fixed and undirected graph, the global consensus update process is taken as an input $u_i(t + 1)$ of the local training updated parameter $\theta_i(t + 1/2)$ for each agent i, which is described as:

$$u_i(t + 1) = G\sum_{j=1}^{K} a_{ij}(\theta_i(t + 1/2) - \theta_j(t + 1/2)), \tag{3.22}$$

where a_{ij} is the entries of the adjacency matrix $\mathcal{A}$ of the graph, G is the control parameter.

A normalized adjacent matrix $\mathcal{A}'$ and a normalized Laplacian matrix can be obtained by:

$$\mathcal{A}' = \frac{1}{d_{max}}\mathcal{A}, \tag{3.23}$$

$$\tilde{\mathcal{L}} = I_K - \mathcal{A}', \tag{3.24}$$

where d_{max} is the maximum degree of the graph, I_K is an identity matrix, with K representing the number of agents in the graph. $\tilde{\mathcal{L}}$ is the normalized Laplacian matrix, which is a diagonal matrix as a fixed and undirected graph is concerned in this chapter.

By substituting (3.22) and (3.24) to (3.20), we have,

$$\begin{aligned}\boldsymbol{\theta}(t+1) &= (I_K \otimes I_n + \tilde{\mathcal{L}} \otimes I_n G)\boldsymbol{\theta}(t+1/2)\\ &= (I_K \otimes I_n + \tilde{\mathcal{L}} \otimes I_n G)(\theta^* + \Delta\boldsymbol{\theta}(t+1))\\ &= (I_K \otimes I_n + \tilde{\mathcal{L}} \otimes I_n G)(\theta^* + A\Delta\boldsymbol{\theta}(t))\\ &= I_K \otimes \theta^* + (I_K \otimes I_n + \tilde{\mathcal{L}} \otimes I_n G)A\Delta\boldsymbol{\theta}(t).\end{aligned} \tag{3.25}$$

Given that the θ^* is constant, we can define a new state function as:

$$\begin{aligned}\Delta\boldsymbol{\theta}(t+1) &= (I_K \otimes I_n + \tilde{\mathcal{L}} \otimes I_n G)A\Delta\boldsymbol{\theta}(t)\\ &= (I_K \otimes A + \tilde{\mathcal{L}} \otimes AG)\Delta\boldsymbol{\theta}(t).\end{aligned} \tag{3.26}$$

The left and right eigenvalue corresponding to eigenvalue 0 of the normalized Laplacian matrix $\tilde{\mathcal{L}}$ are r^T and $\mathbf{1}$, respectively, which satisfies $r^T\mathbf{1} = 1$. We can then perform state transform on (3.26) by $\xi(t) = (M \otimes I_n)\Delta\boldsymbol{\theta}(t)$, where $M = (I_K - \mathbf{1}r^T)$ and the ith row of $\xi(t)$ represents the error between each agent $\Delta\theta_i$ and the mean of all the agents $(\frac{1}{K}\sum_{i=1}^{K}\Delta\theta i)$.

Given that $\Delta\boldsymbol{\theta}(t) = \xi(t) + \mathbf{1}r^T \otimes I_n\Delta\boldsymbol{\theta}(t)$ and $r^T\tilde{\mathcal{L}} = 0$, we can get:

$$\begin{aligned}\xi(t+1) &= (M \otimes I_n)\Delta\boldsymbol{\theta}(t+1)\\ &= ((I_K - \mathbf{1}r^T) \otimes I_n)(I_K \otimes A + \tilde{\mathcal{L}} \otimes AG)\Delta\boldsymbol{\theta}(t)\\ &= (I_K \otimes A + \tilde{\mathcal{L}} \otimes AG)\xi(t),\end{aligned} \tag{3.27}$$

Then, a transform matrix T can be found to satisfy that $T^{-1}\tilde{\mathcal{L}}T = \Lambda$, where Λ is the diagonal form of $\tilde{\mathcal{L}}$, the first column of T is $\mathbf{1}$ and the first row of T^{-1} is r. Let $\eta = (T^{-1} \otimes I_n)\xi$, the equation (3.27) can be converted to:

$$\eta(t+1) = (I_K \otimes A + \Lambda \otimes AG)\eta(t). \tag{3.28}$$

And then,

$$\eta_i(t+1) = (I_K \otimes A + \lambda_i AG)\eta_i(t), \tag{3.29}$$

where λ_i denotes the ith eigenvalue of the diagonal matrix Λ, which is exactly the ith diagonal element of Λ. $\eta_i(t)$ is the ith row of matrix $\eta(t)$.

Given that $\lambda_1(\tilde{\mathcal{L}}) = 0$ and A is neutrally stable, we thus have $\eta_1(t+1) = A\eta_1(t) \to 0$.

To design G, we set the Lyapunov function $V(t)$ as:

$$V(t) = \eta(t)^T (I_K \otimes P)\eta(t), \tag{3.30}$$

where P is a positive definite matrix.

For $i = 2, 3...K$, it can be verified that

$$\begin{aligned} & V_i(t+1) - V_i(t) \\ &= \eta_i(t)^T (A + \lambda_i AG)^T P (A + \lambda_i AG)\eta_i(t) - \eta_i(t)^T P \eta_i(t) \\ &= \eta_i(t)^T [(A + \lambda_i AG)^T P (A + \lambda_i AG) - P]\eta_i(t). \end{aligned} \tag{3.31}$$

Setting $G = -(A^T PA + I)^{-1} A^T PA$, then

$$\begin{aligned} & (A + \lambda_i AG)^T P (A + \lambda_i AG) - P \\ &= A^T PA - 2\lambda_i A^T PA(A^T PA + I)^{-1} A^T PA - P \\ &\quad + (\lambda_i)^2 A^T PA(A^T PA + I)^{-1} \mathbf{M_1} \\ &= A^T PA + [-2\lambda_i + (\lambda_i)^2] A^T PA(A^T PA + I)^{-1} A^T PA \\ &\quad + (\lambda_i)^2 A^T PA(A^T PA + I)^{-1} \mathbf{M_2} A^T PA - P \\ &= A^T PA + [-2\lambda_i + (\lambda_i)^2] A^T PA(A^T PA + I)^{-1} A^T PA \\ &\quad - (\lambda_i)^2 A^T PA(A^T PA + I)^{-2} A^T PA - P \\ &\leq A^T PA + [-2\lambda_i + (\lambda_i)^2] A^T PA(A^T PA + I)^{-1} A^T PA \\ &\quad - P, \end{aligned} \tag{3.32}$$

where $\mathbf{M_1} = A^T PA(A^T PA + I)^{-1} A^T PA$, $\mathbf{M_2} = [-I_n + A^T PA(A^T PA + I)^{-1}]$.

Lemma 1: By Gersgorin Circle Criterion [123], the range of $\lambda(\tilde{\mathcal{L}})$ should be restricted in a disc, that is,

$$|(\lambda - \tilde{\mathcal{L}}_{ii})| \leq \sum_{j=1, j\neq i}^{K} |(\tilde{\mathcal{L}}_{ij})|. \tag{3.33}$$

It can be deduced from (3.24) that the entries of the normalized Laplacian $\tilde{\mathcal{L}}_{ij} \in [0, 1)$, all the eigenvalues of the normalized Laplacian $\tilde{\mathcal{L}}$ should, therefore, locate within a disc centered at 1 with radius of 1, that is, $\lambda(\tilde{\mathcal{L}}) \in (0, 2)$, and then $[-2\lambda_i + (\lambda_i)^2] < 0$.

Therefore, with the condition that A is neutrally stable, we can get:

$$\begin{aligned} & V_i(t+1) - V_i(t) \\ &= \eta_i(t)^T(A+\lambda_i AG)^T P(A+\lambda_i AG)\eta_i(t) - \eta_i(t)^T P\eta_i(t) \\ &= \eta_i(t)^T[A^T PA + [-2\lambda_i + (\lambda_i)^2]\mathbf{M_1} - P]\eta_i(t) \\ &< 0. \end{aligned} \tag{3.34}$$

The matrix P should satisfy the following modified algebraic Riccati equation (MARE) [123]:

$$P = A^T PA - (1-\delta^2)A^T PA(A^T PA + I)^{-1}A^T PA + O, \tag{3.35}$$

where O is a positive definite matrix, and $0 < \delta < 1$.

Therefore, we can conclude that $\Delta\theta(t)$ will gradually converge to zero as the increase of iteration t using the proposed distributed training method, and all agents would converge to a unique model with their parameters $\theta_i (i = 1, 2..K)$ converging to the optimal parameter θ^*. This completes the proof. ■

Remark 4: The normalized Laplacian matrix $\tilde{\mathcal{L}}$ is a symmetric matrix and can be transferred to the diagonal form Λ, because a fixed and undirected graph is concerned in this chapter. However, in the case of the directed graph such that the normalized Laplacian matrix $\tilde{\mathcal{L}}$ cannot be transferred to the diagonal form Λ, we can obtain its Jordan form, which will still make the above convergence analysis hold [124].

Remark 5: Under Assumption 3, A is a constant matrix. However, the matrix A is more likely to be varying with iterations in real simulation and application. In this case, the above proof still holds, if only $A(t)$ is neutrally stable at all iterations, and the convergence property of finite products of SIA (stochastic, indecomposable, aperiodic) matrices [125] can be used to support this proof.

Based on the above analysis, all empirical risks $E_k (i = 1, 2, ..., K)$ converge to the minimal empirical risk E^* and all model parameters $\theta_i (i = 1, 2, ..., K)$ converge to the optimal model parameter θ^*, which implies the same conclusion that the proposed distributed neural networks can converge to the same optimal model.

3.4 Simulation and Discussion

The proposed distributed training algorithm for multi-layer neural networks contains a two-phase update procedure. The first phase is training with local sub-dataset, which is performed simultaneously at all agents over the graph, that is, multiple neural networks with the same structure are trained only with their own local sub-dataset. In the second phase, these locally updated neural networks communicate with their directly connected neighbors to globally update their model parameters using the consensus algorithm. As described in the above section, this two-phase update process allows all the agents to converge to the optimal model, and these models would show comparable performance with a centralized training model based on the entire dataset.

Besides the fact that there is no need to exchange the data samples or collected all the data samples centrally, another main advantage of this proposed distributed training algorithm is its simple structure and great expansibility. That is, it will not affect the convergence and effectiveness of this method when a new agent joins in or leaves as long as it always exists a spanning tree over the graph, which is also suitable for changeable graph topology [126].

The decentralized graph in Figure 3.1(b) is taken as an example, where the entire dataset is partitioned and distributed evenly on six agents, and six neural networks with the same structure are also initialized on each agent. For distributed training, the information of the model parameter is allowed to be shared only among directly connected neighbors, and there is no exchange of data samples. The following three algorithms are compared to verify the effectiveness of the proposed distributed training method.

- **Centralized training**: this is a single neural network training with the entire dataset, which can be taken as a baseline for the distributed training method.
- **Distributed training**: in this case, the training dataset is distributed evenly on each agent over the decentralized graph, and each agent trains a neural network with its own sub-dataset, with the consensus algorithm globally updating their model parameters.

- **Local training**: as before, each agent only trains with its own sub-dataset without communication, accuracy or error is averaged throughout the nodes.

In all of the above algorithms, Relu function is taken as the hidden layer activation function, with model parameter extracted randomly from a uniform distribution over the interval $[-1, +1]$, and all the input variables are normalized between 0 and 1. Classification Accuracy and Mean Absolute Percentage Error (MAPE) are used to evaluate the performance of these algorithms for the task of classification and regression, respectively. The whole algorithms are implemented in Tensorflow and Matlab.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}|\frac{y_i - \hat{y}_i}{y_i}| \tag{3.36}$$

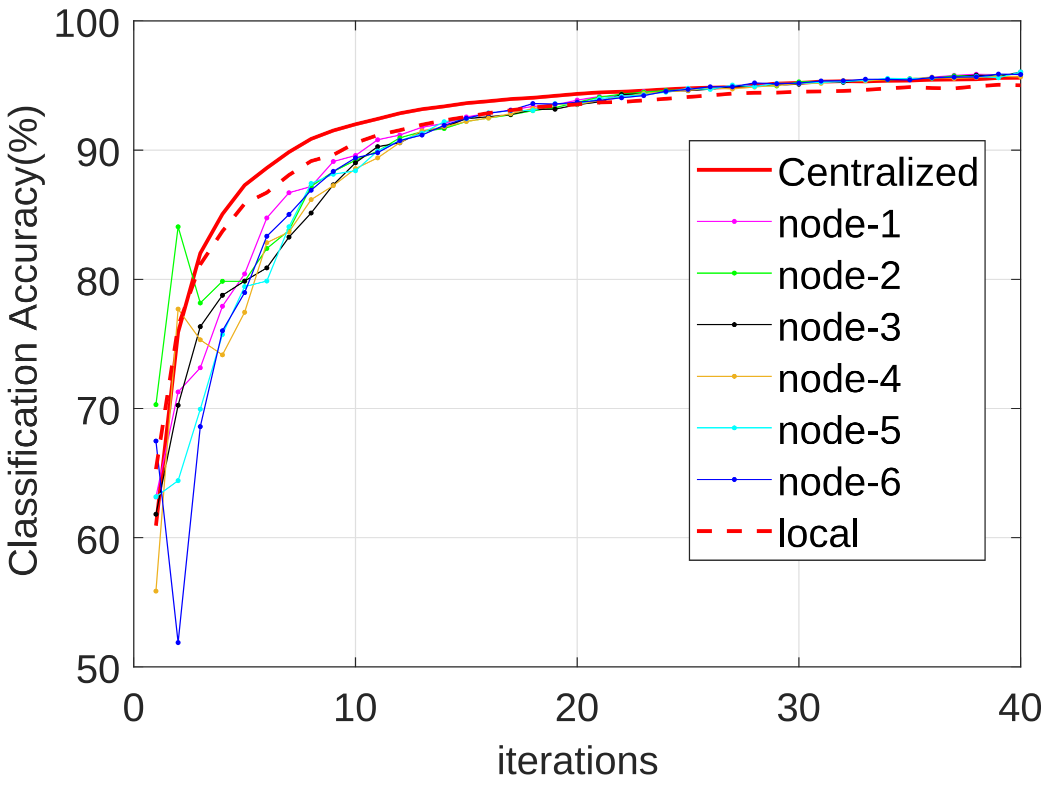
where n denotes the number of samples, y_i and $\hat{y}_i$ are the actual and estimated value of the ith sample, respectively.

The proposed distributed training algorithm is tested on three public datasets (downloaded from UCI open datasets) for the tasks of binary classification, multi-labeled classification, and regression, respectively. The characteristics of these datasets and modeling parameters are summarized in Table 3.2, where η and α are the learning rate and regularization coefficient of L_2 regularization method [127].

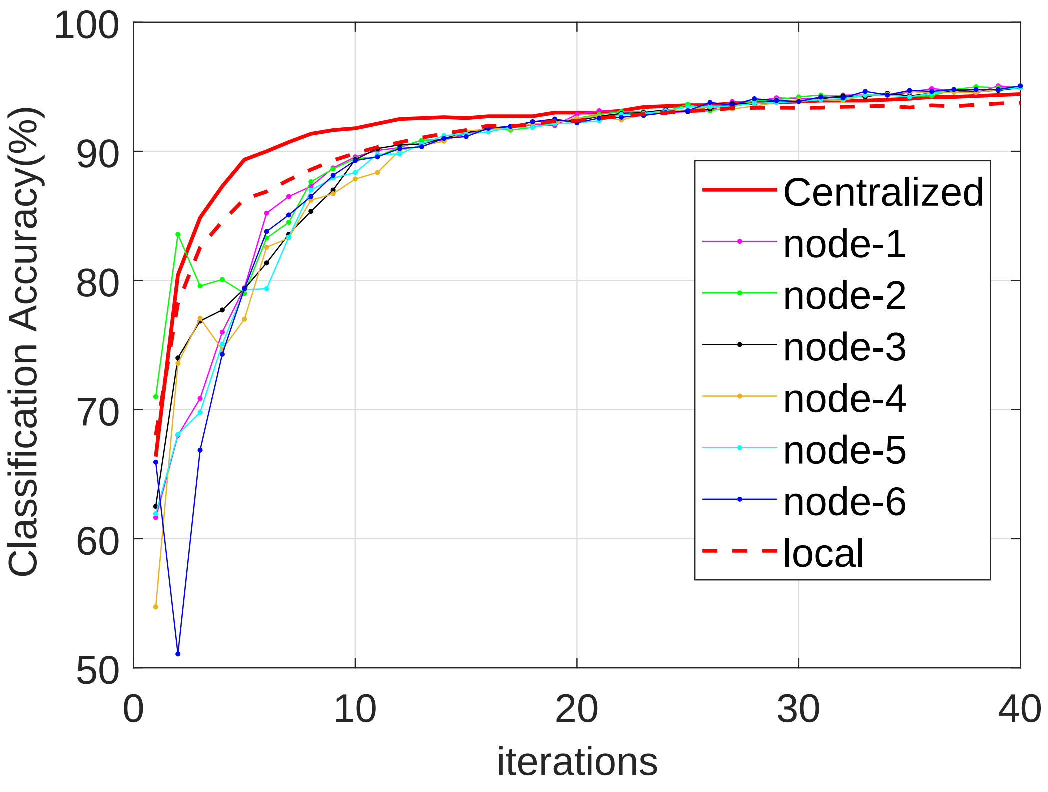
Table 3.2: The description of the datasets and modeling parameters

Dataset	twonorm	pendigits	cpusmall
Features	20	16	12
Train samples	6000	6000	6000
Test samples	1400	1494	2192
Hidden nodes	100	200	100
α	0.1	0.01	0.1
η	0.08	0.2	0.0001
Task	Classification (2 classes)	Classification (10 classes)	Regression

Figures 3.2 to 3.4 are the simulation results on the above three datasets in Table 3.2, where we can find that all of the six agents over the graph can exhibit a comparable performance with centralized training using the proposed distributed training algorithm, even though each agent of distributed training shows more fluctuations and

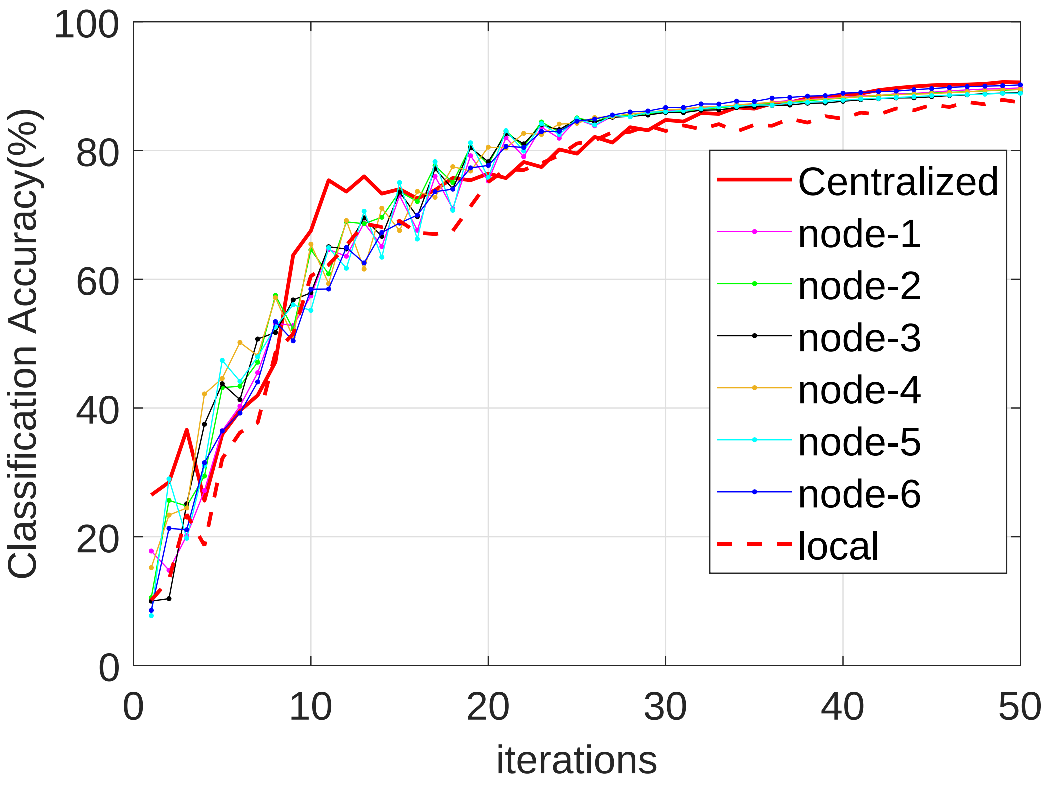


(a) Train

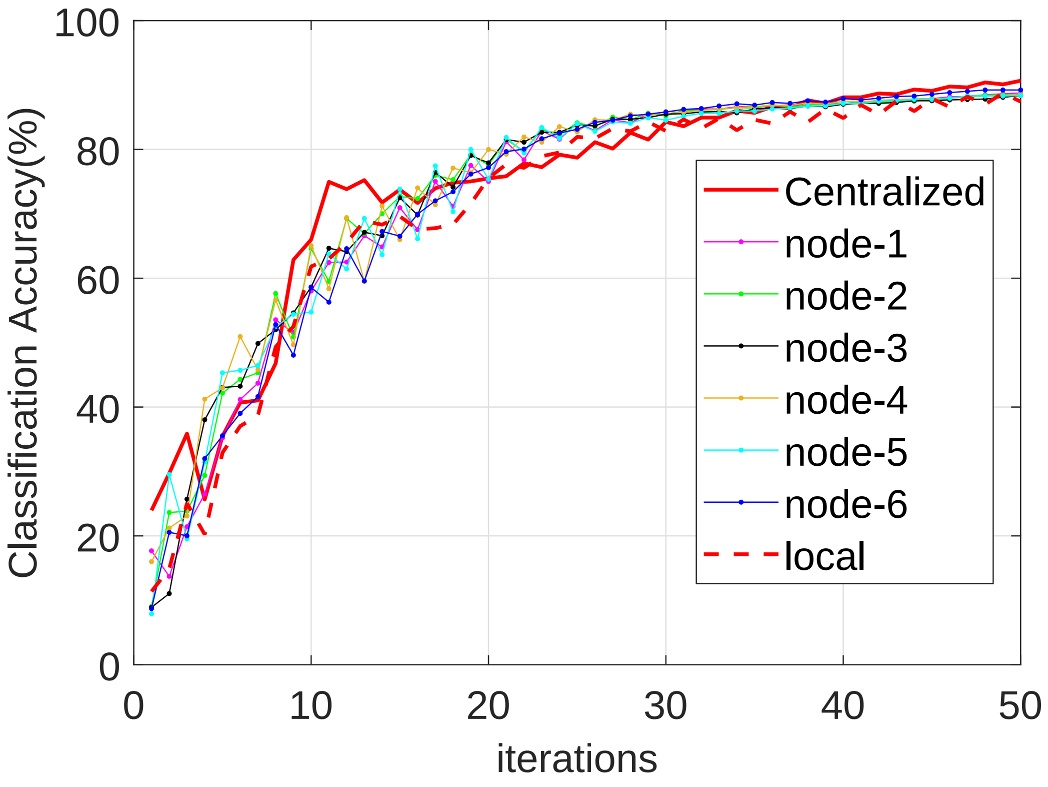


(b) Test

Figure 3.2: The performance on dataset twonorm.

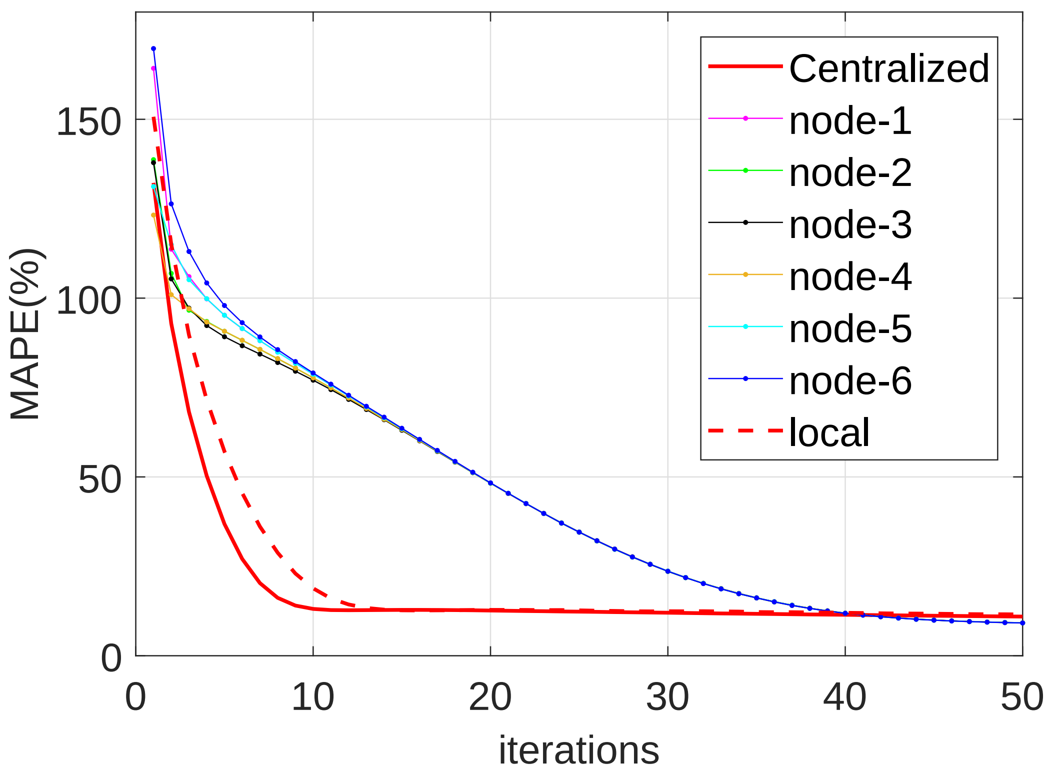


(a) Train

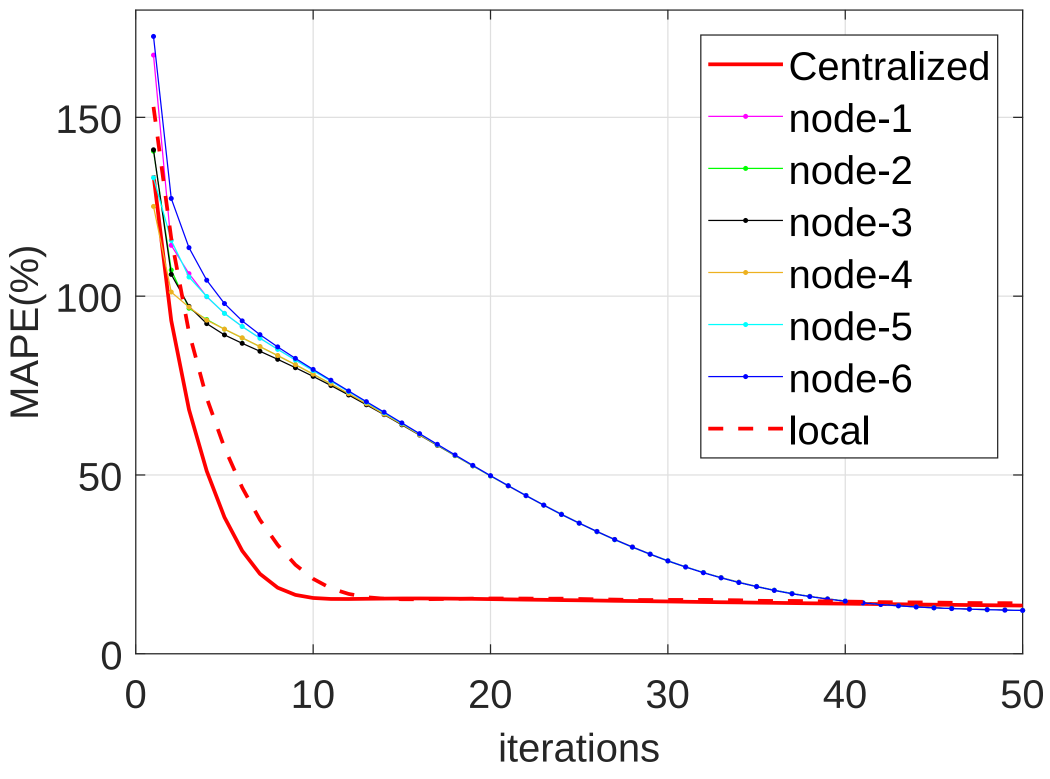


(b) Test

Figure 3.3: The performance on dataset pendigits.



(a) Train



(b) Test

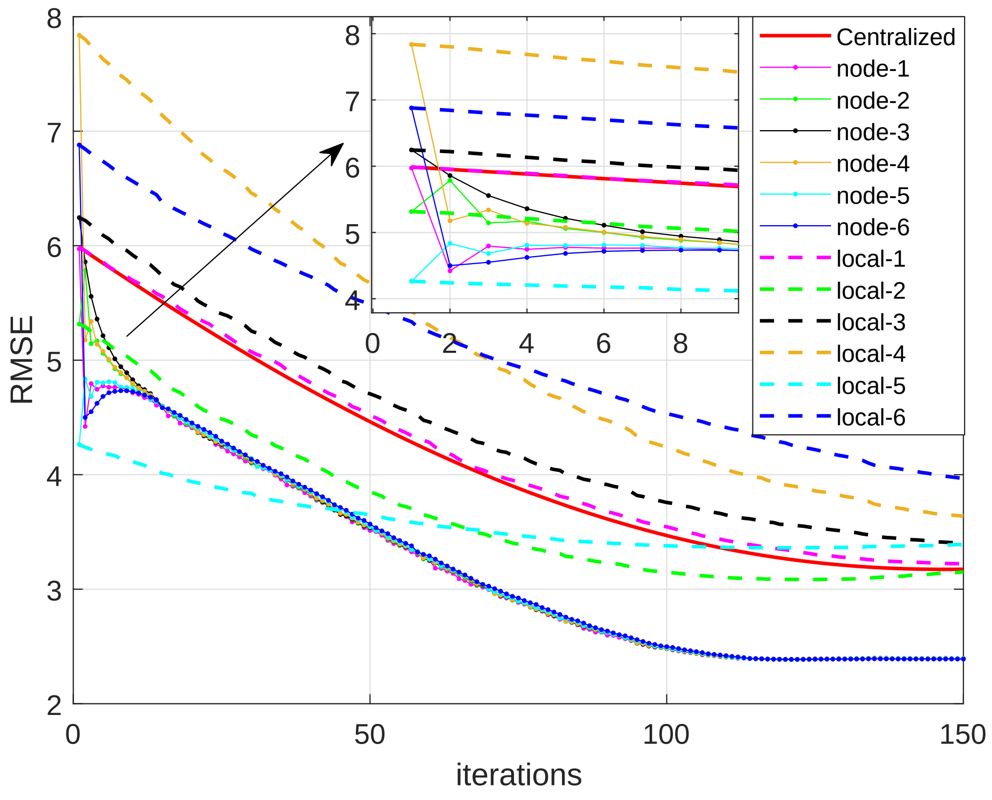
Figure 3.4: The performance on dataset cpusmall.

worse accuracy at the initial iterations. Local training shows only a little worse performance than centralized training and distributed training, as these three problems are relatively easy. It is worth noting that all the agents gradually converge to the same model during a certain number of iterations, although their initial statuses are significantly different, which can verify the theorems proposed in this chapter. For the classification tasks, twenty and thirty steps are required for all the agents to achieve consensus for the binary and multi-labeled classification, respectively, and a little more steps are needed to allow all the agents to exhibit a similar classification accuracy as the centralized training model. As for the task of regression, the six agents converge to each other after fifteen iterations, while more iterations are required for them to catch up with and even exceed the performance of the centralized training model.

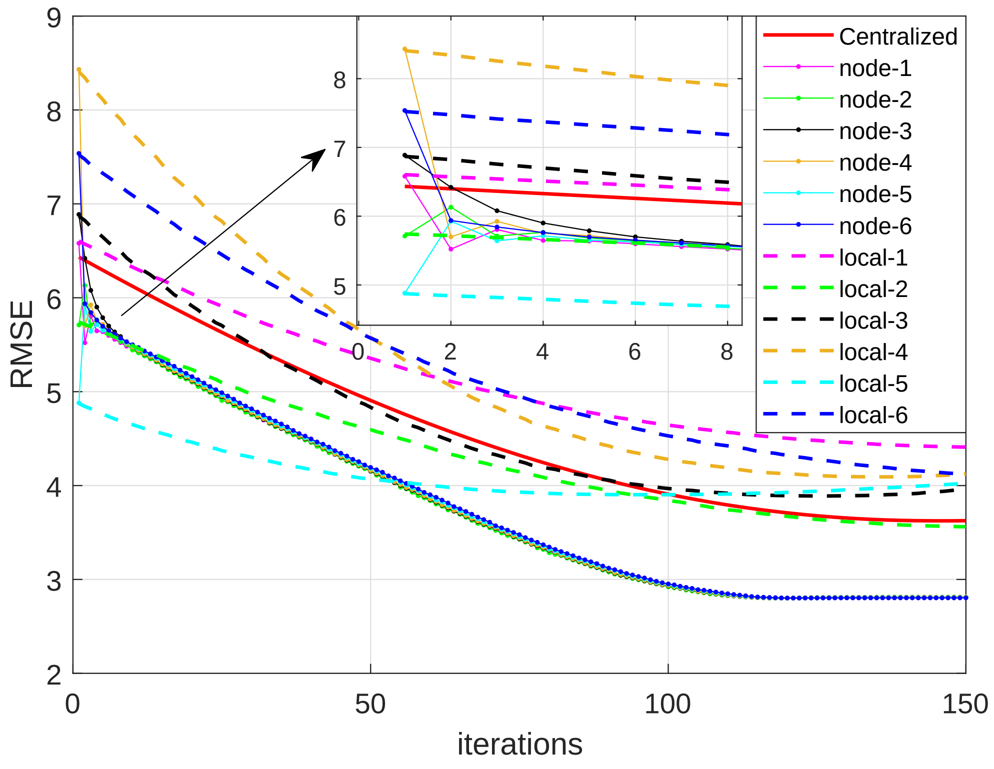
Other optimization methods can also be taken for gradient descent and modeling hyper-parameters (hidden agents, regularisation and learning rate) to further improve the performance of distributed training, but it is out of the scope of this chapter. Common modeling methods are used in this chapter for the convenience of the comparison between distributed training and centralized training.

As the above datasets are easy to fit, local training can also have good performance. To further verify the superiority of distributed training, the proposed algorithm is tested on a large-scale dataset BlogFeedback (downloaded from the UCI Machine Learning Repository) which contains more than 50000 samples and 280 features for the regression task. In this case, each agent has 6000 samples, and the rest samples are taken as the test dataset, with Root Mean Squared Error (RMSE) used to evaluate their performance. A head to head comparison is made between distributed training and local training, the performance of which is shown in Figure 3.5, where node-i and local-i (i=1,2,...6) denote the performance of agents with and without the consensus process, respectively.

We can find from Figure 3.5 that all the agents in distributed training can converge to the same optimal model after 110 iterations, which shows much better performance than centralized training and local training. The training process helps each agent to find a better solution in its basin of attraction, while the consensus process not only



(a) Training dataset



(b) Test dataset

Figure 3.5: The comparison between distributed training and local training.

decreases the upper bound of the parameter gaps between each agent and the optimal model, but also helps agents escape the basin of attraction with a local minimum.

The partially magnified figure in Figure 3.5 specifically shows the influence of the consensus process on each agent, where all agents converge to each other in 10 iterations even though their initial statuses are significantly different. All locally trained agents without consensus exhibit a slow converge rate individually, while all agents with the consensus process approach each other significantly during the first a few steps and then converge to the optimal model in union. This attraction of each other drives all agents to a good status in a few consensus steps and to converge to the optimal model faster in the long run, even though some agents (node-2 and node-5 in this experiment) get worse at the beginning steps.

Generalization capability is a key factor for a trained model, which represents the performance of a model on the data that it never learns from. Usually, we use a testing dataset to evaluate the generalization capability of a model. From Figures 3.2 to 3.5, we find that the distributed training model shows great performance on testing dataset for all of the four simulation experiments, which is comparable to the performance of the model on training dataset. This verified the great generalization capability of the proposed distributed training method.

3.5 Conclusion

Distributed training has received great attention over the last decade due to its wide real-world applications on large-scale and privacy-concerned machine learning problems. It is common nowadays that all the data samples cannot be collected centrally and the exchange of data samples is not allowed for computation restrictions or privacy protection. However, the model needs to learn from the entire dataset instead of training with only local sub-dataset. For this problem, this chapter proposed a distributed training method using the consensus algorithm for multi-layer neural networks over a decentralized graph without a central agent. Theoretical analysis of distributed neural networks shows that the performance of distributed training could exhibit nearly the same performance with centralized training when enough consensus steps are taken, but this method is still computationally expensive. Furthermore, convergence analysis gives the proof that distributed training allows all the agents over a decentralized graph to converge to the optimal model even with only a single consensus step after each training iteration, which could greatly decrease the communication cost. This theorem is verified by simulation, which demonstrates that the proposed distributed training algorithm for the multi-layer neural networks can achieve comparable or even better performance as the centralized training model based on the entire dataset.

Chapter 4

Distributed Heuristic Adaptive Neural Networks with Variance Reduction in Switching Graphs

4.1 Introduction

Chapter 3 has proposed the distributed training framework for multi-layer neural networks, where the weighted connectivity matrix is fixed during the distributed training process and the standard gradient descent method is used to optimize these neural networks. Building on the above chapter, we are expected to optimize the consensus process by adaptively adjust the weighted connectively matrix and optimize the learning process to reduce the computational costs.

In this chapter, we propose a distributed heuristic adaptive training algorithm in switching decentralized communication graphs, and similarly, a single consensus step is required among connected agents after each training iteration. Besides, the stochastic variance reduced gradient is used to optimize the neural networks on each agent to improve its convergence rate. After the analysis of the consensus-based distributed neural networks with a fixed weighted connectivity matrix, we design the heuristic adaptive consensus algorithm to adaptively adjust the weighted connectivity matrix,

which speeds up the agents to reach the optimum. The theoretical analysis and simulation experiments give the result that the proposed heuristic adaptive consensus algorithm drives all agents to the optimal model faster, and the stochastic variance reduced gradient can greatly reduce the fluctuation caused by stochastic gradient descent and improve its convergence rate. The main contributions can be summarized as follows:
1) The heuristic adaptive consensus algorithm is proposed to adaptively adjust the weighted connectivity matrix for distributed training neural networks;
2) The proposed method is suitable for the switching communication graphs, which is robust to the failure of any agent.

The remainder of this chapter is organized as follows. Section 4.2 designs the heuristic adaptive consensus algorithm for distributed training; Section 4.3 presents the stochastic variance reduced gradient with comparisons to batch gradient descent and stochastic gradient descent; Section 4.4 analyses the convergence of the distributed adaptive training algorithm in switching communication topology; Section 4.5 conducts simulation experiment and compares four algorithms on a UCI dataset, which demonstrate the superiority of the proposed distributed adaptive neural networks; Section 4.6 concludes this chapter.

4.2 Heuristic Adaptive Consensus Algorithm

As explained in the above chapter, the weighted connectivity matrix W of the consensus algorithm is fixed throughout the training process, which means that an agent's influence on its neighbors is constant and only depends on the graph topology and the consensus strategy. This results in that all agents would approach each other equally during the consensus process, no matter how good or bad an agent's performance is.

A more reasonable idea is that the agent with better performance should exhibit more influence on its neighbors, that is, all the agents should approach a larger step to its better neighbors. Here, the root mean squared error (RMSE) is applied to assess the performances of the agents. The straightforward idea is that, the smaller the error of an agent is, the larger value it should have in the weighted connectivity matrix W. For a communication topology, we define a connectivity matrix $D = [d_{ij}] \in \mathbb{R}^{K\times K}$, where $d_{ij} = 1$ if the edge $(i, j) \in \mathcal{E}$ or $i = j$, otherwise $d_{ij} = 0$. The weight matrix $C = [c_{ij}] \in \mathbb{R}^{K\times K}$ in terms of the $RMSE = \sqrt{E(\mathcal{D}_i, \theta_i)}$ of all agents is given by

$$c_{ij} = \frac{d_{ij}e_j}{\sum_{j=1}^{K} a_{ij}e_j}, \tag{4.1}$$

$$e_i = \frac{1}{\sqrt{E(\mathcal{D}_i, \theta_i)}}, \tag{4.2}$$

where e_i represents the reciprocal of the $RMSE$ of agent i.

However, if the weight matrix C directly depends on their performance, all agents will immediately converge to the currently better agent. This may lose the agents' search capability and impair its robustness. For example, in the situation that the currently best agent is in a local minimum, all other agents are more likely to be attracted into this basin of attraction with the local minimum based on the weight matrix C. The idea from particle swarm optimization (PSO) [128, 129] is therefore introduced to design the weighted connectivity matrix W, where an agent with the smaller error only leads to the bigger possibility of obtaining a large value in the weighted connectivity matrix. Based on this, we can obtain the weighted connectivity matrix W by the combination of a bounded random matrix $B = [b_{ij}] \in \mathbb{R}^{K\times K}$ and the weight matrix C.

$$w_{ij} = b_{ij}c_{ij}, \tag{4.3}$$

$$b_{ij} = \sigma\alpha + (1 - \sigma), \tag{4.4}$$

where $\alpha \in [0, 1]$ is a random number, $\sigma \in [0, 1]$ is a user-defined parameter to determine the bound of the random matrix B.

In this way, $w_{ij} \in [(1 - \sigma)b_{ij}, b_{ij}]$ is a random number defined by b_{ij}. Due to the fact that the better the performance of an agent is, the larger b_{ij} is, the interval $[(1 - \sigma)b_{ij}, b_{ij}]$, as well as the expectation of w_{ij}, thus adaptively changes according to the performances of agent i and its directly connected neighbors. Even though that w_{ij} is a random number within this interval, all agents will approach larger steps to their better neighbors in the long run, and reserve the searching ability in the domain based on all agents by the randomness.

It is notable that the sum of any row of the matrix W obtained from (4.3) may be smaller than 1, that is, $\sum_{j=1}^{K} w_{ij} = \sum_{j=1}^{K} b_{ij}c_{ij} \leq 1$ because $b_{ij} \leq 1$ and $\sum_{j=1}^{K} c_{ij} = 1$, and this will result in divergence. To satisfy the requirement that $\sum_{j=1}^{K} w_{ij} = 1$, $\forall i$, we define another weight parameter β_i, which is allocated to the historical best model parameter of agent i up to the current iteration. Then, we get the new adaptive update rule for θ_i',

$$\beta_i = 1 - \sum_{j=1}^{K} w_{ij}, \tag{4.5}$$

$$\theta_i'' = \sum_{j=1}^{K} w_{ij}\theta_j' + \beta_i\theta_i'^{*}, \tag{4.6}$$

where θ_i'' denotes the optimized model parameter of θ_i' using the heuristic adaptive consensus algorithm, with $\theta_i'^{*}$ being the historical best model parameter of agent i.

This heuristic adaptive consensus algorithm is more efficient and robust, compared with the consensus algorithm with a fixed weighted connectivity matrix. First, the value of the weight matrix C depends on the performance of all agents, which means that each agent will adaptively approach a larger step to its better neighbors. Second, the introducing of the bounded random matrix B keeps the agents' random searching ability during the consensus process, where the tunable parameter σ can adjust the agents' local and global searching ability by defining the bound of matrix B. Third, the consideration of the historical best model parameter $\theta_i'^{*}$, to some extent, avoids the agent getting worse.

4.3 Stochastic Gradient Descent with Variance Reduction

Batch gradient descent methods for the training of a neural network is of great computation cost, especially when it comes to large-scale machine learning problems. Faced with this problem, stochastic gradient descent (SGD) is an excellent alternative method of gradient descent to reduce the computation cost and improve its efficiency, which is also widely used to train the parameters of a model in distributed systems [107,130]. The local gradient of a randomly selected sample from the training dataset was used to estimate the global average gradient, stochastic gradient descent (SGD) can therefore greatly decrease the computation cost but introduce the variance for the reason that there is a significant difference between the global average gradient and local gradient obtained from a single sample even though they are equivalent in expectation. Too large variance introduced by SGD would impair the convergence performance, although the decaying learning rate can be used to alleviate this negative influence [131].

To improve the performance of SGD, those methods that help to reduce the variance are of great interest to be designed, which contributes to better performance and allows a relatively larger learning rate. The methods proposed by Roux et al. [132] and Shalev-Shwartz and Zhang [133] can achieve such variance reduction effect for SGD, which could lead to a linear convergence when the objective function is smooth and strongly convex. These methods are suitable for training convex linear prediction problems, such as logistic regression and least squared regression. In references [134–136], a new variance reduction technique was presented to speed up the convergence of SGD, which could keep SGD converging at a constant rate. However, these methods are designed to be used in a centralized learning system instead of the distributed learning system.

4.3.1 Batch Gradient Descent

Back-propagation is a popular method for the training of neural networks to minimize the empirical risk on a dataset, which consists of two iterative steps. That is, the computation of empirical risk and the optimization of model parameter using the gradient descent method.

$$E(\mathcal{D}, \theta) = \frac{1}{n} \sum_{i=1}^{n} l(d_i, \theta), \tag{4.7}$$

$$l(d_i, \theta) = \frac{1}{2} (\hat{y}_i - y_i)^2, \tag{4.8}$$

$$\hat{y}_i = f(x_i, \theta), \tag{4.9}$$

where n is the number of data samples of $\mathcal{D}$, $E(\mathcal{D}, \theta)$ and $l(d_i, \theta)$ are the empirical risks over the model parameter θ on the dataset $\mathcal{D}$ and a sample $d_i = (x_i, y_i)$, respectively. f is a function over the model parameter θ, which maps x_i to the y_i, with $\hat{y_i}$ being the estimated output.

Batch gradient descent [137] is a standard method to optimize the parameter θ, the update rule of which is

$$\theta' = \theta - \eta \nabla E(\mathcal{D}, \theta), \tag{4.10}$$

$$\nabla E(\mathcal{D}, \theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla E(d_i, \theta) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta} l(d_i, \theta), \tag{4.11}$$

where θ' is the trained model parameter of θ, with η representing the learning rate. $\nabla E(d_i, \theta)$ denotes the gradient on the sample d_i over the model parameter θ.

4.3.2 Stochastic Variance Reduced Gradient

We can find from (4.11) that n derivatives need to be computed at each iteration for batch gradient descent, which could be computationally expensive and time-consuming, especially when the entire dataset is large. For this problem, stochastic gradient descent (SGD) [138] is a good alternative method, which randomly selects a sample d_i $(i = 1, 2, ..., n)$ from the entire dataset to approximate the full gradient $\nabla E(\mathcal{D}, \theta)$. The update rule of SGD is

$$\theta' = \theta - \eta \nabla E(d_i, \theta). \tag{4.12}$$

It is important to point out that the expectation $\mathbb{E}[\cdot]$of the stochastic gradient $\nabla E(d_i, \theta)$ in (4.12) is the same as the full gradient in (4.10) because

$$\mathbb{E}[\nabla E(d_i, \theta)] = \frac{1}{n} \sum_{i=1}^{n} \nabla E(d_i, \theta) = \mathbb{E}[\nabla E(\mathcal{D}, \theta)]. \tag{4.13}$$

By estimating the full gradient using a single randomly selected sample at each iteration, SGD can greatly reduce the computational cost but incur the variance because of randomness. This kind of randomness may be helpful for the algorithm to escape from the local minimum, while too large variance also impairs its convergence rate, especially when the learning rate is large. For this problem, the stochastic variance reduced gradient (SVRG) [134] method is used to reduce the variance, where an estimated parameter $\tilde{\theta}$ and its corresponding full gradient $\nabla E(\mathcal{D}, \tilde{\theta})$ are kept as a reference, and the gradient v of SVRG is calculated by the following equation

$$v = \nabla E(d_i, \theta) - \nabla E(d_i, \tilde{\theta}) + \nabla E(\mathcal{D}, \tilde{\theta}), \tag{4.14}$$

$$\nabla E(\mathcal{D}, \tilde{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla E(d_i, \tilde{\theta}), \tag{4.15}$$

where d_i represents a randomly selected sample, the $\nabla E(d_i, \theta)$ is the stochastic gradient of the random sample d_i over the current model parameter θ, $\nabla E(d_i, \tilde{\theta})$ is the stochastic gradient of the random sample d_i over the estimated model parameter $\tilde{\theta}$, and $\nabla E(\mathcal{D}, \tilde{\theta})$ is the full gradient on the entire dataset over the estimated model parameter $\tilde{\theta}$.

In (4.14), the item $\nabla E(d_i, \tilde{\theta})$ is to reduce the variance caused by randomness through the minus of $\nabla E(d_i, \theta)$ by $\nabla E(d_i, \tilde{\theta})$, and the full gradient item $\nabla E(\mathcal{D}, \tilde{\theta})$ is to offset $\nabla E(d_i, \tilde{\theta})$ because they are the same in expectation as explained in (4.13). We thus have that the gradient v of SVRG is the same as the stochastic gradient in expectation, that is

$$\begin{aligned} \mathbb{E}[v] &= \mathbb{E}[\nabla E(d_i, \theta) - \nabla E(d_i, \tilde{\theta}) + \nabla E(\mathcal{D}, \tilde{\theta})] \\ &= \mathbb{E}[\nabla E(d_i, \theta)]. \end{aligned} \tag{4.16}$$

Notably, the full gradient does not need to be computed at every iteration but kept fixed over m iteration, which only leads to less extra computational cost compared with SGD. More specifically, the computational complexity of batch gradient descent and SGD are $\mathcal{O}(n)$ and $\mathcal{O}(1)$ for each iteration, respectively, while that is $\mathcal{O}(2 + \frac{n}{m})$ for SVRG.

To analyze the convergence of (4.14), suppose that both $\tilde{\theta}$ and θ converge to the optimal θ^* after enough iterations, then $\nabla E(\mathcal{D}, \tilde{\theta}) \longrightarrow 0$ and $\nabla E(d_i, \tilde{\theta}) \longrightarrow \nabla E(d_i, \theta)$, therefore $\nabla E(d_i, \theta) - \nabla E(d_i, \tilde{\theta}) + \nabla E(\mathcal{D}, \tilde{\theta}) \longrightarrow \nabla E(d_i, \theta) - \nabla E(d_i, \tilde{\theta}) \longrightarrow 0$.

4.4 Convergence Analysis

In this section, it will be proved that, even though with a single consensus communication at every training step, the proposed distributed heuristic adaptive neural networks ensure all agents converge to the optimum. This also holds in the switching graphs and the situation that any new agent quits or participates in the communication graph.

Assumption 1: The training process $\mathcal{T}$ based on any dataset $\mathcal{D}_i$ from the entire dataset $\mathcal{D}$ would lead to a contraction of the empirical risk, that is

$$E(\mathcal{D}, \theta_i') = E(\mathcal{D}, \mathcal{T}(\mathcal{D}_i, \theta_i)) \leq \mu E(\mathcal{D}, \theta_i),\ 0 < \mu < 1, \tag{4.17}$$

where $E(\mathcal{D}, \theta_i,)$ is the empirical risk of the ith neural network with parameter θ_i over the entire dataset, θ_i' represents the updated θ_i by the training process $\mathcal{T}$ with dataset $\mathcal{D}_i$. For simplicity, $E(\mathcal{D}_i, \theta_i)$ and $E(\mathcal{D}_i, \theta_i')$ are aliased as E_i and E_i', respectively.

Assumption 2: $E(\mathcal{D}, \theta)$ is a convex function about θ, with θ^* and E^* being the optimal solution and the minimal empirical risk, respectively.

As shown in (4.6), θ_i'' is the updated model parameter of θ_i' using the heuristic adaptive consensus algorithm, the corresponding empirical risk is

$$E(\theta_i'') = E(\sum_{j=1}^{K} w_{ij}\theta_j' + \beta_i \theta_i'^*),\ i = (1, 2, ..., K), \tag{4.18}$$

where $E(\theta_i'')$ represents the empirical risk over the model parameter θ_i'', which is aliased as E_i''.

Proposition 1: Based on the condition that the weighted connectivity matrix satisfies $w_{ij} \in [0, 1)$ and $\sum_{j=1}^{K} w_{ij} + \beta_i = 1$ $(i, j = 1, 2, ..., K)$, we have

$$\|(E_1'', E_2'', ...E_N'')\|_\infty \leq \|(E_1', E_2', ...E_N')\|_\infty, \tag{4.19}$$

where $\|\cdot\|_\infty$ denotes the max-norm, the '=' holds when $E_i' = E_j', \forall i, j$. Here, we define $\mathcal{F}' = (E_1', E_2', ...E_N')$ and $\mathcal{F}'' = (E_1'', E_2'', ...E_N'')$.

Proof:

$$\begin{aligned}
&\|(E''_1, E''_2, ...E''_N)\|_\infty \\
&= \max_k \|E''_k\| \\
&= \max_k \|E(\sum_{j=1}^{K} w_{kj}\theta'_j + \beta_k \theta'^*_k)\| \\
&\leq \max_k \|E(\sum_{j=1, j\neq k}^{K} w_{kj}\theta'_j + (w_{kk} + \beta_k)\theta'^*_k\| \\
&\leq \max_k \| \sum_{j=1, j\neq k}^{K} w_{kj}E(\theta'_j) + (w_{kk} + \beta_k)E(\theta'_k)\| \\
&\leq \max_k \|E(\theta'_k)\| \\
&= \|(E'_1, E'_2, ...E'_N)\|_\infty.
\end{aligned} \tag{4.20}$$

Remark 1: The switching of graph topology and the joining or leaving of a new agent change the value and dimension of the connectivity matrix D, respectively. However, these changes would only influence the value or dimension of the weighted connectivity matrix W, which still meets the condition that $\sum_{j=1}^{K} w_{ij} + \beta_i = 1$ ($i = 1, 2, ..., K$). Therefore, Proposition 1 still holds in switching communication graphs and the situation that any new agent quits or participates in the graph.

Theorem 1: Under Assumptions 1 and 2, all E_k converge to the minimum E^* in both fixed and switching graphs, in spite of their initial statues for the proposed distributed heuristic adaptive neural networks.

Proof: We define the maximal gap between the empirical risks of all agents $\mathcal{F} = (E_1, E_2, ...E_N)$ and the minimum E^* as $R(\mathcal{F})$, which is given by

$$\begin{aligned}
R(\mathcal{F}) &= \max_k(\|E_k\| - \|E^*\|) \\
&= \|(E_1, E_2, ...E_N)\|_\infty - \|E^*\|,
\end{aligned} \tag{4.21}$$

where it always holds that $\|E^*\| \leq \|E_k\|$.

For both the fixed graph and the switching graphs cases, by Proposition 1 to (4.21), we have

$$\begin{aligned} R(\mathcal{F}'') &= \max_k(\|E_k''\| - \|E^*\|) \\ &= \|(E_1'', E_2'', ...E_N'')\|_\infty - \|E^*\| \\ &\leq \|(E_1', E_2', ...E_N')\|_\infty - \|E^*\| \\ &= R(\mathcal{F}'). \end{aligned} \tag{4.22}$$

Based on (4.17), it is obtained

$$\begin{aligned} R(\mathcal{F}') &= \max_k(\|E_k'\| - \|E^*\|) \\ &\leq \mu\|E_k\| - \|E^*\| \\ &< \mu(\|(E_1, E_2, ...E_N)\|_\infty - \|E^*\|) \\ &< R(\mathcal{F}), \end{aligned} \tag{4.23}$$

where $0 < \mu < 1$. Therefore, we can get

$$R(\mathcal{F}'') \leq R(\mathcal{F}') < R(\mathcal{F}). \tag{4.24}$$

The equation (4.24) implies that the maximal gap $R(\mathcal{F})$ keeps descending and all agents' empirical risks $(E_1, E_2, ...E_N)$ will approach the minimum E^*, as the distributed training process carries on. ■

Remark 2: Assumption 1 is a common concept for neural network training problem, where a single or several training steps usually leads to a better model.

Remark 3: Assumption 2 is generally used to prove the convergence of machine learning problems [119, 120]. The convexity of the empirical risk guarantees the decline of the largest empirical risk among all agents with the consensus process. This idea may be also suitable for the non-convex neural networks optimization problem, which is validated in the numerical simulation.

4.5 Simulation and Discussion

There is a two-phase update process for the distributed heuristic adaptive neural networks, which consists of local training and global consensus. During the first phase

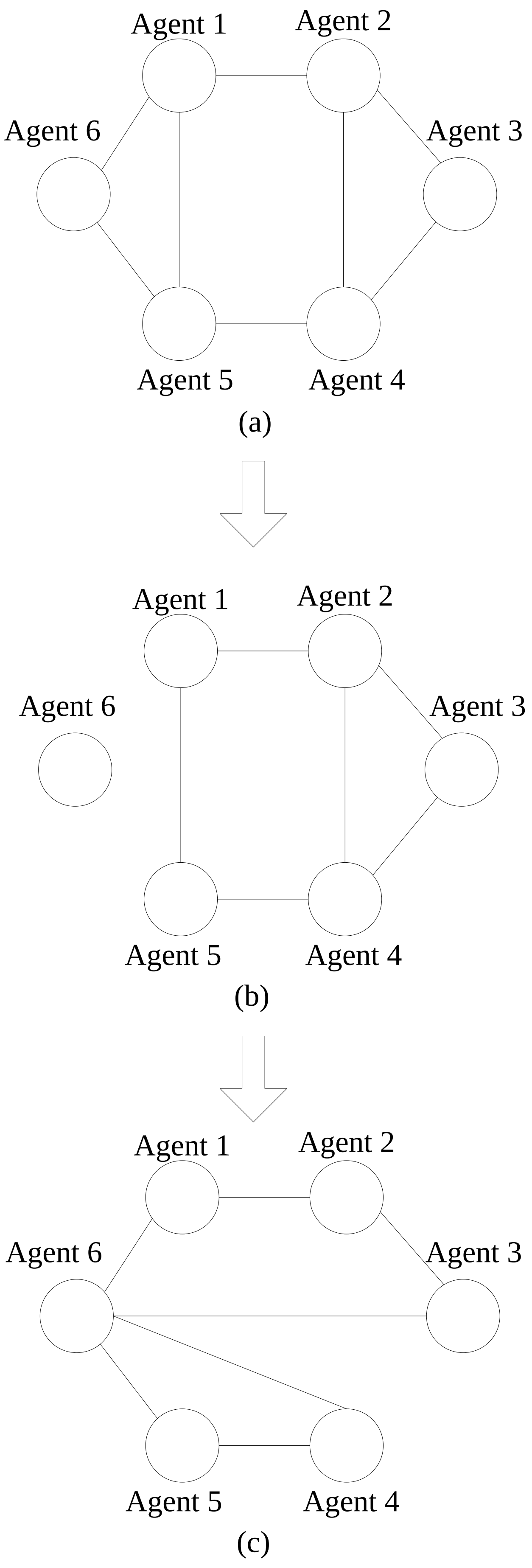


Figure 4.1: The switching communication graphs for the simulations.

of local training, each neural network (agent) trains the model parameter on its local sub-datasets simultaneously, where the stochastic variance reduced gradient is used to reduce the variance introduced by SGD and improve its convergence rate. Then, all neural networks share their model parameters among connected neighbors to update the models in the second phase of global consensus, where the heuristic adaptive consensus algorithm is used to adaptively adjust the weighted connectivity matrix to speed up the convergence rate. Combining local training and global consensus, all agents converge to the optimum with only a single consensus communication at every training step.

On the one hand, the sub-datasets of all agents are not required to be shared among neighbors or uploaded to a centralized node, which helps to preserve the sensitive and private data. On the other hand, this proposed distributed training framework is based on the decentralized communication topology with great flexibility and expansibility. This means that the convergence of the distributed neural networks is unaffected in the switching graphs and the situation that an agent quits or participates in this communication graph on the condition that it always remains a spanning tree [126]. The procedure for the distributed heuristic adaptive neural networks is detailed in Table 4.1.

Table 4.1: The procedure of distributed heuristic adaptive neural networks

```
Algorithm: Distributed adaptive training for neural networks
Inputs: The neural network architecture, the sub-datasets $(\mathcal{D}_1, \mathcal{D}_2, \cdots \mathcal{D}_k)$ on each agent, the number of agents $K$, and the corresponding connectivity matrix $D$ of the communication topology.
Outputs: The optimum model parameter $\theta^*$.
1: Initialize the model parameter $\theta_k$ for all agents in the communication topology.
2: Each agent $k$ optimizes the neural network using SVRG with its local sub-dataset ${}^k\mathcal{D}$ to get the updated $\theta_k'$.
3: Globally update all agents' model $\theta_k'$ using the proposed heuristic adaptive consensus algorithm to obtain $\theta_k''$.
4: Return to step 2 with globally updated $\theta_k''$.
5: Check the stop condition (such as a certain number of training steps).
6: Obtain the optimum model parameter $\theta^*$.
```

We take Figure 4.1(a) as an example of the communication topology for distributed training, where the entire dataset is evenly partitioned and stored on six initialized neural networks (agents) over the graph. Firstly, these agents are trained on their local sub-datasets and share their model parameters among directly connected neighbors under the communication graph shown in Figure 4.1(a). After some iterations, we assume that the connection between the agent 6 and the communication graph is cut off. In this case, the agent 6 can only train with its own local sub-dataset but not communicate with other agents, while the other five agents form a new communication topology as shown in Figure 4.1(b). At last, the agent 6 joins in the graph again and the six agents form another communication topology as shown in Figure 4.1(c). This changing process of the topologies in the simulation is to demonstrate that all agents still converge to the optimum in switching communication graph and the situation that any agent quits or participates in the communication graph on the condition that it always has a spanning tree.

In this chapter, four different algorithms are compared to validate the superiority of the distributed heuristic adaptive neural networks with variance reduction. These algorithms are tested on a large-scale dataset BlogFeedback [139] (downloaded from the University of California, Irvine (UCI) Machine Learning Repository) for the task of regression, which contains 52397 samples with 280 features. Besides, the neural networks on all agents have the same structure, with the sigmoid function being the activation function. All neural networks are randomly initialized in the uniform distribution $[-1, +1]$ with all inputs normalized in $[0, 1]$. The following four algorithms are conducted in the above-described switching graphs as shown in Figure 4.1, with RMSE applied to assess their performance.

- **Distributed heuristic adaptive training using batch gradient**. Six neural networks are trained locally using batch gradient methods and the heuristic adaptive consensus algorithm is used to globally update their model parameters over the graph.

- **Distributed training using batch gradient**. As before, batch gradient descent is used to locally optimize the neural networks, while consensus algorithm with a fixed weighted connectivity matrix is used to make all agents converge to each other.

- **Distributed heuristic adaptive training using SGD**. In this case, SGD is used to optimize the neural networks, with the heuristic adaptive consensus algorithm globally updating their model parameters.
- **Distributed heuristic adaptive training using SVRG**. Similar to before, the heuristic adaptive consensus algorithm is used for distributed training, with SVRG locally optimizing neural networks.

Figures 4.2 to 4.5 are the simulation results on the above four algorithms, where we can find that the agents of all the four figures can approach the optimal model while converging to each other after enough iterations, despite the great difference in their initial statuses. Meanwhile, in all the figures except for Figure 4.4, there is a clear gap between the agent 6 and the other agents from 21^{st} to 31^{st} iteration, when the agent 6 leaves the graph 4.1(b) and joins in again to form a new graph 4.1(c), respectively. More specifically, after the agent 6 leaves the communication graph at 21^{st} iteration, the other five agents in Figure 4.1(b) can still converge to the optimum in union, while the convergence speed of agent 6 slows down, comparing to other five agents. This means that the proposed distributed heuristic adaptive training method is more efficient and robust to the failure of an agent over the graph. Furthermore, after the agent 6 joins in and form another communication graph at 31^{st} iterations, all the six agents in Figure 4.1(c) approach each other again and converge to the optimum in union. This demonstrates that the distributed training method is suitable for switching graphs with a spanning tree.

We can find from Figure 4.2 that the heuristic adaptive consensus algorithm requires 20 and 30 iterations for all agents achieving consensus and converging to the optimal model, respectively. By comparison, as shown in Figure 4.3, it is required 10 and 50 iterations for consensus algorithm with the fixed weighted connectivity matrix to reach consensus and the optimum, respectively. Although more iterations are needed to achieve consensus when it is compared with the consensus algorithm, the heuristic adaptive consensus algorithm requires much fewer iterations to reach the optimal model, which means that this method is more efficient and suitable for distributed training neural networks.

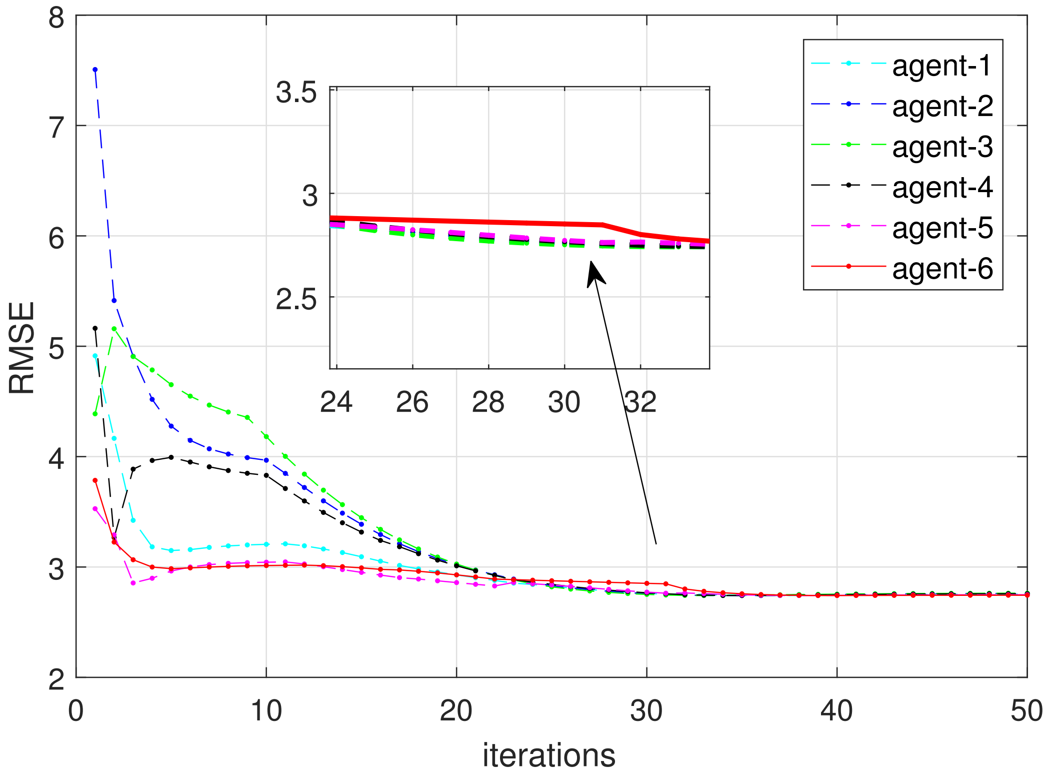


Figure 4.2: Distributed heuristic adaptive training using batch gradient.

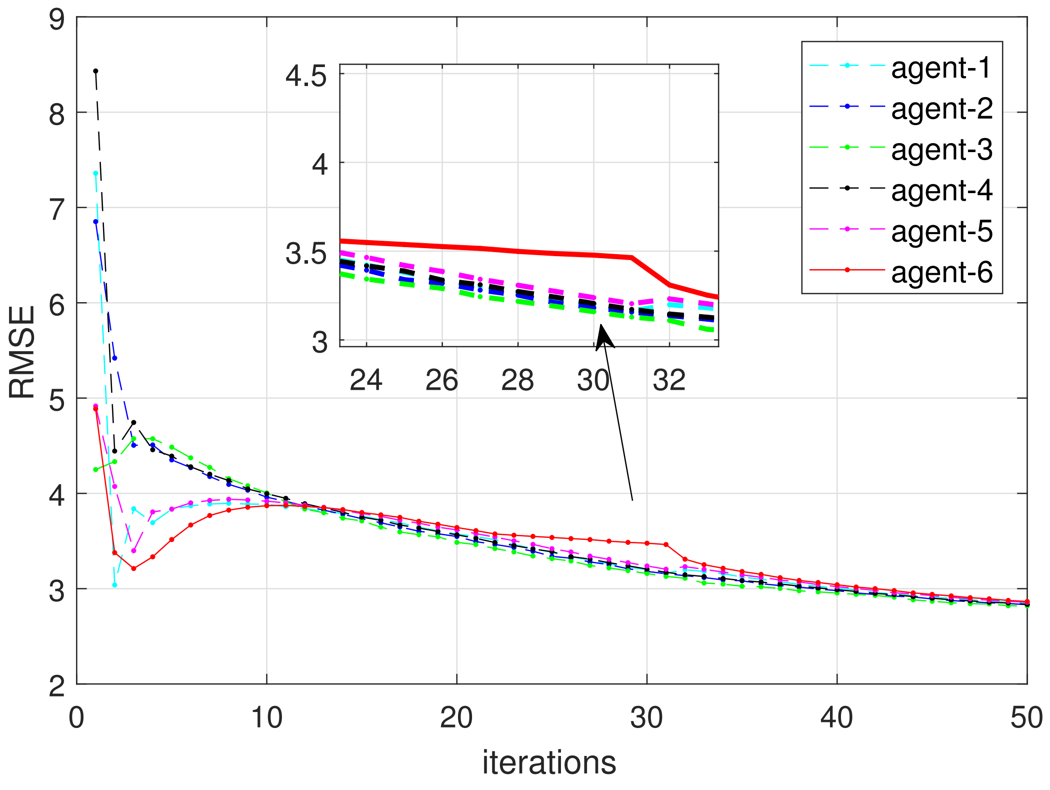


Figure 4.3: Distributed training using batch gradient.

Figure 4.4 is the performance of distributed training using SGD, where we can not find the gap shown in other figures, because the agents have not converged between 21^{st} to 31^{st} iterations. Besides, we can find that there are more fluctuations in Figure 4.4 than in Figure 4.2, which is harmful to the convergence. Therefore, a smaller learning rate should be taken for SGD to ensure convergence, which would further impair the convergence rate. As shown in Figure 4.5, the simulation result of distributed training with SVRG exhibits nearly the same performance as the batch gradient (Figure 4.2), which is more smoothly and shows fewer fluctuations, compared with SGD (Figure 4.4). This verifies the effectiveness of the SVRG method, which shows better performance with only a little extra computational cost. What deserves to be mentioned is that a larger learning rate can be taken for SVRG because of the reduction of the variance introduced by stochastic gradient, which would further improve their performance and convergence rate. Here, the same learning rate was used in the simulations for all of the four algorithms for the convenience of comparisons.

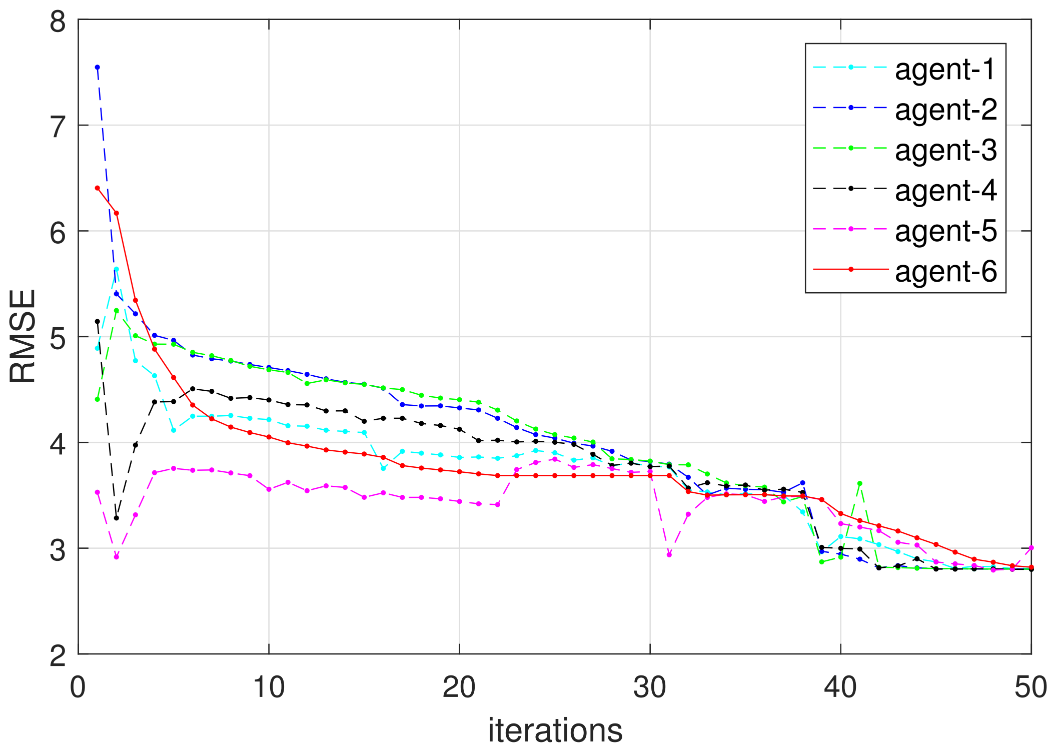


Figure 4.4: Distributed heuristic adaptive training using SGD.

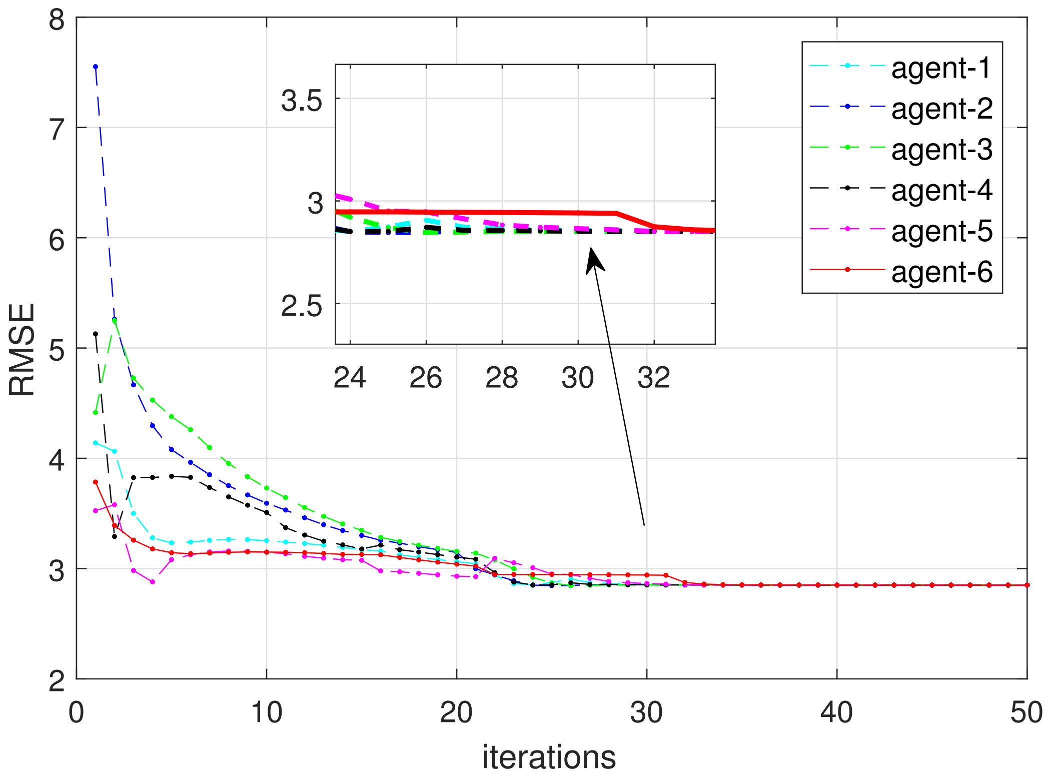


Figure 4.5: Distributed heuristic adaptive training using SVRG.

4.6 Conclusion

In recent years, distributed training has become a hot research topic for its excellent ability in machine learning problems on massive and privacy-related data. Besides, it is common nowadays that data samples are not available to be centralized or shared among agents because of computational limitations or privacy issues. However, it is still expected that the model is built based on the whole dataset rather than only learns from its local sub-dataset. For this problem, a distributed training method is designed based on the stochastic variance reduced gradient and the heuristic adaptive consensus algorithm for neural networks connected in switching communication graphs.

Through the theoretical analysis, we show that the SVRG reduces the variance introduced by SGD and improve its convergence rate with only a little extra computational cost. For distributed neural networks, it is proved that the consensus-based distributed training with sufficient consensus communication obtains the identical performance as the model built on the whole dataset. However, this method is computationally expensive and too straightforward as all agents are considered equally during the consensus process. Furthermore, the heuristic adaptive consensus algorithm is proposed to adaptively adjust the weighted connectivity matrix, where a better agent is more likely to have a larger influence on its neighbors. Convergence analysis demonstrates that all agents in switching graphs can still converge to the optimum with a single consensus communication at every training step, which saves plenty of communication costs. At last, simulations give the results that the heuristic adaptive consensus algorithm requires fewer iterations than consensus algorithm with the fixed weighted connectivity matrix for all agents to reach the optimum, and SVRG greatly decreases the fluctuation caused by SGD and improve its performance.

Chapter 5

Distributed Training Algorithm for Deep Neural Networks with Communication Compression

5.1 Introduction

Chapters 4 has promoted the distributed training algorithm (proposed in chapter 3) with the heuristic adaptive consensus algorithm and the stochastic variance reduced gradient to optimize the training process and reduce computational costs, which is effective and efficient. However, another issue for the distributed training algorithms is the expensive communication costs among connected agents, because all agents need to share their model parameters or gradients with others frequently during the distributed training process. This problem is especially important for deep neural networks, which usually has a large amount of model parameters. Thus, we are expected to reduce the communication costs in the distributed training process.

In this chapter, we explore to propose a distributed training framework for deep neural networks with communication compression, which not only reduces communication costs but also ensures the model accuracy. Inspired by the idea from consensus control that decentralized connected agents can reach consensus with only local communication, we design a consensus-based distributed training algorithm, which is proved to be able to make all agents converge to the optimum. Then, the error-compensated compression method is applied to reduce the communication burden during the consensus process of distributed training. Simulation study demonstrates that the distributed training with communication compression shows great performance on both IID and non-IID datasets, while significantly saving communication costs.

The remainder of this chapter is organized as follows. Section 5.2 proposes the problem of distributed training based on some practical challenges. Section 5.3 designs the distributed neural network with the convergence analysis, and proposes the distributed training framework with communication compression combined with the error-compensated compression strategy. Section 5.4 conducts the experimental simulation to verify the effectiveness of the proposed algorithm. Section 5.5 concludes this chapter.

5.2 Problem Formulation

A huge amount of data is generated and collected by different intelligent devices every day, and these data are often concerned with privacy and security related issues. This makes it not appropriate to be processed by a centralized server. Distributed training seems a promising solution for this problem, but the main issue of which is the massive communication costs among connected agents. Therefore, we expect to reduce the amount of communication during the distributed training process. Besides, the diversity of data should be also taken into account. The distributed stored data may be particularly different on the distribution and size, as the data from different devices depend heavily on their environmental usage scenarios.

For these problems, we plan to design a distributed training method, which should meet the following requirements:
1) It is suitable for decentralized communication topology to alleviate the communication burden on the center node and improve its extensibility;

2) There is no sharing of data samples among connected agents out of privacy issues;
3) Communication compression strategy is considered to reduce communication costs during the training process;
4) It is robust to the non-IID dataset for the diversity of different agents.

5.3 Distributed Neural Networks with Communication Compression

5.3.1 Distributed Neural Networks

Since that the consensus algorithm can drive all models in a decentralized model to their mean value, a simple idea for realizing distributed training is to average all models with enough consensus steps after each training iteration. It is easy to demonstrate that this distributed training method has almost the same performance as centralized training. However, the large number of consensus steps among connected agents leads to expensive communication costs, which is especially true for deep neural networks with millions of model parameters.

Even though a single consensus communication does not drive all agents to their average, but it still makes all agents approach each other. Given that the weighted connectivity matrix W satisfies $\sum_{j=1}^{K} w_{ij} = 1$ and $w_{ij} \in [0, 1)$, we have

$$\begin{aligned} \max_k \|\theta'_k\| &= \max_k \|\sum_{j=1}^{K} w_{kj}\theta_j\| \\ &\leq \max_k \|\theta_k\|. \end{aligned} \tag{5.1}$$

Similarly, we also have $\min \|\theta'_k\| \geq \min \|\theta_k\|$, and this means that a consensus step always reduces the upper bound and improve the lower bound of all the agents in the communication topology.

Inspired by the idea that gradient descent method optimizes the model parameter and the consensus algorithm draws all agents to approach each other, we proposed the consensus-based distributed training framework for neural networks, which requires consensus communication only once at every training step. This algorithm consists of two update phases, which are written as

$$\theta'_k = \theta_k - \eta \nabla E(\theta_k), \tag{5.2}$$

$$\boldsymbol{\theta}'' = W\boldsymbol{\theta}', \tag{5.3}$$

where θ'_k is the locally trained model parameter of $\theta_k \in \mathbb{R}^{1\times N}$, $\boldsymbol{\theta}' = [\theta'_1, \theta'_2 ... \theta'_K]^T \in \mathbb{R}^{K\times N}$ and $\boldsymbol{\theta}''$ are the parameter matrix of all agents before and after single consensus communication, with K being the number of agents.

Local training defined by (5.2) is the first phase, where all agents over the graph simultaneously train their model parameters based on locally stored sub-datasets using gradient descent method. The second phase is globally consensus update described by (5.3), where these locally trained agents share their model parameters with directly connected neighbors to globally update their model parameters using the consensus algorithm. By repeating these two procedures, all the agents converge to the optimal model, and their overall performance is similar to the model trained on the entire dataset.

On the one hand, each agent only needs consensus communication once after every training step, which avoids repetitive consensus communication and alleviate the communication costs. On the other hand, all agents are not required to share their local data or upload them to a centralized server, which benefits the privacy protection. Besides, this decentralized communication topology has great flexibility and expansibility, that is, the training process is unaffected in switching communication graphs, provided that the topology always remains a spanning tree [126].

The process of the distributed training algorithm for neural networks is detailed in Table 5.1.

Table 5.1: The process of distributed neural networks

Algorithm 1: Distributed training for neural networks

```
Inputs: The architecture of neural networks and sub-datasets ($\mathcal{D}_1, \mathcal{D}_2, \cdots \mathcal{D}_K$) stored on each agent, the communication topology with the number of agents $K$, and the weighted connectivity matrix $W$.
Outputs: The optimum neural network model parameter $\theta^*$.
1: Initialize the neural network model parameter $\theta_k$ of each agent over the communication topology.
2: Every agent optimizes the neural network using gradient descent with its stored sub-dataset $\mathcal{D}_k$ to obtain the locally optimized $\theta'_k$.
3: Update the parameter matrix $\boldsymbol{\theta}'$ for all agents with single consensus communication to obtain $\boldsymbol{\theta}''$.
4: Back to step 2 with globally updated $\boldsymbol{\theta}''$.
5: Check the stop criterion (such as a predefined number of training iterations).
6: Return the optimum neural network parameter $\theta^*$.
```

5.3.2 Convergence Analysis

For the distributed training problem, the update rule for all agents' model parameters using the Algorithm 1 is

$$\boldsymbol{\theta}^{t+1} = W(\boldsymbol{\theta}^t - \eta \nabla E(\boldsymbol{\theta}^t)), \tag{5.4}$$

where $\boldsymbol{\theta}^t = [\theta_1^t, \theta_2^t ... \theta_K^t]^T$ and $\nabla E(\boldsymbol{\theta}^t) = [\nabla E({\theta_1}^t), \nabla E({\theta_2}^t) ... \nabla E({\theta_K}^t)]^T$ are the concatenation of all agents' model parameters and gradients at the tth iteration, respectively.

Assumption 1: The gradients $\nabla E(\theta_k)$ of all agents' empirical risks are Lipschitz continuous with constant $L_e > 0$.

Assumption 2: The weighted connectivity matrix W satisfies $W = W^T$ and $\mathbf{1}^T W = \mathbf{1}^T$, where $\mathbf{1} = [1\ 1 ... 1]^T \in \mathbb{R}^{K\times 1}$.

Theorem 1: The overall empirical risk $\sum_{k=1}^{K} e(\theta_k)$ of all agents converge to the minimum using the proposed distributed training algorithm under Assumptions 1 and 2.

Proof: Inspired by [140], we design a function to represent the update process of the model parameters using the distributed training method, which is defined as

$$J(\boldsymbol{\theta}) = \frac{1}{2\eta}\|\boldsymbol{\theta}\|^2_{I-W} + F(\boldsymbol{\theta}), \tag{5.5}$$

where $\|\boldsymbol{\theta}\|_{I-W}^2 \triangleq \langle \boldsymbol{\theta}, (I-W)\boldsymbol{\theta}\rangle$ with I being $K \times K$ unit matrix, and $\nabla F(\boldsymbol{\theta}) = W\nabla E(\boldsymbol{\theta})$.

The gradient of $J(\boldsymbol{\theta})$ is

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{\eta}(I-W)\boldsymbol{\theta} + W\nabla E(\boldsymbol{\theta}). \tag{5.6}$$

Then, we reformulate (5.4) as

$$\begin{aligned}\boldsymbol{\theta}^{t+1} &= \boldsymbol{\theta}^t - \eta[\frac{1}{\eta}(I-W)\boldsymbol{\theta}^t + W\nabla E(\boldsymbol{\theta}^t)] \\ &= \boldsymbol{\theta}^t - \eta\nabla J(\boldsymbol{\theta}^t),\end{aligned} \tag{5.7}$$

where we find that (5.5) generates the same sequences of $\boldsymbol{\theta}$ as (5.4), using gradient descent with the same learning rate η.

Also, it is obtained from (5.7) that

$$\langle \nabla J(\boldsymbol{\theta}^t), \boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t\rangle = -\frac{\|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t\|^2}{\eta}. \tag{5.8}$$

Since that all $\nabla E(\theta_k)$ is L_e-Lipschitz, $|\nabla E(\theta_k^{t+1}) - \nabla E(\theta_k^t)| \leq L_e|\theta_k^{t+1} - \theta_k^t|$.

$$\begin{aligned}|\nabla F(\boldsymbol{\theta}^{t+1}) - \nabla F(\boldsymbol{\theta}^t)| &= |W(\nabla E(\boldsymbol{\theta}^{t+1}) - \nabla E(\boldsymbol{\theta}^t))| \\ &\leq L_e|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t|.\end{aligned} \tag{5.9}$$

Thus,

$$\begin{aligned}&|\nabla J(\boldsymbol{\theta}^{t+1}) - \nabla J(\boldsymbol{\theta}^t)| \\ &= |\frac{1}{\eta}(I-W)(\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t) + \nabla F(\boldsymbol{\theta}^{t+1}) - \nabla F(\boldsymbol{\theta}^t)| \\ &\leq (\frac{1}{\eta}(1-\lambda_{min}(W)) + L_e)|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t|.\end{aligned} \tag{5.10}$$

where $\lambda_{min}(W)$ is the minimum eigenvalue of the matrix W.

We define $L_l = \frac{1}{\eta}(1-\lambda_{min}(W)) + L_e$, then $\nabla J(\boldsymbol{\theta})$ is L_l-Lipschitz.

It is therefore

$$\begin{aligned}J(\boldsymbol{\theta}^{t+1}) - J(\boldsymbol{\theta}^t) \leq &\langle \nabla J(\boldsymbol{\theta}^t), \boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t\rangle \\ &+ \frac{L_l}{2}\|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t\|^2.\end{aligned} \tag{5.11}$$

Combining (5.8) and (5.11) we have

$$\begin{aligned}&J(\boldsymbol{\theta}^{t+1}) - J(\boldsymbol{\theta}^{t})\\&\leq -\frac{1}{2}(\frac{1}{\eta}(1+\lambda_{min}(W)) - L_e)\|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^{t}\|^2.\end{aligned} \tag{5.12}$$

Sum (5.12) from 0 to $+\infty$, we get

$$\begin{aligned}&J(\boldsymbol{\theta}^{t+1}) - J(\boldsymbol{\theta}^{0})\\&\leq -\frac{1}{2}(\frac{1}{\eta}(1+\lambda_{min}(W)) - L_e)\sum_{t=0}^{\infty}\|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^{t}\|^2.\end{aligned} \tag{5.13}$$

Set the learning rate η to satisfy $0 < \eta < \frac{1+\lambda_{min}(W)}{L_e}$, it is obtained that $J(\boldsymbol{\theta})$ is non-increasing and upper bounded by $J(\boldsymbol{\theta}^0) < +\infty$. therefore

$$\sum_{t=0}^{\infty}\|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^{t}\| < +\infty, \tag{5.14}$$

$$\|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^{t}\| \longrightarrow 0, t \longrightarrow \infty, \tag{5.15}$$

and we have $\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^{t} \longrightarrow \mathbf{0} \in \mathbb{R}^{K\times N}$, when $t \longrightarrow \infty$.

Combining with (5.4), when $t \longrightarrow \infty$ we have

$$W(\boldsymbol{\theta}^{t} + \eta\nabla E(\boldsymbol{\theta}^{t})) = \boldsymbol{\theta}^{t}. \tag{5.16}$$

Multiply both the right and left sides of (5.16) with $\mathbf{1}^T$, it is obtained

$$\mathbf{1}^T W(\boldsymbol{\theta}^{t} + \eta\nabla E(\boldsymbol{\theta}^{t})) = \mathbf{1}^T\boldsymbol{\theta}^{t}. \tag{5.17}$$

Therefore, we have $\sum_{k=1}^{K}\nabla E(\theta_k^t) = 0$ when $t \longrightarrow \infty$, which means that $E(\boldsymbol{\theta}) = \sum_{k=1}^{K} e(\theta_k)$ converges to the minimum. ■

5.3.3 Error-compensated Compression Strategy

During the consensus process, all agents need to share the model parameters with their neighbors, which brings heavy communication costs on the whole system, especially for deep neural networks with a huge number of model parameters. We, therefore, intend to compress the model parameter before each agent shares its model parameter with neighbors.

Bit-clipping is a simple and efficient compression method, which directly sets the lower p bits into zero for any decimal real number. For example, provided $p = 6$ we compress 1.2345678 into 1.2 with its lower 6 bits set to zero. This compression method saves a lot of storage memory but keeps the most valuable information on the model parameter. A straightforward idea is to share the model parameter after compressing the model parameter. In this way, only part model parameters are shared during the consensus process, which greatly reduces communication costs.

However, the compression method still loses some important information of the model parameter, which is harmful to the convergence. For this problem, the error-compensated compression method [141] is proposed, the main idea of which is to store the residue of the model parameter after the compression in the previous step and accumulate it to the current model parameter. Thus, the compression residue is not discarded but accumulated to update the model in the long run, which not only saves the communication resource but also benefits the convergence [142].

The update rule of the error-compensated compression strategy mainly consists of two steps. The first step is to compress the model parameter after combining it with the stored error, and this error is then updated in the second step.

$$\tilde{\theta} = \theta + \delta, \tag{5.18}$$

$$\delta = \tilde{\theta} - \mathcal{P}(\tilde{\theta}), \tag{5.19}$$

where θ and δ are the model parameter and the stored error, respectively. $\tilde{\theta}$ and $\mathcal{P}(\tilde{\theta})$ are the error-compensated model parameter before and after the compression process, with $\mathcal{P}$ being the compression strategy. The process of the error-compensated compress strategy is summarized in Table 5.2

Table 5.2: The procedure of the error-compensated compress strategy

```
Algorithm 2: Error-compensated compression strategy
Inputs: The model parameter $\theta$, the stored error $\delta$, the compression strategy $\mathcal{P}$.
Outputs: The compressed model parameter $\mathcal{P}(\tilde{\theta})$ and updated error $\delta$.
1: Compute the error-compensated model parameter $\tilde{\theta} = \theta + \delta$.
2. Compress the error-compensated model parameter to $\mathcal{P}(\tilde{\theta})$ using the compression strategy $\mathcal{P}$.
3: Update the error $\delta = \tilde{\theta} - \mathcal{P}(\tilde{\theta})$.
4. Return the compressed model parameter $\mathcal{P}(\tilde{\theta})$ and the updated error $\delta$.
```

5.3.4 Distributed Neural Networks with Error-compensated Communication Compression

Combining with the consensus algorithm and the error-compensated compression strategy, we propose the distributed training framework for neural networks with communication compression, which mainly consists of three steps. For each agent k, it maintains a local model parameter θ_k, a compressed error-compensated model parameter $\mathcal{P}(\tilde{\theta})$, and a stored error δ.

The first step is local training, where the model parameter θ_k is locally updated using gradient descent by (5.2). The second step is model compression, where the stored error is added to the locally updated model parameter in (5.18), which is then compressed by the compression strategy $\mathcal{P}$ with error updating by (5.19). The third step is global consensus, the compressed error-compensated model parameters are shared among connected neighbors, and the new consensus update process for each agent k is

$$\theta''_k = w_{kk}\theta'_k + \sum_{j\in\mathcal{N}(k)} w_{ij}\mathcal{P}(\theta'_j), \tag{5.20}$$

where $\mathcal{N}(k)$ represents the set of the neighbors of agent k, and it is notable that $k \notin \mathcal{N}(k)$.

The specific process of the distributed neural networks with error-compensated communication compression is summarized in Table 5.3.

Table 5.3: The process of distributed neural networks with error-compensated communication compression

Algorithm 3: Distributed training with communication compression

```
Inputs: The architecture of neural networks and sub-datasets ($\mathcal{D}_1, \mathcal{D}_2, \cdots \mathcal{D}_K$) stored on each agent, the compression strategy $\mathcal{P}$, the communication topology with the number of agents $K$, and the weighted connectivity matrix $W$.
Outputs: The optimum neural network model parameter $\theta^*$.
1: Initialize the neural network model parameter $\theta_k$ and the stored error $\delta_k$ of each agent over the communication topology.
2: Every agent optimizes the neural network using gradient descent with its stored sub-dataset $\mathcal{D}_k$ to obtain the locally optimized $\theta'_k$.
3: Compress the locally trained model parameter $\theta'_k$ with the stored error $\delta_k$ using Algorithm 2 to get $\mathcal{P}(\tilde{\theta})$ and the updated error $\delta_k$.
4: Receive $\mathcal{P}(\tilde{\theta}_j)$ from its neighbor $j \in \mathcal{N}(k)$ and globally update the parameter $\theta'_k$ over the graph using (5.20) to obtain $\theta''_k$.
5: Return to step 2 with updated error $\delta_k$ and globally optimized $\theta''_k$.
6: Check the stop criterion (such as a predefined number of training iterations).
7: Obtain the optimum neural network model parameter $\theta^*$.
```

5.4 Simulation and Discussion

5.4.1 Experimental Setup

We benchmark the distributed training framework with communication compression on the handwritten digit images dataset MNIST [143], which is a standard image classification task. This dataset consists of 55000 training samples and 10000 test samples, where each sample (image) is given one of the 10 labels representing the handwritten digit from 0 to 9. The convolutional neural network [144] is used for this image classification task, and we use a four-layer CNN in this simulation, which consists of two convolution layers and two fully connected layers with more than 3 million parameters.

The decentralized communication topology in Figure 5.1 is taken as an example, where ten sub-datasets are stored on these agents over the graph, and ten CNNs with the same structure are also initialized on each agent. In this distributed training framework, there is no exchange of data samples, and all agents are only allowed to share the information of model parameters among their directly connected neighbors.

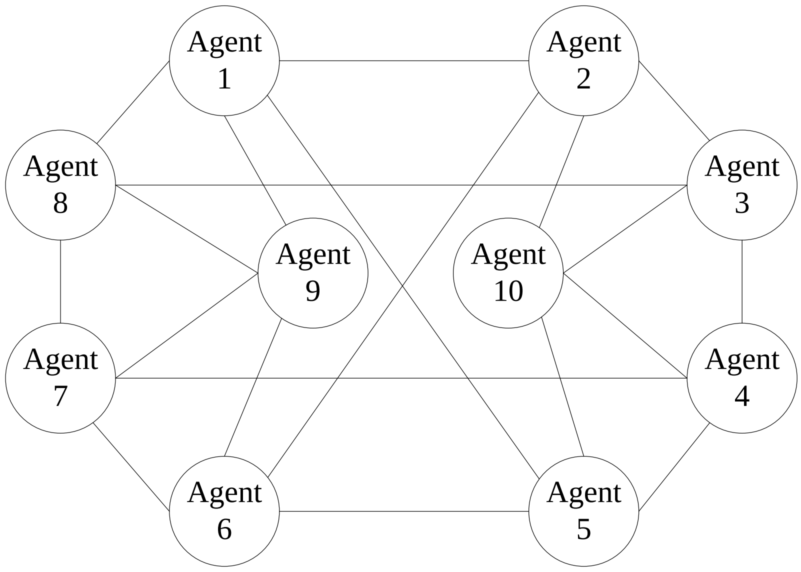


Figure 5.1: The decentralized communication topology.

On the one hand, we consider the ideal situation that the sub-datasets stored on different agents are IID data. For the balanced IID data case, the entire dataset is partitioned and distributed evenly stored on the ten agents over the communication graph. On the other hand, it is more reasonable in the practical scenario that the training sub-datasets stored on different agents depend heavily on their local environments and usage patterns, these sub-datasets are non-IID and highly divergent from the perspective of size and distribution. In the unbalanced non-IID data case, we consider the extreme situation that each agent has access to the data with only one label. The entire dataset is partitioned based on the label (from 0 to 9) and distributed on the ten agents over the graph. For both the IID and non-IID data cases, the following three algorithms are compared in this simulation.

- **Local training**: each agent trains its CNN model only based on the locally stored sub-dataset, and there is no communication among these agents.
- **Centralized training**: all agents are connected in a centralized topology, and the central server is used to average all local models after every local training step.

- **Distributed training (Algorithm 1)**: all agents are connected over the decentralized communication graph, each agent trains the model on its sub-dataset and share their full model parameters among directly connected neighbors, with the consensus algorithm globally updating their model parameters.
- **Distributed training with error-compensated communication compression (Algorithm 3)**: the consensus algorithm is used to globally update the model parameters after local training, with the error-compensated compressed model information shared among directly connected neighbors.
- **Distributed training with communication compression (Algorithm 4)**: the consensus algorithm is used to globally update the model parameters after local training, while the compressed model information using bit-clipping only (without error-compensation) are shared among directly connected neighbors.

5.4.2 Simulation Results

In general, the model parameter is stored and shared in the floating-point number form, which usually has 8 significant digits. Bit-clipping with $p = 6$ (compression rate is 1/4) is taken as the compression method for Algorithm 2 in this simulation, which saves 3/4 communication costs. For both IID and non-IID datasets, we evaluate the performance of all algorithms on the same test dataset, which contains 10000 samples with all of the ten labels.

Figures 5.2 and 5.3 are the simulation results of the above five algorithms on the IID and the non-IID datasets, respectively, where the averaged classification accuracy of all the ten agents is taken as the performance for each algorithm. We find from Figure 5.2 that the classification accuracy on the test dataset of all the five algorithms increase dramatically at the beginning training process and then achieve convergence. It is notable that the distributed training with communication compression (Algorithm 4) shows much worse performance than the other four algorithms. Especially, instead of improving the model accuracy, distributed training with bit-clipping only (without error-compensation) significantly decreases agents' performance. This means that the sharing of incomplete or inaccurate model information hurts the model accuracy to a large extent. Except for Algorithm 4, the local training exhibits the lowest model accuracy, although the classification accuracy of local training rises faster than distributed training and distributed training with error-compensated communication compression (Algorithm 3) at the beginning training stage. The centralized training, distributed training, and distributed training with error-compensated communication compression have nearly the same performance on the test dataset, which verifies the generalization capability of the proposed Algorithm 3. In particular, the distributed training with error-compensated communication compression algorithm saves 3/4 communication costs while the brought decrease in accuracy is little and negligible.

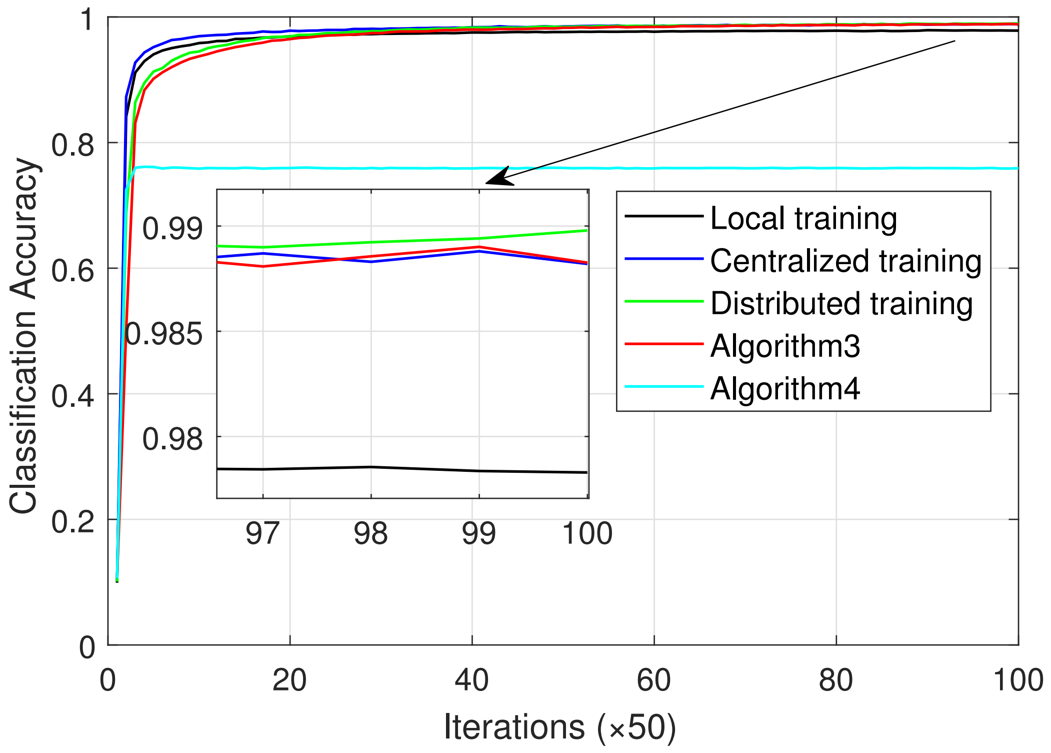


Figure 5.2: The performance of all algorithms on the IID dataset.

It is seen from Figure 5.3 that the classification accuracy of the local training algorithm on the non-IID dataset is always around 0.1 over the whole training process, this is because each agent learns the samples with only one label. Notably, Algorithm 4 on the non-IID dataset also cannot achieve convergence, and the model accuracies of which are around 0.1. This shows that distributed training on the non-IID dataset is more sensitive to the sharing of incomplete model information than on the IID dataset. Besides, centralized training and distributed training have comparable good accuracy

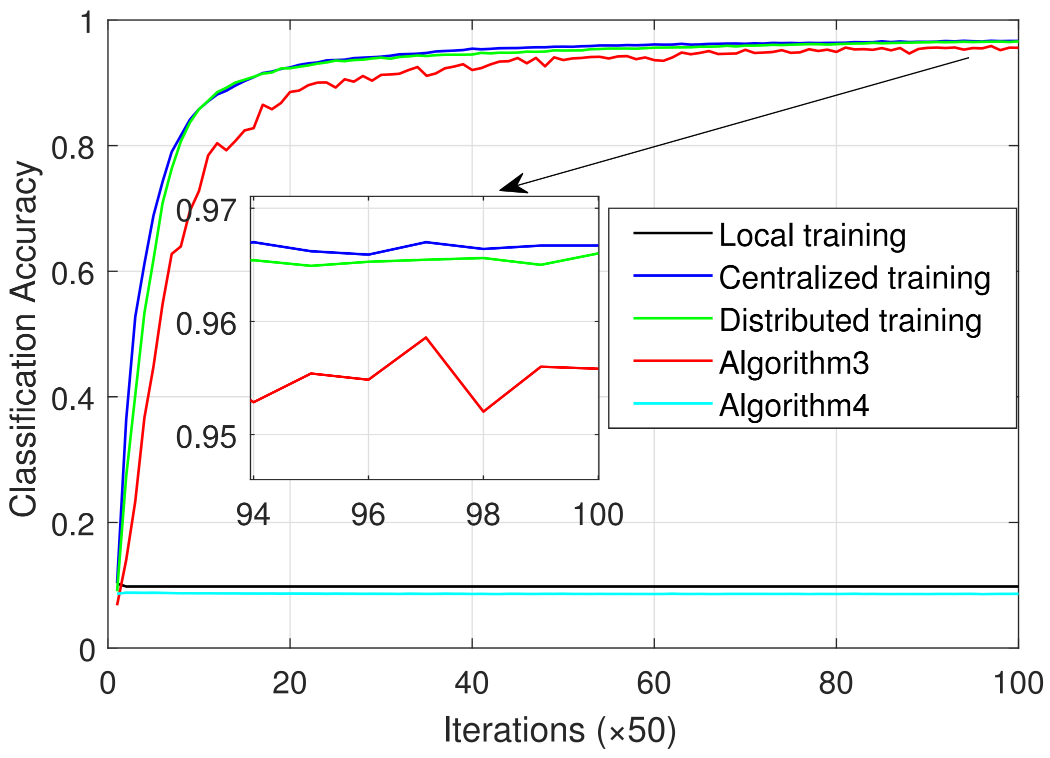


Figure 5.3: The performance of all algorithms on the non-IID dataset.

on the non-IID dataset, even though samples with only one label are available for each agent. This means that the consensus process effectively helps each agent learn from other agents' sub-datasets, as long as all agents are connected over a graph with a spanning tree. For Algorithm 3, the averaged accuracy of all agents shows only a little worse performance than that of distributed training. However, it has far better performance than the local training algorithm, which means that the distributed training algorithm with an error-compensated compression strategy is robust to the non-IID dataset.

To further analyze the influence of the compression rate of the compression strategy with and without error-compensation on the classification accuracy of distributed training, we test the distributed training with communication compression and error-compensated communication compression (Algorithm 3 and Algorithm 4) using bit-clipping with p from 0 to 7, on both IID and non-IID datasets. Correspondingly, the compression rate ranges from 0 to 1 at 1/8 intervals, with the compression rate of 0 and 1 representing the local training and the distributed training without communication compression (Algorithm 1), respectively.

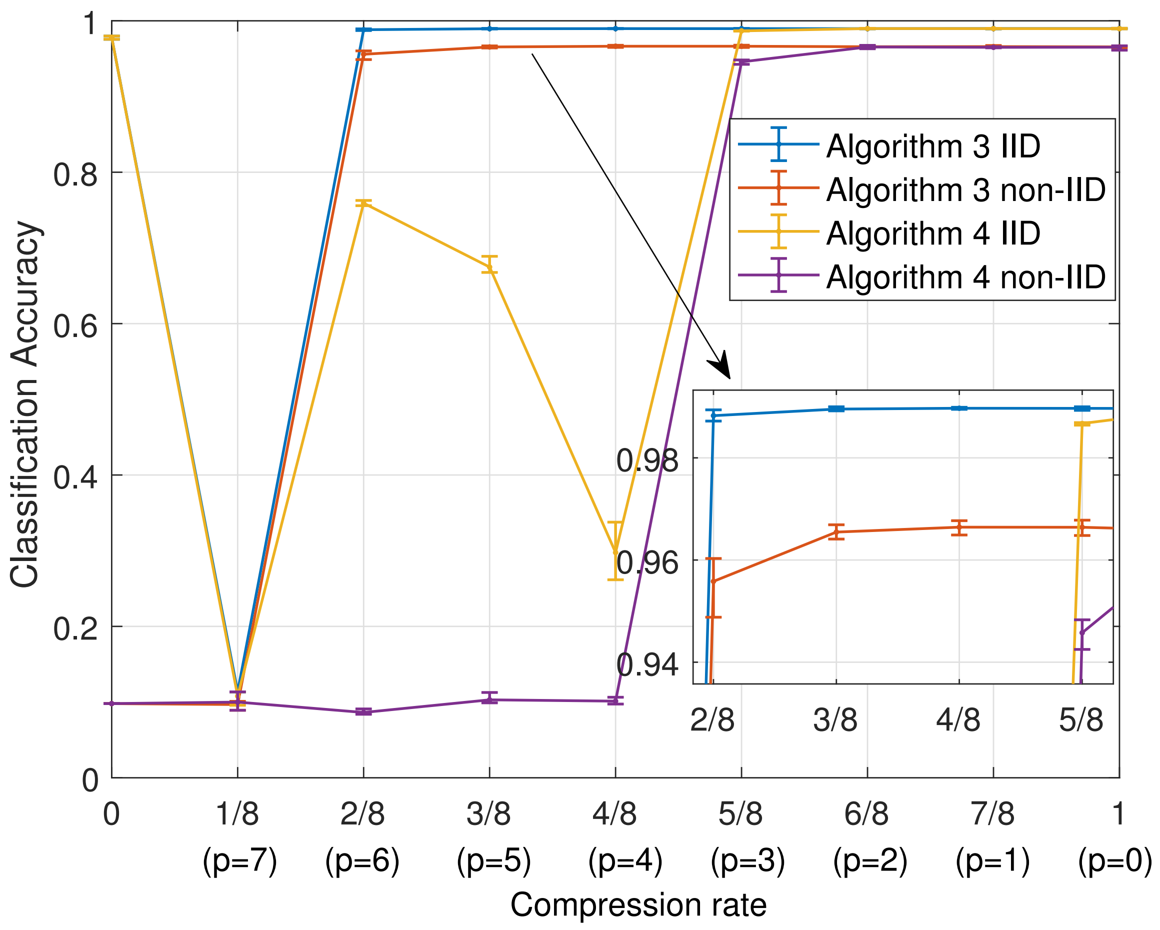


Figure 5.4: The influence of the compression rate on both IID and non-IID datasets.

Figure 5.4 describes the performance of Algorithms 3 and 4 on both IID and non-IID datasets, with the compression rate ranging from 0 to 1, where each point shows the averaged value, the lower bound, and the upper bound of the classification accuracies for the ten agents in the decentralized communication topology. The classification accuracies of these points that do not converge are around 0.1, which equals the reciprocal of the number of classification classes. As mentioned above, the local training method does not converge on the non-IID dataset, while it shows fair performance on the IID dataset. With 1/8 compression rate, neither Algorithm 3 nor Algorithm 4 achieve convergence on IID or non-IID datasets, which means that too aggressive compression harms the convergence of the distributed training algorithms with communication compression even though the error compensated strategy is applied.

It is easy to find from Figure 5.4 that Algorithm 3 shows far better performance than Algorithm 4 on both IID and non-IID datasets. More specifically, with the increase of compression rate, Algorithm 3 starts to show good performance on both IID and non-IID datasets when the compression rate is larger than 1/8, while Algorithm 4 cannot achieve convergence or falls in local optimum with poor accuracy until the compression rate reaches 5/8. When the compression rate reaches and exceeds 6/8, there is almost no difference between Algorithm 3 and 4, as the main model information is kept and only an unimportant part is clipped in these cases.

For the IID dataset, the performance of distributed training with a 2/8 communication compression rate is far better than that of the local training model (compression rate is 0), which demonstrates that the error-compensated communication compression strategy significantly improves the model accuracy with only 1/4 more communication costs. After that, the classification accuracy rises slightly at the 3/8 compression rate and then almost stays the same with the increase of the compression rate. Similarly, for the performance on the non-IID dataset, the classification accuracy of the distributed training with error-compensated communication compression first grows up (compression rate form 1/8 to 3/8) and then almost keeps unchanged with the increase of the compression rate. It is also notable that the gap between the upper and the lower bound of distributed training with communication compression on the IID dataset is much smaller than that on the non-IID dataset. This shows that the unbalance of the dataset is harmful to the convergence of the distributed training method.

From the overall view, the distributed training with error-compensated communication compression using bit-clipping with $p = 5$ (compression rate is 3/8) shows nearly the same performance as the centralized training and distributed training algorithm, but saves 5/8 communication costs. This verified that the proposed distributed training with error-compensated communication compression method is effective and efficient.

5.5 Conclusion

The combination of artificial intelligence, information technology, and IoT drives the development of distributed computing to offer more efficient and diversified services. This development considerably enriches our daily life but also brings new challenges to distributed machine learning, including the huge burden on the server for processing massive data, privacy protection, communication costs on distributed training, and dealing with non-IID data.

In this chapter, we propose a consensus-based distributed training framework with error-compensated communication compression to deal with both IID and non-IID datasets. The consensus-based distributed training algorithm is designed to free the center node, which not only benefits the privacy protection but also ensures the flexibility and expansibility of the communication system. It is proved in the convergence analysis that the distributed neural networks converge to the model with minimal empirical risk on the whole dataset by consensus communication on model parameters, even though there is no sharing of data among all agents. The error-compensated compression method with bit-clipping is then applied to compress the model parameter before sharing, which significantly reduces communication costs. In the simulation study, it is shown that distributed training with error-compensated communication compression is applicable on both IID and non-IID datasets, and has much better performance than the local training even with a small compression rate of 1/4. With a 3/8 compression rate, it shows almost the same classification accuracy as distributed training and centralized training but saves a lot of communication costs. Furthermore, both distributed training methods with and without error-compensated communication compression exhibit more differences among agents on the non-IID dataset than that on the IID dataset.

In future work, more efficient model compression methods can be considered to further reduce the communication costs while ensuring model accuracy in distributed training algorithms.

Chapter 6

Blockchain Empowered Distributed Adaptive Learning in Vehicular Networks

6.1 Introduction

Based on the distributed training algorithms proposed in the above chapters, this chapter is to further promote privacy protection and communication security in distributed training process combining with the blockchain technology, and apply the blockchain empowered distributed learning algorithm in vehicular networks for the classification of traffic lights.

Leveraging machine learning, a vehicle trains a model based on its collected data for decision making or predicting in different driving-related applications. However, the computing capability of a single vehicle is not enough to train an accurate and efficient model [145]. Furthermore, the limitation on sensor quantity and driving scenarios of a single vehicle also hinders the performance of the single-vehicle intelligence on complex driving environments.

To deal with this problem, a direct and effective solution is to take advantage of multi-vehicle intelligence [146], which is realized by the cooperation of many vehicles in a vehicular network. Cloud-based training [147] is widely adopted by manufacturers and service providers, where a cloud server with powerful computing ability and huge storage is used to gather the sensing data from vehicles to train a model, which is subsequently pushed to each vehicle. Besides the heavy communication burden on transmitting the huge raw data, the potential possibility of privacy leakage and security issues are non-ignorable and noteworthy. In recent years, the blockchain is regarded as a promising technology to build the trust mechanism and address privacy security problems [148], especially for those systems with distributed communication protocol, for example, the Internet of Things [149] and vehicular networks [150].

Federated learning requires a powerful central server to coordinate the whole distributed training process, which poses a heavy communication burden on the center, especially for the systems with high latency and low bandwidth. As contrast, the decentralized communication topology is more flexible and expansible, and is also robust to the failure or disconnection of any participants [88], which is thus more suitable for vehicular networks.

In this chapter, we aim to design a distributed learning framework combined with the blockchain technology to realize the multi-vehicle intelligence in vehicular networks, which also ensures the data privacy security and defends against malicious participants in the distributed learning process. The main contributions are summarized as follows:
1) A distributed learning framework combined with the blockchain is proposed to allow all vehicles in a vehicular network to train a combined model, while keeping their private data locally stored and ensuring the security of the communication process;
2) Based on the model accuracy and the computing power of each vehicle, an adaptive consensus strategy and the proof of contribution protocol are designed to encourage vehicles to make more contributions to the distributed learning process.

6.2 System Model and Distributed Adaptive Learning

6.2.1 System Model

We consider the situation that a number of vehicles with sensing and computing ability are connected in a vehicular network to collaboratively train a combined model for a given task, without the sharing of raw data. To address the problems of privacy protection and malicious vehicles, a secure information sharing and communication mechanism needs to be built, which not only ensures the security of the data-sharing process but also guarantee the data-sharing events traceable and unchangeable. The designed vehicular network for collaborative learning is shown in Figure 6.1, which consists of a blockchain module and a distributed learning architecture.

The distributed learning architecture makes it possible for all vehicles in the communication topology to train a combined model without revealing their local data, while the blockchain module builds secure communication connections among all the vehicles by encrypting the sharing information and recording all data-sharing events for further audit. More specifically, every data-sharing event is recorded as a transaction in the blockchain, which needs to be verified before writing to a block. Because of its limited storage, we only use blockchain to record the data-sharing events and the corresponding retrieve information other than recording the raw shared model information.

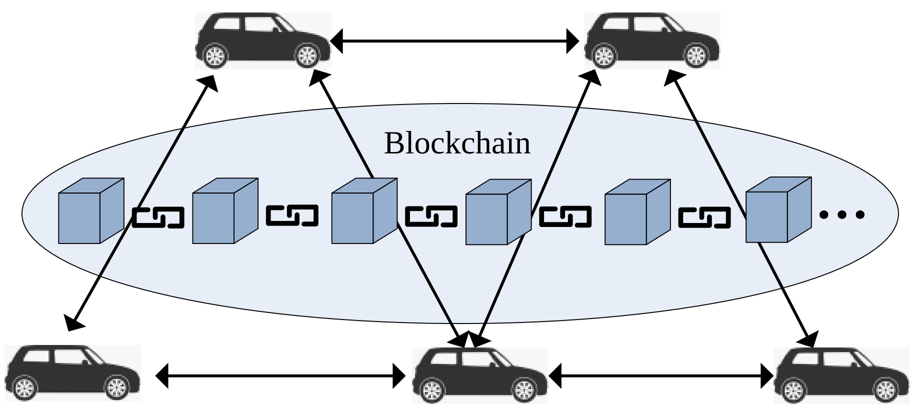


Figure 6.1: The blockchain empowered vehicular network.

6.2.2 Distributed Learning

The limited sensing and computing capacity of an independent vehicle makes it insufficient to deal with the complex driving environment and the increasingly diverse driving-related demands. A widely adopted approach is centralized modeling, where a cloud server is used to process and analyze the data collected and uploaded by the vehicles. After centrally trained the model on the server, vehicles download the global model from the cloud server, which is then used for predicting or decision-making combing with the real-time collected data. In this centralized strategy, the vehicles are only responsible for collecting data and uploading it to the cloud server, while the modeling and training process is conducted on the cloud server with powerful computing capacity. However, this strategy brings some potential problems, including privacy leakage, huge data transmission and storage.

A good solution is the popular federated learning, where all vehicles are involved in the sensing and modeling process, and the central server is only responsible for gathering and aggregating the model or gradient information from the vehicles. However, the main shortcoming of federated learning is the requirement of a center to coordinate the combined modeling process. This not only decreases the expansibility and the robustness of the whole system but also poses a heavy communication burden on the central server when a large number of vehicles are involved.

As mentioned in Chapter 3, the proposed consensus-based distributed learning algorithm has a two-phase updating procedure, where gradient descent is first used to train vehicles' local models based on their local datasets, and the consensus algorithm is then applied to drive all models to approach each other. The specific process is described as follows:

$$\theta'_k = \theta_k - \eta \nabla E(\theta_k), \tag{6.1}$$

$$\boldsymbol{\theta}'' = W\boldsymbol{\theta}', \tag{6.2}$$

where θ'_k is the locally trained model $\theta_k \in \mathbb{R}^{1\times N}$ of vehicle k, with η being the learning rate, $\boldsymbol{\theta}''$ is the updated models of $\boldsymbol{\theta}' = [\theta'_1, \theta'_2 ... \theta'_K]^T \in \mathbb{R}^{K\times N}$ using the consensus algorithm.

6.2.3 Adaptive Consensus Algorithm

In the above distributed learning algorithm, the weighted connectivity matrix W is kept fixed during the whole training process, which only depends on the consensus method and the communication topology of vehicles. This means that the influence of any vehicle on its neighbors is invariable and each vehicle approaches equally to its neighbors in the consensus process, regardless of the performance and computing contribution of each vehicle.

In practical applications, different vehicles in the distributed learning system commonly have different computing power, model accuracy, dataset size, batch size, etc. If all vehicles are always taken equally in the distributed training process, it will make vehicles unwilling to contribute more computing resources and better models to the distributed training system. Thus, the vehicle with better performance and more computing contribution should have a larger influence on its neighbors, which urges all vehicles to make more contributions to the whole training process. The classification accuracy and the root mean squared error are generally used to evaluate the performance of each vehicle for the classification task and the regression task, respectively. Here, we use the same verification dataset for each vehicle to evaluate their model.

Taking the classification task as an example, the vehicle with higher model accuracy should have more influence on its neighbors, and corresponding a larger value in the weighted connectivity matrix W. It is notable that there is a base accuracy (equals to $1/L$, L is the number of classes) for the classification task no matter how good or bad a model is, and this part of accuracy should not be taken into consideration for evaluating the performance of each vehicle. Besides, we define the computing contribution r_i of vehicle i as

$$r_i = \gamma \frac{b_i n_i}{\max(b_i)\max(n_i)}, \tag{6.3}$$

where γ is a user-defined parameter to balance the influence of model accuracy and computing contribution, b_i and n_i are the batch size and the dataset size, respectively, with $\max(b_i)$ and $\max(n_i)$ being their corresponding maximum value, respectively.

The weighted connectivity matrix W in terms of model accuracy and computing contribution is then defined as

$$w_{ij} = \frac{a_{ij}(c_j - \lambda)r_i}{\sum_{j=1}^{K} a_{ij}(c_j - \lambda)}, \tag{6.4}$$

where a_{ij} represents the connection status between vehicles i and j, c_j denotes the classification accuracy of vehicle j, with λ being the base accuracy that needs to be omitted. Besides, as the model accuracy commonly rises with the training process moving on, it is reasonable that the λ should rise with the increase of iterations, which should also be bounded. We then define

$$\lambda = \begin{cases} 1/L + \alpha * t & \text{if } \lambda < \beta \\ \beta & \text{if } \lambda \geq \beta \end{cases}, \tag{6.5}$$

where t is the number of iterations, α and β are user-defined parameters to adjust the λ.

Compared with the normal consensus algorithm with a constant W, the proposed adaptive consensus algorithm is more efficient and robust. On the one hand, the model accuracy is used to adaptively adjust the weighted connectivity matrix, which means that better vehicle commonly has a larger influence on its neighbors. On the other hand, the computing contributed is considered, where b_i represents the consumed computing resource at the current iteration, while n_i reflects the size and diversity of the data samples. With the consideration of model accuracy and computing contribution, the proposed adaptive consensus algorithm is used for the consensus process in distributed learning.

6.3 Blockchain Empowered Distributed Learning Framework

The distributed learning framework avoids the direct sharing of raw data by exchanging the model information among connected vehicles. However, the sharing of models still have the risk of data leakage, as the raw data may be extrapolated from the model or gradient information [49]. Besides, in this framework, we originally assume that the vehicles in the communication topology are all honest, but there may be some malicious vehicles in the vehicular network, such as the vehicles from the rivals or hackers. They may share inaccurate or incorrect information with their neighbors to damage the collaboratively learning process out of different motivations, and the shared model information from honest vehicles may be intercepted and then used for deducing the original data for unknown intention. To avoid these bad situations, a trust mechanism need to be built to protect privacy security and prevent the attack from malicious vehicles. Thus, the blockchain technology is considered.

6.3.1 Blockchain

Blockchain is commonly regarded as a decentralized ledger among all participants, which is originally proposed to build the trust mechanism in digital currency trading [151]. In blockchain networks, no center is required and all participants have the same status as they all have the entire copy of the blockchain, which contains all the transaction information. Therefore, it is extremely hard or even impossible to forge or tamper the information written in the block, which greatly benefits the privacy security.

Hash algorithm [152] and asymmetric encryption [153] are two key techniques in blockchain to ensure the security of communication and the integrity of the ledger. The hash algorithm transforms an input with any length or size to output with fixed length and this operation is irreversible. Because of the above two characteristics, the hash algorithm is widely used in file verification and digital signature. The asymmetric encryption algorithm includes a private key and a public key, where the file that is encrypted by the private key can be only decrypted by the corresponding public key, and vice versa. Compared with the traditional encryption method, the asymmetric encryption algorithm avoids the sharing of keys that are used for both encryption and decryption, which greatly improves communication security and has been applied in authentication and secure communication.

The three blockchain-related operations used in this chapter are as follows:

$$\hat{M}_i = \mathcal{H}(M_i), \tag{6.6}$$

$$\tilde{M}_i = \mathcal{E}(M_i, p_i), \tag{6.7}$$

$$M_i = \mathcal{Y}(\tilde{M}_i, q_i), \tag{6.8}$$

where M_i is the local model of vehicle i, $\hat{M}_i$ is the hash value of M_i after the hash operation $\mathcal{H}$. (p_i, q_i) is the pair of private and public keys of vehicle i, $\tilde{M}_i$ is the encrypted model of M_i, with $\mathcal{E}$ and $\mathcal{Y}$ being the encryption and decryption operations, respectively.

6.3.2 Blockchain Empowered Vehicular Network

The connected vehicles are the main body of the blockchain empowered vehicular network, which is responsible for the data collection, model training, as well as the updating of the blockchain.

For a specific collaborative learning task, a vehicle broadcasts its request to all the vehicles in the vehicular network, and write the first block of the blockchain, which includes a verification dataset and all initialization information, such as the model structure, hyper-parameters, etc. Vehicles that are willing to join in the collaborative learning task first register in the blockchain, and generate a pair of public key and private key, which is used to encrypt and decrypt the shared information, respectively. Meanwhile, each vehicle shares its public key with its connected neighbors. During the consensus communication, each vehicle first encrypts its local model using its neighbors' public keys, and then send the encrypted model information to its corresponding neighbors. After receiving the encrypted model information from its neighbors, the vehicle is able to decrypt the encrypted information using its private key to obtain the local models from its neighbors, which are then used for the consensus process.

Meanwhile, every model sharing event is recorded as a transaction and stored in the blockchain, the detail of which is shown in Figure 6.2. Even though the sharing information of a local model is much smaller than the raw data, it is still intensive to put all local models on the blockchain, which generally has limited storage space. Instead, we record the hash value of the local model in the blockchain.

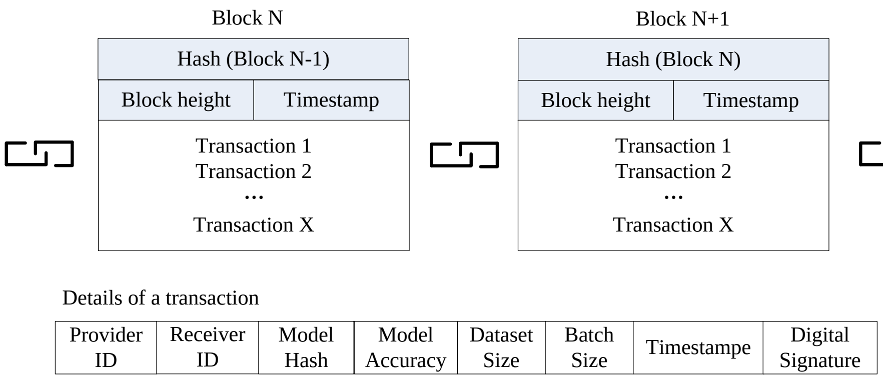


Figure 6.2: Records of the blockchain.

6.3.3 Blockchain Empowered Distributed Adaptive Learning Framework

Combing the blockchain technology and the proposed adaptive consensus algorithm, we design the blockchain empowered distributed adaptive learning (BDAL) algorithm. After local training, each vehicle computes the hash value of its local model and encrypts the local model, the encrypted model is then transmitted to its neighbors with the sharing event transaction broadcast to the blockchain. The schematic diagram of the communication process between any two connected vehicles is shown in Figure 6.3, The procedure of the proposed BDAL algorithm is detailed as follows:

Step 1) Initialization: a vehicle broadcasts the task in the vehicular network and builds the first block of the blockchain, other vehicles voluntarily register in the blockchain and generate their own public and private keys $(p_i, q_i), i \in [1, 2, ..., K]$. After joining in this blockchain, each vehicle i retrieves the initialization information from the first block to initialize its local model, and sends its public key to its connected neighbors $j \in \mathcal{N}(i)$.

Step 2) Local learning: each vehicle collect data from its environment and trains the local model based on its private data samples using gradient descent method, as shown

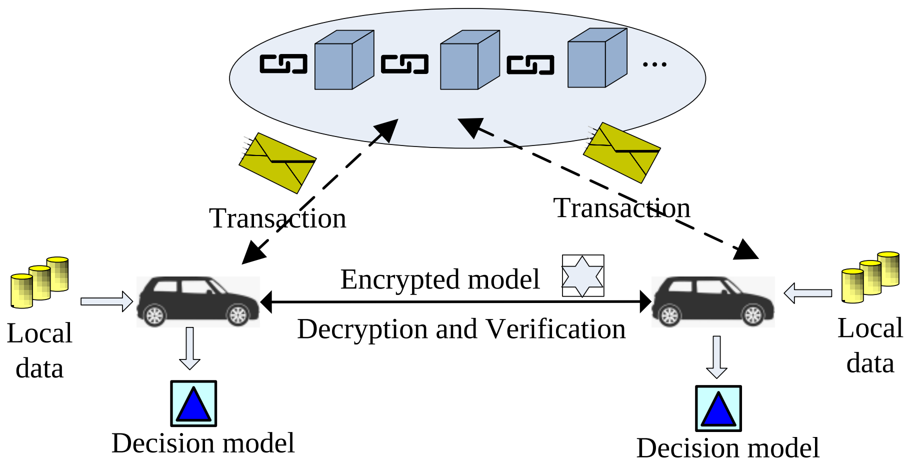


Figure 6.3: The communication process between any two connected vehicles.

in (6.1).

Step 3) Model sharing with encrypting: each vehicle i shares its local model M_i with its directly connected neighbors $\mathcal{N}(i)$ after encrypting the local model using (6.7) with corresponding public keys of their neighbors $q_j, j \in \mathcal{N}(i)$. Meanwhile, the vehicle i computes the hash value $\hat{M}_i$ of its local model M_i using (6.6) and broadcasts these model sharing event transactions (as detailed in Figure 6.2) in the vehicular network.

Step 4) Decryption and verification: after receiving the encrypted models from its neighbors $\mathcal{N}(i)$, each vehicle i decrypts these models using (6.8) with its private key p_i. Besides, the vehicle computes the hash value of the decrypted model and compares it with the model hash value recorded in the corresponding transaction for verification. In practical application, we conduct the verification operation with an user-defined small possibility to save computing resources.

Step 5) Global consensus: all vehicles globally update their model with these decrypted models from their neighbors using the adaptive consensus algorithm, combining (6.2) and (6.4).

Step 6) New block generation: all vehicles compete for the opportunity to write the new block to the blockchain using the designed Proof of Contribution (PoC) protocol. The vehicle that wins the competition broadcasts its new generated block to other vehicles for verification. After passed the verification, the new block is added to the blockchain.

The step 2) - step 6) repeat until a good enough model is obtained or a pre-defined number of iterations is reached.

6.3.4 Proof of Contribution

The traditional Proof of Work (PoW) protocol [154] brings heavy computing burden and time consumption, which is unsuitable for vehicle terminal with relatively low computing power. To address this problem, we propose the protocol of proof of contribution (PoC) to elect the vehicle to write a new block, which consists of the contribution of model accuracy and computing resources. Similar to 6.4, we define the proof of contribution as $O_i = (c_i - \lambda)r_i$, where the vehicle with the highest value of O_i has the opportunity to write the new block. This also encourages all vehicles to contribute more computing power and offer better models to the vehicular network.

6.3.5 Security Analysis

The security of the proposed blockchain empowered distributed adaptive learning algorithm is multi-fold. First, the proposed learning framework avoids sharing the raw data, and encrypts the shared model information using the public key of a vehicle's directly connected neighbor. This means that only the designated recipient vehicle is able to decrypt the shared model information and use it for model updating, which avoids the data abuse and leakage. Second, the adaptive consensus algorithm adjusts the weighted connectivity matrix based on their model accuracy and computing contribution, this alleviates the influence of malicious vehicles on the whole learning process, as the malicious vehicle with bad model accuracy also has a corresponding low value in the weighted connectivity matrix. Third, the verification process ensures all vehicles to offer models with real accuracies, all model sharing events are recorded as transactions in the blockchain, which are traceable and tamper-proof.

6.4 Simulation and Discussion

We test the proposed algorithm on a traffic signal dataset, which consists of eight different traffic signals as shown in Figure 6.4, and there are 500 samples in each class. Taking the vehicle communication topology shown in Figure 6.5 as an example, where eight vehicles are connected in the decentralized vehicular network. We first divide the dataset into a training dataset, a verification dataset, and a testing dataset according to the proportion of 7:1:2. Then, the training dataset is divided evenly into eight sub-datasets and assigned to each vehicle, while all vehicles share the same verification and testing datasets. To differentiate the computing contribution, the batch size is designed to be $B = [8, 8, 16, 16, 32, 32, 64, 64]$ for each vehicle, respectively, where the batch size b_i equals $B[i]$.

A five-layer convolutional neural network is designed for this classification task, and the Adam gradient descent [155] with 10^{-4} learning rate is taken as the optimization method. The first convolution layer is 32 filters of the size 5×5 kernel with 1 stride, while the second convolution layer is 64 filters of the size 3×3 kernel with 1 stride, and a max-pooling layer is connected behind each convolution layer. There are 256 and 8 nodes for the fully connected layer and the output layer, respectively. The rectified linear unit (ReLU) is taken as the activation function for both convolution and fully connected layers, with softmax being the activation function of the output layer. The following four algorithms are compared in this simulation, which includes training with and without a malicious vehicle. It is notable that the distributed training framework is not sensible to the above hyper-parameters, even though these hyper-parameters are carefully tuned for the local learning case for the specific classification task in this chapter. This means that we do not need to additionally tune hyper-parameters for the distributed training process.

- **Federated learning**: each vehicle computes the local gradient based on its private dataset and uploads it to the central server, while the server aggregates these local gradients to update the global model, which is then pushed to all vehicles.

- **Distributed learning**: all vehicles train the local models on their private datasets and communicate with their connected neighbors in the decentralized communication topology, while the consensus algorithm is used to globally update their models.

- **Blockchain empowered distributed adaptive learning (BDAL)**: All vehicles are connected over a decentralized topology, and blockchain technology is used to protect the data-sharing communication in the distributed learning process, with the adaptive consensus algorithm globally updating their models.

- **Local learning**: each vehicle trains its model only based on its local dataset, no communication happens among these vehicles.

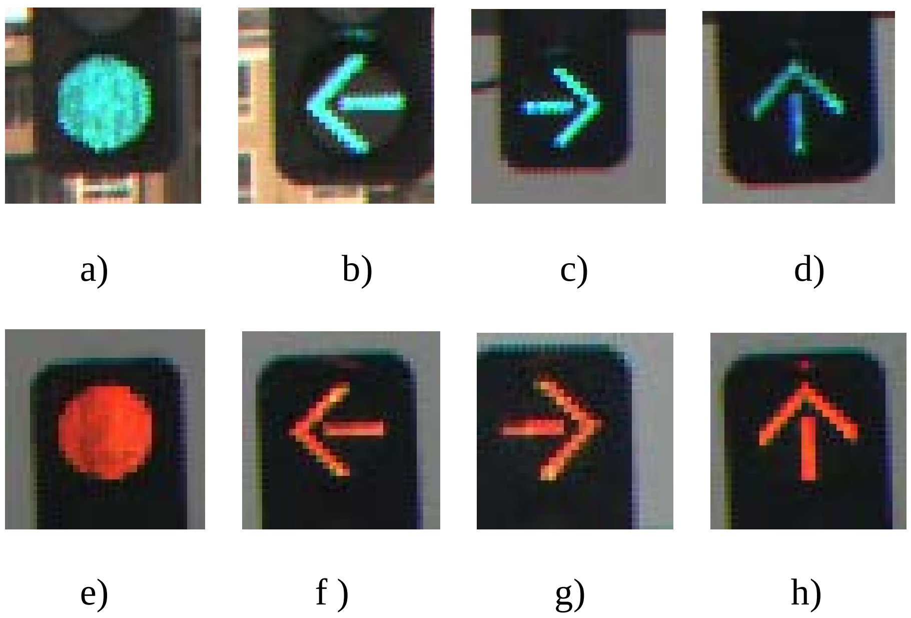


Figure 6.4: Different traffic signals in the dataset.

First, we consider the ideal situation that all vehicles are honest in the vehicular network. Figures 6.6 and 6.7 compare the above four algorithms on classification accuracy on the same testing dataset and the training loss, respectively, where the averaged value of all vehicles is used to represent the performance of each algorithm. It is seen from Figure 6.6 that all algorithms except local learning achieve good performance on the traffic signal classification task, and both distributed learning and the proposed BDAL shows a little better performance than the federated learning algorithm, while local

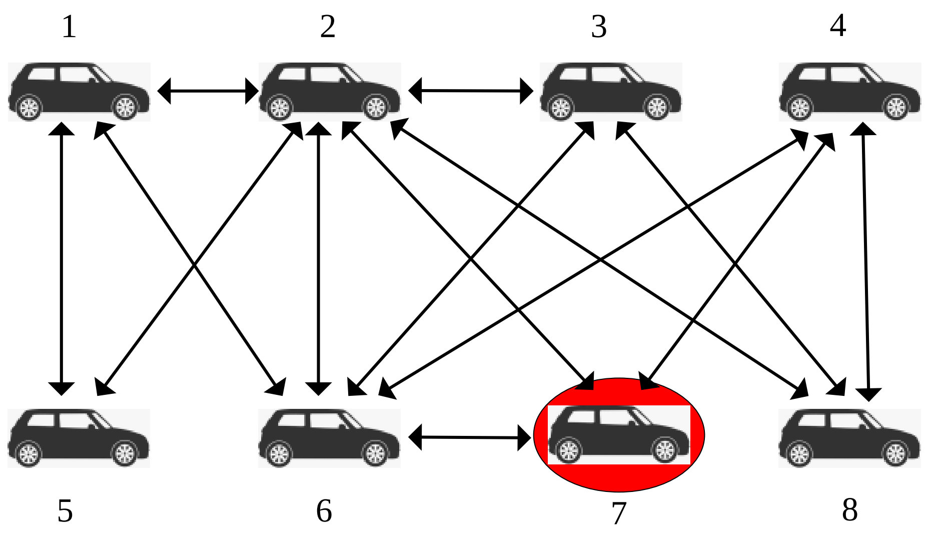


Figure 6.5: The communication topology of the vehicular network.

learning exhibits the worst accuracy. The generalization capability of the proposed BDAL algorithm is verified by its great performance on the testing dataset. Besides, the gaps between the upper and lower bounds of all vehicles' accuracies for local learning are quite large, which means that there are considerable differences among different vehicles. This shows that the local learning algorithm depends heavily on the initialization and local dataset, while the other three algorithms are all with little differences among different vehicles, and thus the upper and lower bounds of them are omitted in the figure. Similar results are also found from Figure 6.7, where the loss of federated learning, distributed learning, and BDAL all converge to zero after a number of iterations, while the loss of local learning does not reach the convergence.

To test the immunity to the malicious vehicle of the BDAL algorithm, we suppose that there is a malicious vehicle (with red background in Figure 6.5) in the vehicular network, which always shares a bad model with low accuracy. It is found from Figure 6.8 that federated learning and distributed learning are both vulnerable to the malicious vehicle, and they show as bad accuracy as the malicious vehicle. As a comparison, the BDAL algorithm still exhibits good accuracy on the classification task, which means that the proposed BDAL algorithm is robust to the malicious vehicle in the vehicular network. Similar results are also seen from Figure 6.9, where the loss of BDAL decreases to the minimum with the increase of iterations, while the loss of federated

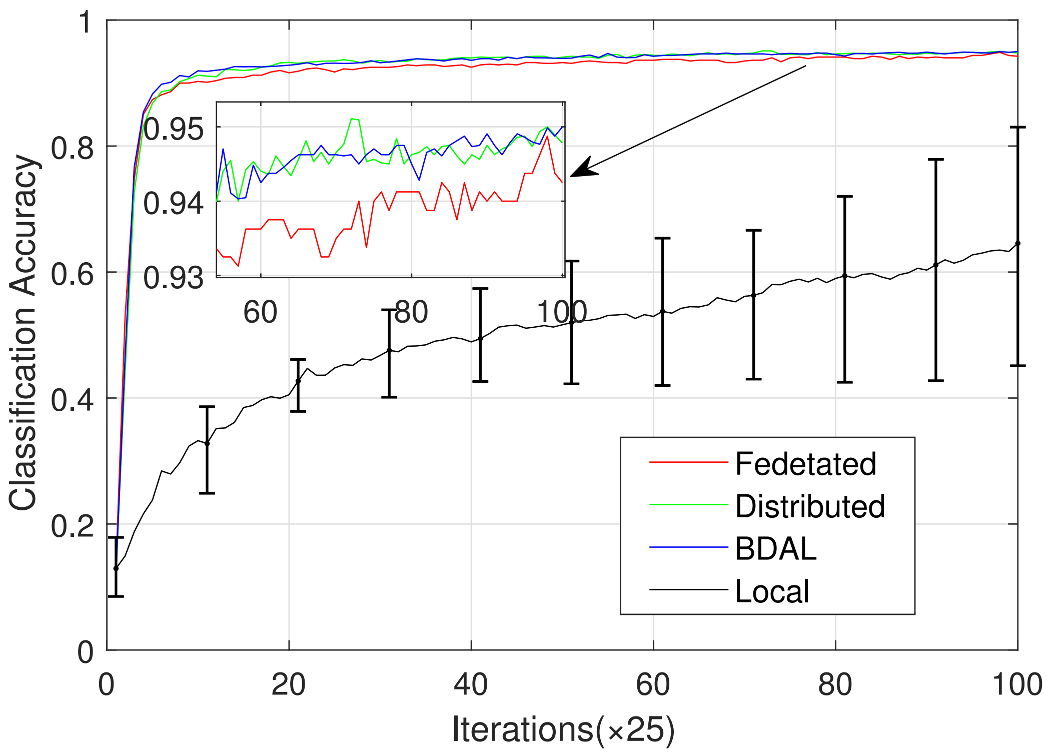


Figure 6.6: The classification accuracy on testing dataset of each algorithm without malicious vehicle.

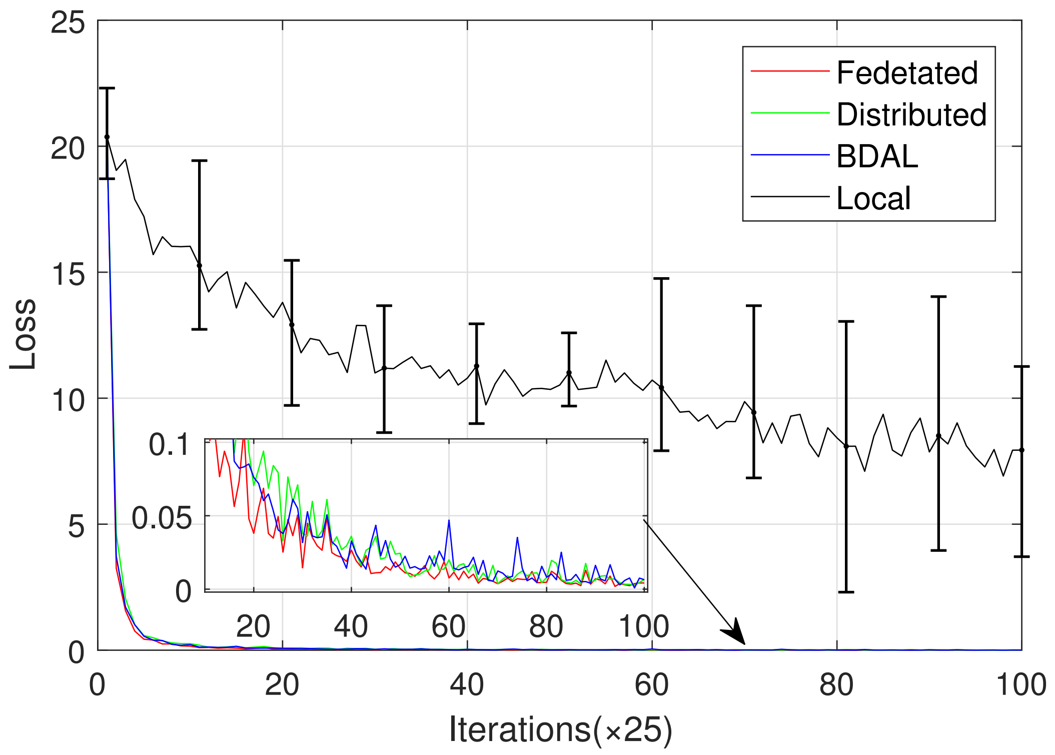


Figure 6.7: The training loss of each algorithm without malicious vehicle.

learning and distributed learning are both affected by the malicious vehicle and stay around the initial value.

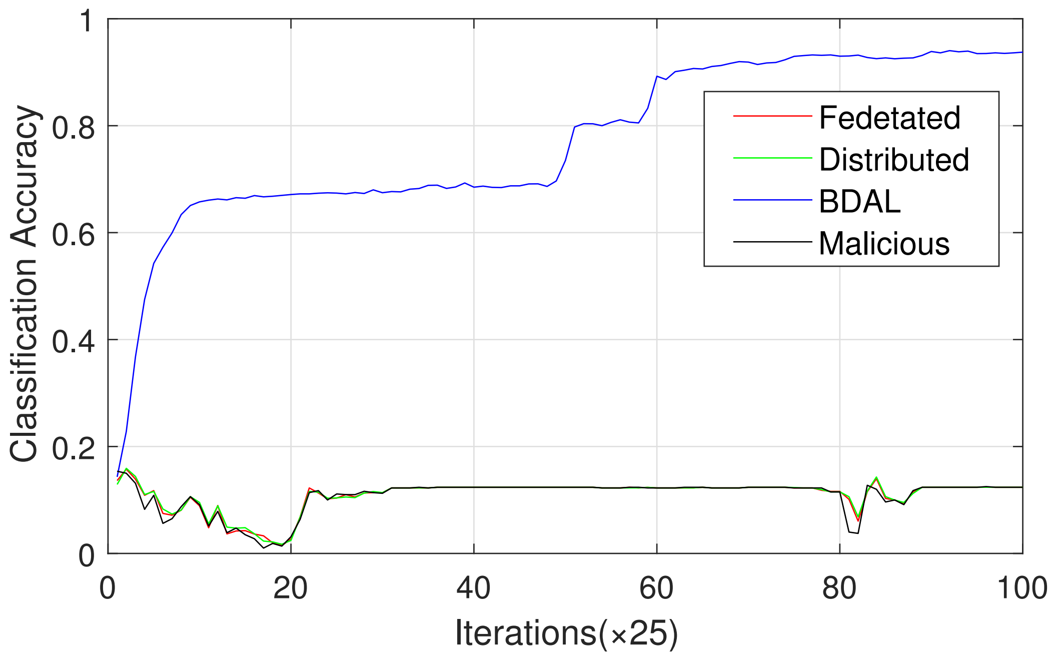


Figure 6.8: The classification accuracy on testing dataset of each algorithm with a malicious vehicle.

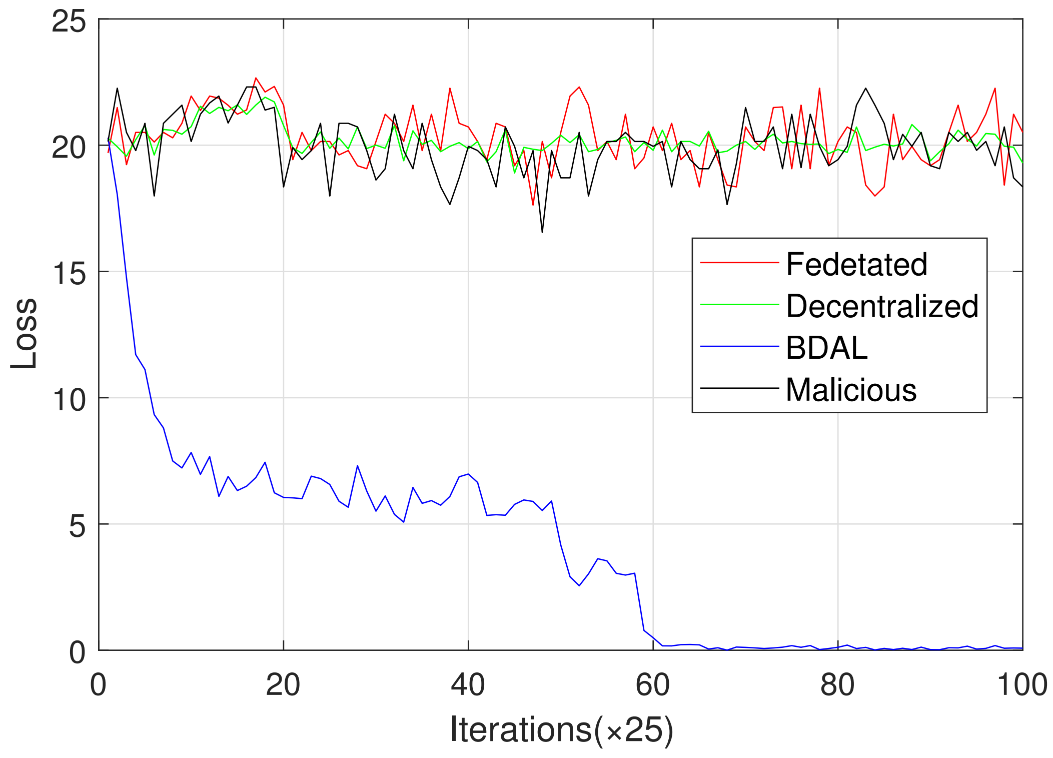


Figure 6.9: The training loss of each algorithm with a malicious vehicle.

6.5 Conclusion

In this chapter, a blockchain empowered distributed learning framework is proposed to realize the multi-vehicle intelligence in vehicular networks, which consists of the distributed learning framework and blockchain-based communication module. To circumvent the heavy communication burden on the center and improve the robustness of the centralized topology, the decentralized communication topology is developed for the vehicles in the vehicular network, and the consensus strategy is introduced for distributed training. Furthermore, the adaptive consensus algorithm is designed, which considers the model accuracy and computing contribution of each vehicle to adaptively adjust the weighted connectivity matrix during the consensus process. Moreover, the blockchain-based communication mechanism is introduced to ensure communication security, where the communication process is encrypted and recorded in the tramper-proof block. Besides, the proof of contribution protocol is designed to elect the vehicle to generate the new block in the blockchain, which not only saves computing costs but also encourage each vehicle to offer more computing power and a better model for the distributed learning in a vehicular network.

At last, a real-world traffic signal dataset is used to test the proposed BADL algorithm with comparison to local learning, federated learning, and distributed learning. The comparison study shows that the BADL algorithm has much higher accuracy than local learning and exhibits comparable performance with federated learning and distributed learning. In particular, both the federated learning and distributed learning are vulnerable to the malicious vehicle in the vehicular network, while the BADL algorithm is immune to the malicious attack.

Chapter 7

Distributed Training Framework for Reinforcement Learning

7.1 Introduction

Chapters 3-6 have proposed the distributed training algorithms and provided some optimization methods to improve its performance, reduce its computational costs and communication costs, and enhance its communication security. In this chapter, we are expected to extend the distributed training algorithm to reinforcement learning using the similar consensus strategy.

Reinforcement learning is about an agent exploring an unknown environment to learn an optimal policy for a given task by iteratively trial and learning. Through iteratively updating the values of actions based on the reward from the environment, the agent obtains the optimal strategy for a specific task. In the conventional tabular method [156], the values of all actions in different states are stored in a table, and an agent learns to update this table by iteratively interacting with the environment until the convergence of the table.

However, this tabular method is not suitable for complex reinforcement learning problems, especially for those with too many or even infinite states and actions, where the difficulties are not only about the storage of the large value table, but also the computational resources to fill them accurately. Therefore, the function approximation method is proposed, which aims to construct a prediction function for the action value based on the agent's existing interactive experience with the environment. Neural network approximation is a popular choice because of its excellent fitting capacity [157], among which, a representative work is deep Q-network (DQN) [80]. With the combination of Q-learning and deep neural networks, the deep Q-network has reached human-level performance on atari games [158, 159].

In recent years, the rapid development of mobile computing and Internet of Things provides a possibility to unleash the computing power of the device (or edge) side, where the training data samples are generated or collected. By pushing the learning process to the terminal devices, it not only takes better advantage of device computational power and alleviates the cloud server burden, but also offers more efficient service (or control) with a quicker response as it is closer to the practical control entities.

The existing distributed reinforcement learning algorithms are mostly based on the parameter server framework, which requires a central server to summarize all agents' model information or to gather all agents' experience data samples. This framework shows great efficiency by running multiple agents in parallel, but has two main issues. One issue is the heavy communication burden of the central server because all agents need to communicate with it simultaneously, and the other one is that the privacy issue makes it unavailable to collect all agents' experience data samples centrally. To deal with these issues, this chapter aims to design a distributed deep reinforcement learning framework based on a decentralized communication topology to free the central server and avoid revealing the private experience data.

In the distributed learning setting, the data storage, model computing, and updating are all pushed to the edge side, which unleashes the edge computing potential and alleviates the burden of the central server. Without a center to integrate all agents' information, the consensus algorithm is applied to make all agents approach each other during the learning process. Based on this, the deep distributed reinforcement learning framework is proposed, the convergence analysis of which demonstrates that the proposed method can drive all agents to the same optimum model. The simulation study shows that the distributed learning method has far better performance than its single learning counterpart.

The structure of this chapter is as follows. Section 7.2 proposes the distributed reinforcement learning problem; Section 7.3 designs the distributed deep reinforcement learning framework based on the decentralized communication topology with a server, and analyzes the convergence of the proposed algorithm. Section 7.4 conducts the comparison simulation to demonstrate the superiority of the distributed learning algorithm. Section 7.5 concludes this chapter.

7.2 Problem Formulation

Recently, the increasing computing power of intelligent devices and the development of Internet-of-Things have promoted the progress of edge computing, which not only takes better advantage of the distributed computational resources but also offers more efficient service. This also drives the exploration of the distributed reinforcement learning framework to deal with more complex problems in a more efficient manner. The current parallel training framework for reinforcement learning algorithms are mostly based on the parameter-server communication topology, where a central server is required to gather the information from all working agents. The main shortcoming of this architecture is the heavy communication burden on the central server, especially for those systems with high latency and low bandwidth. Besides, the growing concern on privacy protection also makes it unavailable to gather all agents' experience data centrally.

For these problems, we are trying to design a distributed training framework for the deep reinforcement learning algorithm, which is mainly based on the decentralized communication topology and requires only a little initialization information from the central server. Furthermore, this framework allows homogeneous agents to learn from other agents' experience to accelerate the training process, without the actual sharing of experience data samples.

7.3 Distributed Reinforcement Learning Framework

7.3.1 Distributed Deep Q-networks

A reinforcement learning agent can be divided into a learner and an actor. Common parallel reinforcement learning algorithms are based on the parameter-server communication architecture as shown in Figure 7.1 (a), where there is a central leaner connected with many actors. In this learning process, the central learner gathers the experience data from the actors, which is then used to update the reinforcement learning model. It significantly improves efficiency by learning the experience from different actors simultaneously. However, the centralization of experience data samples may not be available in some practical situations out of privacy issues, and the physical distance between the learner and actors also delays its response time.

Some researchers have proposed the asynchronous reinforcement learning framework [79, 160], where a central server connects with multiple agents in the parameter-server topology, each learner asynchronously updates the Q-network on the central server based on the experience of the corresponding actor on interacting with the environment. This not only benefits the privacy protection but also brings a quicker response, as it shortens the distance between the learner and the actor.

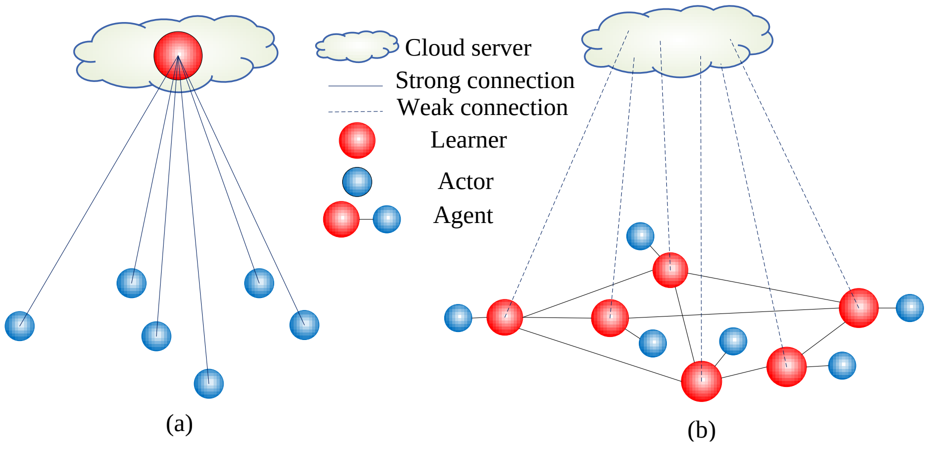


Figure 7.1: (a) Parameter-server communication topology. (b) Decentralized communication topology with a server.

However, the main disadvantage of the parameter-server architecture is the heavy communication burden on the central server, especially when the server is connected with large numbers of agents, as all agents need to communicate with the server during the training process. To circumvent this shortcoming, the decentralized communication topology is considered, where each agent only shares information with its connected neighbors, which do not require a center node. In addition, any agent's experience samples cannot be shared among agents due to privacy-preserving issues.

We explore to design a distributed training method for the communication framework shown in Figure 7.1 (b), where there are weak connections between the cloud server and the learners, and strong connections among these learners. In this communication manner, each learner only receives the initialization information from the server, such as the initialized Q-networks, the task, and the reinforcement learning environment. Meanwhile, all learners are allowed to use their neighboring information during the distributed training procedure to accelerate the learning process.

Generally, it requires more than ten times of consensus communication to drive all agents to their average, which costs a lot of computation and communication resources. However, a single consensus step still helps to make all agents approach each other. Based on the property of the weighted connectivity matrix W that $\sum_{j=1}^{K} w_{ij} = 1, \forall i$ and $w_{ij} \in [0,1), \forall(i,j)$, it is obtained

$$\min_k \|\theta'_k\| = \min_k \|\sum_{j=1}^{K} w_{kj}\theta_j\| \quad \geq \min_k \|\theta_k\|, \tag{7.1}$$

where the '=' holds only when all θ_j are equal. Similarly, it has $\max_k \|\theta'_k\| \leq \max_k \|\theta_k\|$, where we find that the lower bound of all model parameters and the upper bound of that decreases after a single consensus communication.

Since the update process of gradient descent optimizes the Q-network and the consensus process drives all Q-networks to approach each other, we propose the consensus-based distributed reinforcement learning algorithm for deep Q-networks. The aim is to train a shared Q-network model among connected agents over a decentralized communication topology without the actual sharing of experience samples.

First, all agents receive the initialization information from the cloud server, and the actors interact with the environment using random actions to accumulate the experience data samples in their replay memories. Then, the learning process begins, where the learner of each agent optimizes its Q-network using the gradient descent based on its own replay memory. After every learning step, all learners communicate with their neighbors to globally update their Q-networks using the consensus algorithm with a single communication. The specific update strategy of the proposed distributed reinforcement learning is

$$\theta'_k = \theta_k - \eta \nabla E(\theta_k), \tag{7.2}$$

$$\boldsymbol{\theta}'' = W \otimes I_m \boldsymbol{\theta}', \tag{7.3}$$

where θ'_k is the optimized Q-network of $\theta_k \in \mathbb{R}^{m\times s}$ using gradient descent method with $\nabla E(\theta_k)$ being the gradient, $\boldsymbol{\theta}''$ is the updated model matrix of $\boldsymbol{\theta}' = [\theta'_1, \theta'_2, ...\theta'_K]^T \in \mathbb{R}^{mK\times s}$ using the consensus algorithm with a single communication.

In this distributed training manner, each agent not only trains the Q-network based on its own replay memory but also learns from other agents' experience. Besides, it reduces the correlation of experience samples by running different agents in parallel as each agent can learn all agents' experience simultaneously. On the one hand, This decorrelation makes the selected data samples in a more stationary manner in every learning round as these agents are usually in different states at any certain time, which benefits the convergence of the learning process. On the other hand, this distributed learning manner makes it suitable for those on-policy reinforcement learning algorithms, such as Sarsa, because these parallel agents generate more experience data sample, which requires a small or even no replay memory, especially there are a large number of agents in this communication topology. The detailed process of the distributed deep Q-networks is summarized in Table 7.1.

7.3.2 Convergence Analysis

Let θ^* be the optimal Q-network model parameter of the reinforcement learning problem.

Assumption 1: The learning process makes each agent's Q-network model parameter θ_k approach the optimal Q-network model parameter θ^*. That is

$$\|\theta'_k - \theta^*\| \le \|\theta_k - \theta^*\|, \; k = (1, 2, ...K), \tag{7.4}$$

Table 7.1: The process of distributed deep Q-networks

```
Algorithm 1: Distributed learning of deep Q-networks
Inputs: The initialization information from the server, including the reinforcement learning environment, the Q-network structure, the number of agents $K$, and the weighted connectivity matrix $W$.
Outputs: The optimum deep Q-network model parameter $\theta^*$.
1: Each learner initializes the Q-network model parameter $\theta_k$ and the target model parameter $\hat{\theta}_k$.
2: Each actor $k$ interacts with its environment and accumulates the experience data samples $d_t^k = (s_t^k, a_t^k, r_{t+1}^k, s_{t+1}^k)$ in the replay memory $e_k$.
3: Each learner trains its Q-network model $\theta_k$ based on the replay memory $e_k$ and updates the target model parameter $\hat{\theta}_k$ after every $T$ iterations.
4: All learners communicate with each other and update their Q-networks $\theta_k$ using the consensus algorithm.
5: Return to step 2 with the optimized Q-network model $\theta_k$.
6: Check the stop criterion (such as a certain number of learning steps).
7: Obtain the optimal Q-network model $\theta^*$.
```

where θ_k' is the optimized Q-network model parameter of θ_k after the local learning process.

Remark 1: As the local learning is a complex non-convex optimization process, (7.4) may not be satisfied at every learning step using gradient descent method, while it should hold after multiple learning steps. In the situation that the model parameter approaches the optimal θ^* over multiple learning steps, we regard the multiple consensus processes over these learning steps as an entire consensus process. This does not influence the correctness of Proposition 1, and still makes the proof hold.

Assumption 2: The communication topology is connected with a spanning tree, and the weighted connectivity matrix W satisfies $\sum_{j=1}^{K} w_{ij} = 1$ and $w_{ij} \in [0, 1)$.

The update rule for each agent using the consensus algorithm is described by

$$\theta_k'' = \sum_{j=1}^{K} w_{kj}\theta_j', \; k = (1, 2, ...K), \tag{7.5}$$

where θ_i'' represents the updated Q-network model parameter θ_i' after the consensus process.

We define $\Delta\theta_k = \theta_k - \theta^*$ and $\Delta\theta_k' = \theta_k' - \theta^*$ as the gaps between the kth Q-network model parameter and the optimal Q-network parameter before and after the learning process, respectively. And we always have $\|\Delta\theta'_k\| \leq \|\Delta\theta_k\|$ based on the Assumption 1. Similarly, $\Delta\theta''_k = \theta''_k - \theta^*$ is defined as the gap between the optimal θ^* and the Q-network after the distributed learning process.

Proposition 1: Based on the Assumption 2, we have

$$\|(\Delta\theta''_1, \Delta\theta''_2, ...\Delta\theta''_N)\|_\infty \leq \|(\Delta\theta'_1, \Delta\theta'_2, ...\Delta\theta'_N)\|_\infty, \tag{7.6}$$

where $\|\cdot\|_\infty$ denotes the max-norm, and the '=' holds only when it satisfies $\Delta\theta'_i = \Delta\theta'_j, \forall(i,j)$.

Proof:

$$\begin{aligned} \|(\Delta\theta''_1, \Delta\theta''_2, ...\Delta\theta''_N)\|_\infty &= \max_k \|\Delta\theta''_k\| \\ &= \max_k \|\sum_{j=1}^{K} w_{kj}\theta'_j - \theta^*\| \\ &\leq \max_k \|\theta'_k - \theta^*\| \\ &= \|(\Delta\theta'_1, \Delta\theta'_2, ...\Delta\theta'_N)\|_\infty. \end{aligned} \tag{7.7}$$

Theorem 1: Under Assumptions 1 and 2, the distributed training Q-network model parameters converge to the optimal Q-network θ^* regardless of the difference in their initial statuses.

Proof: Suppose $G(\boldsymbol{\theta})$ is the maximal gap between all Q-networks $(\theta_1, \theta_2, ...\theta_N)$ and the optimal Q-network model parameter θ^*, this brings

$$G(\boldsymbol{\theta}) = \max_k(\|\theta_k - \theta^*\|) = \|(\Delta\theta_1, \Delta\theta_2, ...\Delta\theta_K)\|_\infty. \tag{7.8}$$

Combining Proposition 1 with (7.8), we get

$$\begin{aligned} G(\boldsymbol{\theta''}) &= \max_k(\|\theta''_k - \theta^*\|) \\ &= \|(\Delta\theta''_1, \Delta\theta''_2, ...\Delta\theta''_N)\|_\infty \\ &\leq \|(\Delta\theta'_1, \Delta\theta'_2, ...\Delta\theta'_N)\|_\infty \\ &= G(\boldsymbol{\theta'}). \end{aligned} \tag{7.9}$$

Based on Assumption 1,

$$\begin{aligned} G(\boldsymbol{\theta'}) &= \max_k(\|\theta'_k - \theta^*\| \\ &\leq \|\theta_k - \theta^*\| \\ &\leq \|(\Delta\theta_1, \Delta\theta_2, ...\Delta\theta_N)\|_\infty \\ &= G(\boldsymbol{\theta}). \end{aligned} \tag{7.10}$$

Thus, it is obtained

$$G(\boldsymbol{\theta''}) \leq G(\boldsymbol{\theta'}) \leq G(\boldsymbol{\theta}). \tag{7.11}$$

The above equation (7.11) demonstrates that upper bound of the gaps between all Q-networks and the optimal model keeps decreasing as the distributed learning process carrying on. This means that the Q-networks of all the agents in the communication topology converges to the optimal Q-network model parameters step by step. ■

7.4 Simulation and Discussion

The proposed distributed deep reinforcement learning framework consists of two update procedures of local learning and global consensus. After all agents pull the initialization information from the cloud server, the first step is local learning with their own copies of the environment, where all agents simultaneously interact with their environment and accumulate experience samples to trained their Q-networks. In this stage, each agent's Q-network is updated locally based on its experience reply memory. In the second step, all agents share the locally optimized Q-networks with their neighbors to globally update their models using the consensus algorithm. This two-phase update procedure makes all the agents converge to the optimum Q-network as presented in the above section.

Besides that no central node and no sharing of experience samples are required for this distributed training framework, another superiority is the flexibility and expansibility of the communication topology. Specifically, there is no limitation of the number of agents in this topology, and the convergence and efficiency of this method are unaffected in changeable communication topology [126] or the scenarios that any new agent participate in or leave this topology, provided that it has a spanning tree all the time.

We take the decentralized communication topology with a server in Figure 7.1(b) as an example, where six agents are strongly connected over the graph. In this communication topology, the experience samples of each actor are not allowed to be exchanged, and the sharing of Q-network information is only allowed among connected agents. Besides, there is a weak connection between these agents and the server, by which the agents pull the initialization Q-networks and some modeling hyper-parameter from the server. A comparison between the distributed learning and the single learning methods is made to demonstrate the effectiveness of the designed distributed training method.

- **Single learning**: a single agent interacts with the environment to update its Q-network, which is a baseline for the distributed learning method.
- **Distributed learning**: multiple agents are connected over a decentralized graph, and each agent trains its own Q-network by interacting with its copy of the environment, which is then globally updated by the consensus algorithm.

The inverted pendulum from the OpenAI gym environment (pendulum-V0) [161] is used to test the proposed distributed deep Q-networks, where the state information is taken as the input of the Q-network, while the corresponding output is the value of the action. The task of the agent is to learn how to keep the pendulum upright based on the reward from the environment, which is calculated by the following equation

$$r = -(\alpha^2 + \dot{\alpha}^2 + 0.001 * u^2), \tag{7.12}$$

where r denotes the reward, $u \in [-2, 2]$ is the action, which is discretized by evenly spaced actions. $\alpha \in [-\pi, \pi]$ is the angle between the inverted pendulum and the vertical direction (clockwise direction is positive), and $\dot{\alpha} \in [-8, 8]$ is the angular velocity.

After pulling the initialized Q-networks from the server, each learner optimizes its

Q-network to minimize the loss on the actor's experience samples, which is given by

$$l(\theta, d_i) = \frac{1}{n}\sum_{i=1}^{n}(q_i - Q(s_i, a_i; \theta))^2, \tag{7.13}$$

where n is the number of samples, s_i' is the next state of s_i for the sample $d_i = (s_i, a_i, r, s_i')$. $Q(s_i, a_i; \theta)$ and $q_i = r + \gamma \max_{a_i'} Q(s_i', a_i'; \hat{\theta})$ are the expected value of state-action pair (s_i, a_i) based on the current Q-network θ and the target value of that based on the target Q-network $\hat{\theta}$, respectively.

In this simulation, we use the accumulated reward $R(t)$ and the moving Q value $Q(t)$ to evaluate the performance of each agent, which are calculated by the following equations:

$$R(t) = \sum_{i=1}^{t} r(i), \tag{7.14}$$

$$Q(t) = \begin{cases} Q(s_t, a_t; \theta_t) & t = 1 \\ 0.99Q(t-1) + 0.01Q(s_t, a_t; \theta_t) & t > 1 \end{cases}, \tag{7.15}$$

where $R(t)$ is the reward from the environment computed by (7.12) at the tth iteration, $Q(s_t, a_t; \theta_t)$ is value of state based on the current Q-network.

It is clear from (7.12) that the agent always receives the negative reward from the environment, except the situation that the pendulum keeps upright, where the reward is zero, and correspondingly, the accumulated reward stops going down. Thus, in Figure 7.2, the horizontal part of each agent implies that the agent has finished the task and learns how to keep the pendulum upright stably.

Besides, we can find from Figure 7.2 that the accumulated rewards of the agents of distributed learning are above that of the single learning agent on the whole, which means that the distributed learning shows better overall performance than single learning. Meanwhile, the horizontal parts of the distributed learning agents occur around 3000 iterations, while the horizontal part of single learning starts to occur around 5900 iterations, which shows that the agents of distributed learning learn how to keep upright earlier than the single learning agent.

As the actual Q value changes dramatically over iterations, we thus use the value of moving Q computed by (7.15) to describe the changes in the current Q value

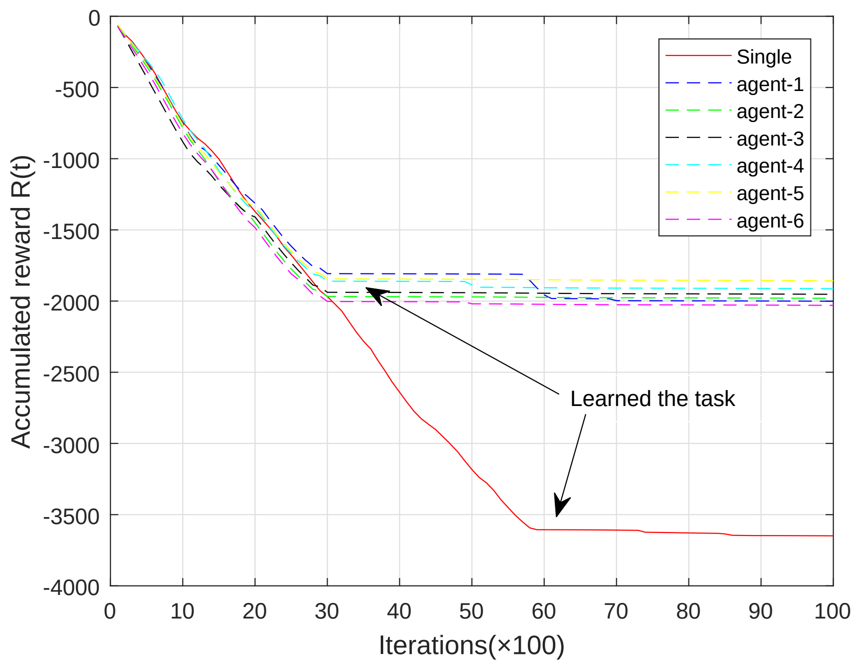


Figure 7.2: The accumulated reward of all agents.

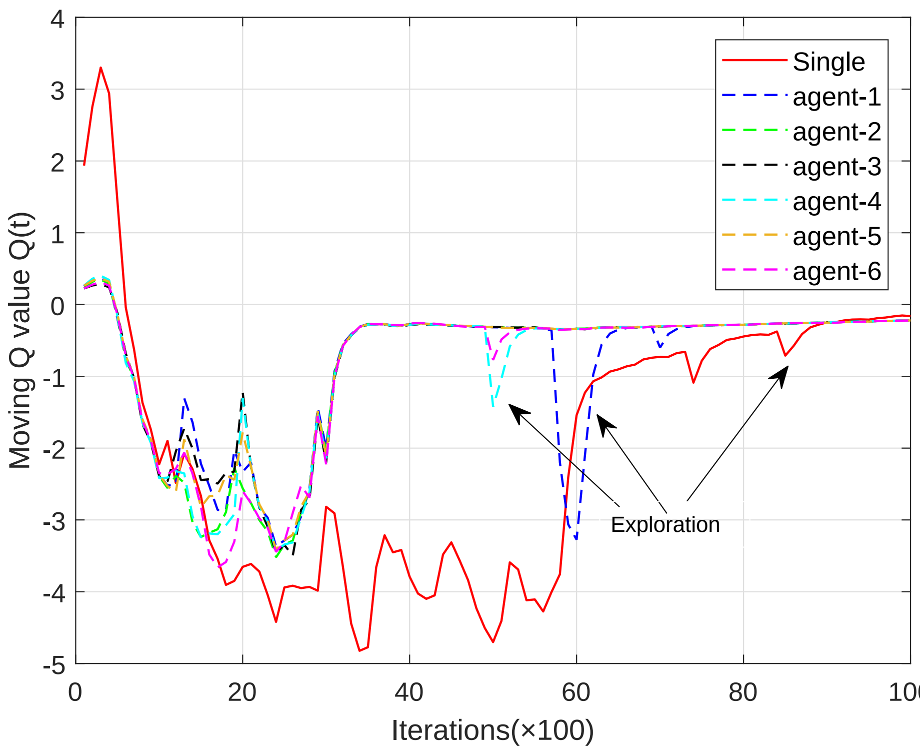


Figure 7.3: The moving Q value of all agents.

$Q(s_t, a_t; \theta_t)$, but the moving Q would have some delay compared to the actual Q value. An agent has learned how to keep the pendulum upright, when the moving Q value closes to zero and keeps horizontal. In Figures 7.3, it is clear that the moving

Q values of the agents using distributed learning are all larger than that of the single learning agent, which means that the agents in distributed learning always choose the actions with a higher value than that of single learning agent.

Furthermore, we use $cos(\alpha)$ to record the angular position of the inverted pendulum, where the agent keeps the pendulum upright when the value of $cos(\alpha)$ is close to 1 (α is close to 0). In Figure 7.4, the values of $cos(\alpha)$ for both distributed learning and single learning agents fluctuate between -1 and 1 at the beginning iterations, where all agents are learning and trying how to keep the pendulum upright. The values of $cos(\alpha)$ of agents in distributed learning reach and keep around 1 after 3000 iterations, when the agents in distributed learning learn to keep the pendulum upright. For single learning agent, the value of $cos(\alpha)$ keeps around 1 from 5900 iterations. This shows that the agents in distributed learning learn to keep the pendulum upright faster than the single learning agent. It is notable in Figures 7.3 and 7.4 that the agents deviates from the ideal position occasionally, and this is because of the exploration. That is, each agent has a small possibility to choose a random action other than the best action in any state.

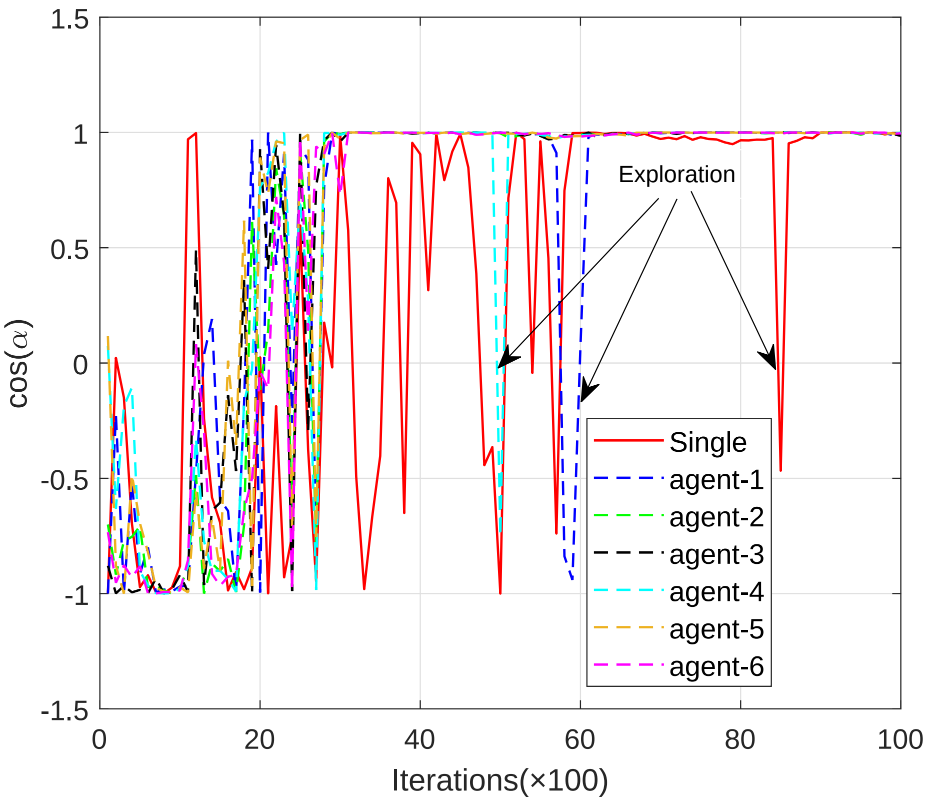


Figure 7.4: The $cos(\alpha)$ of all agents.

7.5 Conclusion

In this chapter, we propose a distributed training framework for deep Q-networks to deal with the large-scale reinforcement learning problems with privacy protection, where each agent can learn from other agents' experience without the actual sharing of experience data samples. This not only takes full advantage of edge computing resources but also alleviate the burden on the central server. In this communication topology, there is a weak connection between the server and the agents to transmit the initialization information for the reinforcement learning task, while all agents are strongly connected in a decentralized communication topology to share Q-network models, where the consensus algorithm is applied to make all agents approach each other. In this way, we change the learning process of deep Q-network into a two-phase update process, where the Q-network of each agent is locally updated based on its own experience first, and the Q-networks of all agents are then globally updated using the consensus algorithm. It is proved that the Q-networks of all agents converge to the optimum using the distributed learning framework for Q-networks. The proposed algorithm is tested on the inverted pendulum environment, which gives the result that the distributed learning algorithm shows better overall performance and learns how to keep the pendulum upright faster than the single learning agent.

Chapter 8

Distributed Deep Reinforcement Learning Method for Traffic Light Control

8.1 Introduction

Building on Chapter 7, we promote the distributed deep Q-networks with convolutional neural network to improve its capacity on image processing, and apply it in a traffic light control problem.

Traffic light plays an important and irreplaceable role in the modern transportation system, and the traffic light control strategy determines the efficiency of the transportation system to a large extent. Inefficient traffic light control strategy is more likely to result in more traffic congestion and longer waiting time for both vehicles and pedestrians, which increases the consumption of energy and time [162]. Thus, how to optimize the traffic light control strategy to improve traffic efficiency and shorten the vehicle waiting time has been a long-standing hot topic in the traffic and transportation field.

Traditional methods for traffic light control mainly include the fixed-time control strategy [163] and the adaptive control strategy [164]. By predefining the traffic light phase and its duration based on traffic demand estimation, the fixed-time strategy shows good performance in regular traffic flows, but performs poorly on time-varying traffic flows. Adaptive control strategy overcomes this drawback by adjusting the traffic light phase duration based on the real-time traffic conditions, such as the sensor data collected by the camera and radar. However, the randomness and complexity of the traffic system make it infeasible to build an accurate dynamic model to represent the actual traffic system without simplification, while the simplified model is often inefficient or inaccurate in real-world applications.

In recent years, the explosive development of artificial intelligence, especially deep reinforcement learning [165], provides a novel and promising solution for the traffic light control problem. Besides, the combination of Internet of Things [166] and the sensor network [167] makes it possible to collect and process large-scale real-time traffic data, which greatly benefits the modeling of the traffic system. Many researchers have done a lot of work on solving the traffic light control problem using deep reinforcement learning algorithms. For example, a convolutional neural network-based Q-network is developed in [168] for optimizing the traffic light control strategy, and a DRQN model is proposed in [169], which combines the convolutional neural network and recurrent neural network to build the Q-network. In [170], the authors combined the double deep Q-network, dueling, and prioritized experience replay to design the 3DQN model, which shows great efficiency and better performance than the DQN model. These researches focus on the optimization of the traffic light control strategy for either a single intersection or a particular traffic flow pattern.

However, the practical traffic flow is mostly time-varying and with great variations, such as the tidal traffic flow and non-uniform traffic demands [171], which makes the problem more complex. Specifically, the traffic demand of one direction may be much higher than its opposite in the morning rush hours (07:00-09:00) and be lower than its opposite in the afternoon rush hours (18:00-20:00) because of the tidal crowd. This situation is quite common for those intersections around large factories and schools. Besides, the traffic demands may also be non-uniform, for example, the demand for left-turn is dominated at one intersection, while the demand for straight is the largest at another intersection. Thus, multi-intersection with different traffic flow patterns should be considered in real-world applications.

For multi-intersection traffic signal timing optimization, the authors proposed a distributed multi-agent Q learning in [172], which takes the traffic information at the neighboring intersections into consideration to improve the overall performance. Ge *et al.* [173] developed a cooperative deep Q-network with Q-value transfer model for multi-intersection signal control, where the latest actions of neighboring intersections are considered in policy learning for cooperatively working. In [174], a multi-agent advantage actor-critic (A2C) algorithm is proposed, which includes neighbors' policies, states, and rewards to improve the observability of each intersection. These studies on multi-intersection problems consider the experience information of neighboring intersections, which dramatically increases the modeling complexity because more state space and/or action space are involved, especially when many intersections are involved. Besides, even though the neighboring information is involved in the reinforcement learning process, each agent still only learns from its own experience samples, while the neighboring experience samples are discarded.

It is well known that reinforcement learning-based methods require a large number of experience samples for the modeling process to optimize the policy for a given task. Besides, the built reinforcement learning model on the experience of a single intersection may turn to be powerless when it comes to the traffic conditions that it never learned in its experience. This point is a common shortcoming for the current deep learning and reinforcement learning algorithms [175], and a feasible solution is to increase the training samples as much as possible, which brings more resource consumption for a single agent. A direct idea is to collect the experience samples from a number of different intersections to train a combined model. However, the collection of traffic data from different intersections is time-consuming and energy-consuming (especially for the traffic picture data) [176], which also increases the communication burden.

Most existing distributed systems are based on centralized communication topology, such as the popular federated learning framework [177], which relies heavily on the central server for the distributed training process. This centralized communication topology suffers from the heavy communication burden on the center and may lead to the whole system shut down because of the failure of the central server.

To circumvent these issues, we are expected to design a distributed reinforcement learning algorithm based on a fully decentralized communication topology for the traffic light control problem. The main contribution of this work is summarized as follows:

1) A distributed reinforcement learning framework is proposed to train a combined model on all agents' experience, without the sharing of experience samples to save communication costs;
2) The proposed method is based on a fully decentralized communication topology, which doesn't require a central server and has great expansibility;
3) The proposed method is suitable for the modeling of different intersections with different flow patterns, which exhibits good robustness.

8.2 Motivations

The increasing vehicle ownership and transportation activities have promoted higher demands on both the road efficiency and road safety. As the key hinge and coordinator, traffic light plays an important role in the road traffic system. A proper traffic light control strategy not only improves transportation efficiency and safety but also saves energy and time consumption.

Nowadays, with the great development of sensor technology and computing power of intelligent devices, it is easy to collect and process road traffic information. However, a practical problem is how to use the rich traffic information and the strong computing power to optimize the traffic light control strategy to offer better and more efficient transportation services. The rapid progress of artificial intelligence offers a perfect opportunity to solve this problem in a more intelligent manner. Besides, the advent of Internet of Things also brings us another possibility to optimize the traffic light control strategy in a more efficient way under the cooperation of a number of traffic light agents.

Based on the above statement, we promote the following two open problems on the optimization of the traffic light control strategy.

1) How to design a deep reinforcement learning model to take full use of the traffic information to optimize the traffic light control strategy;
2) How to take full advantage of the different traffic flow patterns at different intersections to optimize the modeling process for traffic light control and improve its efficiency and effectiveness.

8.3 Traffic System Modeling

8.3.1 Traffic Light Control

Vehicles from different directions commonly have different destinations at an intersection, and the combination of traffic signals for different directions is required to allow all vehicles to pass this intersection safely. A legal combination of traffic light signals of all directions with their corresponding time durations is called a traffic light phase, and the change of all phases in a given sequence forms a traffic light control cycle [178]. Taking the intersection shown in Figure 8.1 is as an example, this intersection contains three in-lanes and three out-lanes at each leg of the intersection. Every in-lane has one or multiple permissible destination directions, including straight, left-turn, right-turn, and U-turn, while each leg has a traffic light signal to control the incoming vehicles. A key problem for the intersection is to develop an optimized traffic light control cycle to increase its traffic efficiency and decrease the total waiting time of all vehicles that pass the intersection.

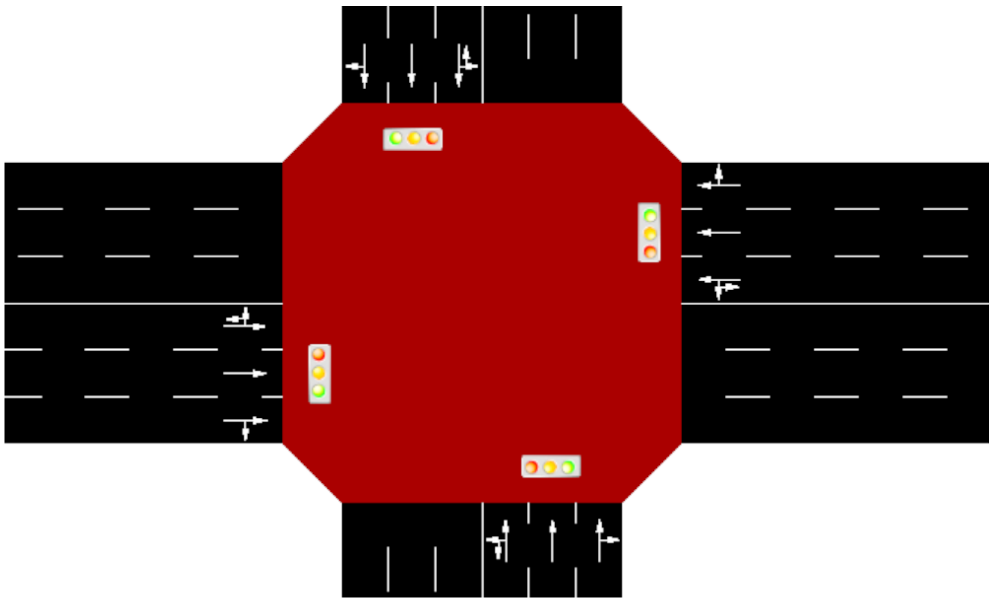


Figure 8.1: An example of the intersection.

8.3.2 System Modeling

Besides the reinforcement learning algorithm design, another key point is to establish the reinforcement learning environment, which consists of the three basic elements of state, action, and reward.

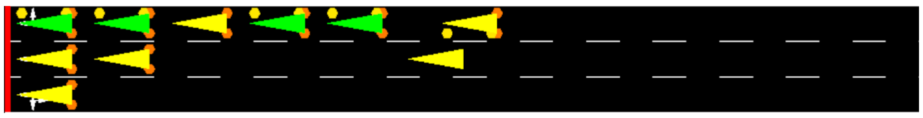
1) State

the element of state s is to describe the agent's current situation in the reinforcement learning environment.

For this traffic light control problem, the element of state should at least include the current traffic light signals and the current traffic flow information. In general, the more useful information the state has, the better it benefits the modeling process. We take the positions and velocities of all the vehicles that are within a certain distance from the traffic light to represent the current traffic flow information. This information is easy to obtain from the sensors, such as camera, radar, and inductive loop. To quantize this traffic flow state, the lanes of each leg of the traffic light (east, west, north, and south) that l length from the stooping line is discretized by a small length

c, by which, each lane is divided into l/c small grids.

Figure 8.2 is taken as an example of the traffic flow state, which shows the traffic information on the east side of the traffic light and its corresponding position and velocity matrix. We can find from this figure that the east lanes are divided into 3×12 grids, with the boolean value 1 and 0 representing whether there is a vehicle in the grid or not, respectively. Correspondingly, the velocity matrix gives each vehicle's velocity, and the value defaults as 0 if there is no vehicle in the grid. Combining the lanes of all directions (east, west, south, and north) together, the traffic flow state forms two 12×12 matrices, which describe all vehicles' current position and velocity information, respectively.



(a)

1	1	1	1	1	1	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0

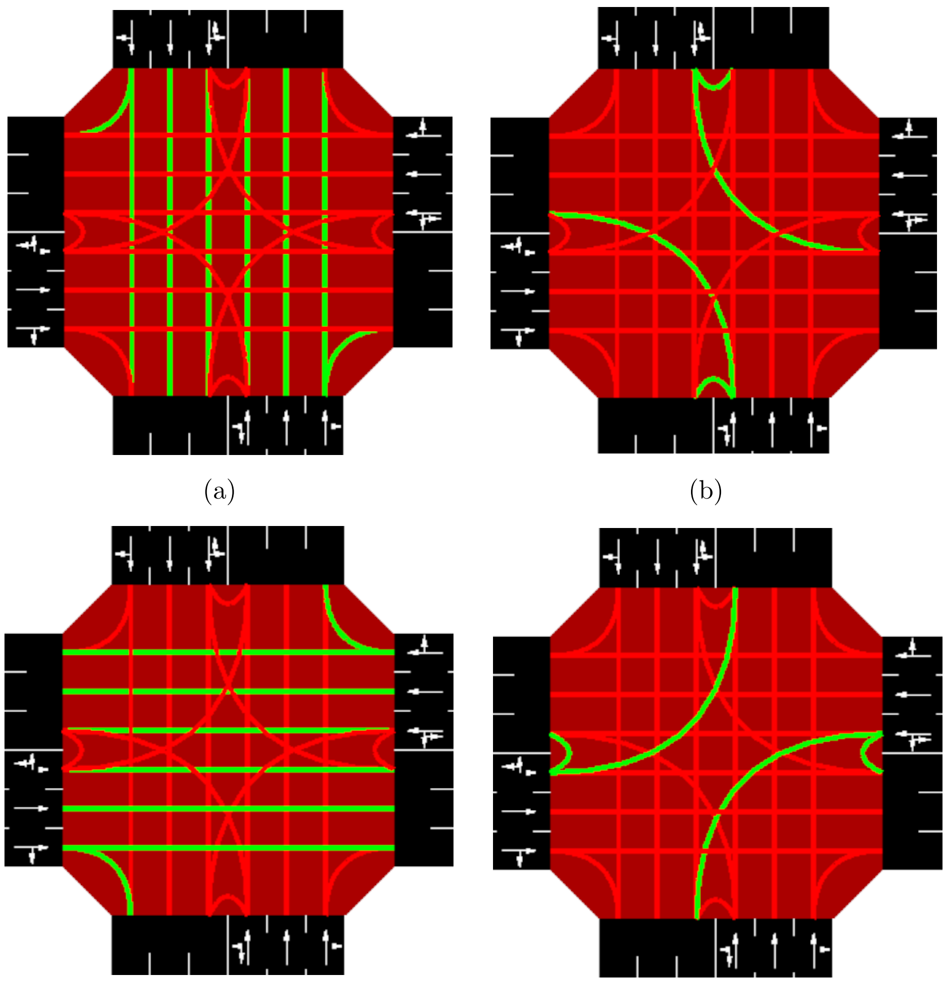
(b)

0	0	0	0	0	0.56	0	0	0	0	0	0
0	0	0	0	0	1.05	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

(c)

Figure 8.2: (a) An example of traffic flow state information. (b) The corresponding position matrix. (c) The corresponding velocity matrix.

As mentioned above, we also need to define the traffic light state. Figure 8.3 shows four standard red-green traffic signals combination, which contains 8.3(a) N-S (North-South) straight with right-turn, 8.3(b) N-S left-turn with U-turn, 8.3(c) E-W (East-West) straight with right-turn, and 8.3(d) E-W left-turn with U-turn. The shift sequence of the above four red-green phases (a-b-c-d) makes up a traffic light cycle. Notably, there is a red-yellow signal phase between any two red-green phases, that is, all current green signals turn to yellow with red signals unchanged. The combination of traffic flow state and traffic light state forms the state of the reinforcement learning environment for the traffic light control problem.



(a) (b)

(c) (d)

Figure 8.3: The four red-green traffic light phases.

2) Action

the element of action is to shift the agent from the current state to the next state, and correspondingly, the agent receives feedback from the environment to evaluate this chosen action.

For this traffic light control problem, we define the action as to whether or not to shift to the next traffic light phase based on the current traffic flow state and traffic light state, which is described by the boolean value 1 and 0. In this case, we only need to take the four red-green traffic light phases as the traffic light state, as the shift of traffic light states is automatically interrupted by a red-yellow phase. More specifically, assuming that the current traffic light phase is (a), the traffic light remains in phase (a) if the action is 0, and contrarily, the traffic changes to the next phase (b) if the action is 1, after a buffer of red-yellow signal phase.

3) Reward

The above-mentioned feedback from the environment is called as reward.

We define the reward for this traffic light control problem as the change of the total vehicle waiting sequence length (the number of vehicles that are waiting) before and after the action, that is

$$r = \sum_{i=1}^{n} l'_i - \sum_{i=1}^{n} l_i, \tag{8.1}$$

where n is the number of lanes of the intersection, l_i and l'_i are the vehicle waiting sequence length of lane i before and after the action, respectively.

At last, we evaluate the performance for this traffic light control problem by the total waiting time of all vehicles that pass the intersection during an episode (a period of time). Therefore, the reinforcement learning algorithm aims to minimize the total waiting time, which is defined by

$$L = \sum_{t=1}^{T} \sum_{i=1}^{n} l_i, \tag{8.2}$$

where L represents the total waiting time during the period of time T.

8.4 Distributed Deep Reinforcement Learning Architecture

8.4.1 Deep Q-network Model for Traffic Light Control

As described in section 8.3.2, we have quantized the traffic flow state into two matrices, which retains almost all useful traffic flow information. The next problem is how to design the Q-network to learn the state information, and we hope to take advantage of the position and velocity matrix as well as possible, which helps the traffic light agent to understand the traffic state better. Convolutional neural network [179] seems a good choice for the design of the deep Q-network structure for traffic light control problem, as it has shown great success in image processing, which is quite similar to the digital matrix. Through the operations of convolution and pooling, the convolutional neural network takes full use of the image to extract useful information, which greatly benefits the modeling process.

The designed CNN-based deep Q-network structure is shown in Figure 8.4, which includes convolutional layers and fully-connected layers. After the operation of convolution and pooling, the processed traffic flow state information is combined with the traffic light state, which is then transmitted to the fully-connected layer to compute the output. This structure takes advantage of CNN to understand the complex traffic flow information, which significantly improves the information utilization and model efficiency. This answers the first question promoted in section 8.2.

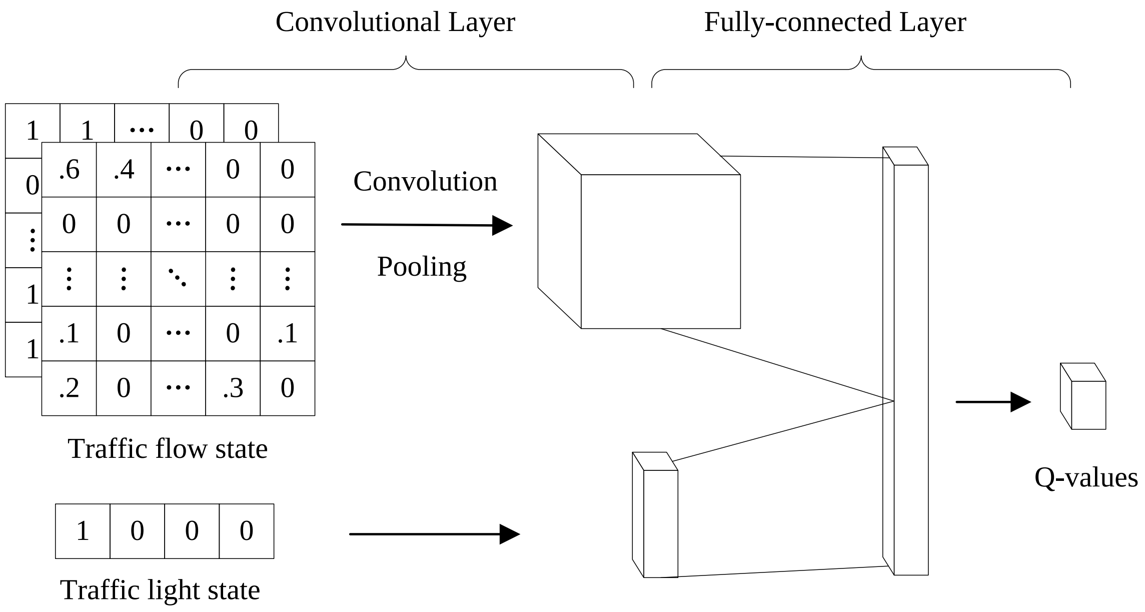


Figure 8.4: The CNN-based deep Q-network structure.

8.4.2 Distributed Deep Q-networks

As stated in section 8.2, it is reasonable that the cooperation of a number of traffic light agents in different intersections benefits the modeling process. Parallel computation is a promising method for multi-agent machine learning modeling, which is mainly based on the centralized and the decentralized communication topologies. The centralized topology as shown in Figure 8.5(a) needs a central server to coordinate all agents during the modeling process, and all agents need to communicate with the server at every iteration [110]. This poses a heavy communication burden on the server and even may cause communication jam, especially for those systems with high latency and low bandwidth.

To circumvent this drawback, the decentralized topology is designed as shown in Figure 8.5(b), where there is no central server and the communication is only required among connected neighbors. This not only frees the server to avoid communication jam but also shows great expansibility and robustness, where the change of communication topology does not affect the convergence and effectiveness of the whole system, provided a spanning tree exists in the topology [89].

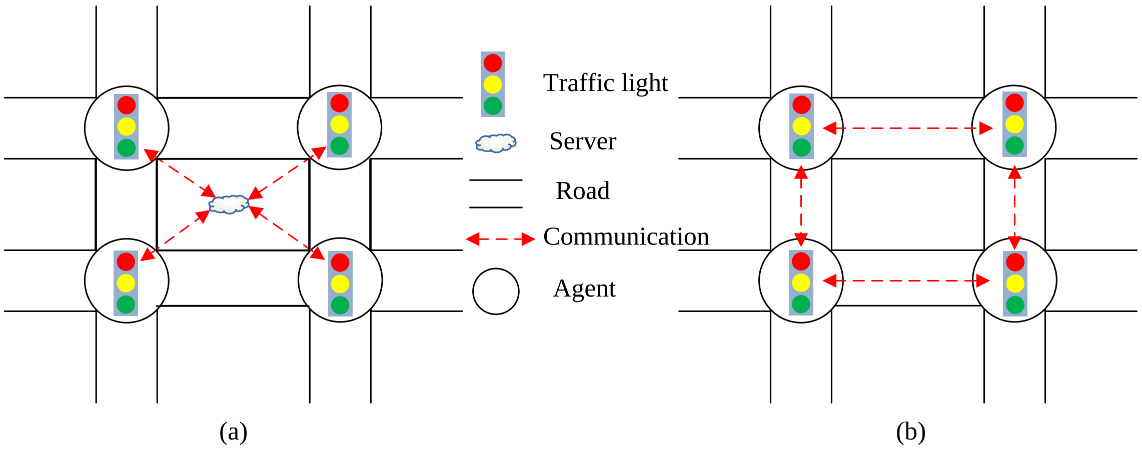


Figure 8.5: Two typical communication topologies for traffic lights.

The consensus algorithm is designed to make all agents over a decentralized topology to approach each other with only communication among connected neighbors. We assume that the decentralized topology is an undirected graph in this chapter, and the Max-degree [92] strategy is used to obtain the weighted connectivity matrix W in this chapter.

As mentioned in Chapter 7, the main process of the proposed distributed training framework for deep Q-networks is as follows,

$$\theta'_k = \theta_k - \eta \nabla E(\theta_k), \tag{8.3}$$

$$\boldsymbol{\theta}'' = W \otimes I_m \boldsymbol{\theta}', \tag{8.4}$$

where θ'_k is the locally trained deep Q-network of agent k with η and $\nabla E(\theta_k)$ being the corresponding learning rate and gradient, respectively. $\boldsymbol{\theta}'' = [\theta''_1, \theta''_2, ..., \theta''_N]$ and $\boldsymbol{\theta}' = [\theta'_1, \theta'_2, ..., \theta'_N]$ is the concatenation of all agents' deep Q-networks after and before the consensus process.

The proposed distributed deep Q-networks for the traffic light control problem consists of a two-phase updating process of local learning and global consensus. Local learning described by (8.3) is the first phase, where all traffic light agents optimize their deep Q-network models based on their local experience samples on interacting with the traffic environment. In the second phase of global consensus (8.4), each traffic light agent shares its local model with its connected neighbors, and the consensus algorithm is applied to update their deep Q-networks globally. This answered the second question promoted in section 8.2.

8.5 Simulation and Discussion

The decentralized topology in Figure 8.5(b) is taken as an example, where four traffic light agents are connected over the decentralized communication graph. Here, we assume that the distance between any two intersections in this figure are far enough that the vehicles from an intersection do not affect the traffic flow of another intersection. In other words, these traffic intersections are uncoupled.

During the distributed learning process, the experience samples of different traffic light agents are not permitted to be shared, while the exchange of deep Q-network model parameters is only allowed among connected neighbors to globally update their deep Q-networks. The CNN-based deep Q-network consists of two convolution layers, two pooling layers, and a fully connected layer. In detail, the first convolution layer is 16 filters of size 4×4 with 1 stride, while the second convolution layer is 32 filters of size 2×2 with 1 stride, and a max-pooling layer is connected behind each convolution layer. Besides, there are 256 and 2 nodes for the fully connected layer and the output layer, respectively. Except that the output layer takes softmax activation function, both convolution and fully connected layers adopt the rectified linear unit (ReLU) activation function, and the Adam gradient descent [155] with 1×10^{-4} learning rate is used for the optimization process. A comparison among the following four algorithms is made to verify the superiority of the proposed distributed deep Q-networks on the traffic light control problem. All the simulations are conducted in SUMO (Simulation of Urban Mobility) [180] and Python.

- **Fixed-time**: the duration of each phase of the traffic light cycle is fixed, and the performance of which is a baseline for the other methods.
- **Local learning**: each agent (traffic light) optimizes the deep Q-network based on its own experience of interacting with the environment, and there is no communication among all agents.
- **Distributed learning**: a number of agents are connected in a decentralized communication topology as shown in Figure 8.5(b), and the consensus algorithm is applied to globally update their deep Q-networks after every local optimization process based on each agent's experience samples.
- **Centralized learning**: agents are connected in a centralized topology as shown in Figure 8.5(a), where a central server is used to average all agents' model after every local learning step.

For all of the above four algorithms, we fix the red-yellow phase on 4s for safety concerns, and set the measuring unit of the red-green phase as 6s to ensure more flexibility and adaptiveness of the control strategy. As the action of the traffic light agent is whether to shift to the next traffic light phase or not, the duration of the red-green phase should be an integer multiple of 6s for local learning, distributed learning and centralized learning algorithms. As the comparison, we try different red-green phase durations in the range of [6s, 60s] with the interval of 6s to obtain the fixed-time control strategy with the minimum average waiting time.

8.5.1 Simulation Results on Homogeneous Traffic Flow

First, we consider the ideal situation that the traffic demands (straight, left-turn, right-turn, and U-turn) from each direction are uniform, that is, the vehicles come from the four directions and their corresponding traffic demand are with the same possibility. Besides, we assume that the four intersections shown in 8.5(b) are under the homogeneous traffic flow pattern. The details of vehicle generation probability are given in Table 8.1, where the value in the table means the probability of generating a corresponding vehicle at every second. Notably, as the demand of U-turn is usually small in practice, we thus still set it with a small value.

Table 8.1: The vehicle generation probability of the homogeneous traffic flow.

From / To	east	west	south	north
straight	0.03	0.03	0.03	0.03
right-turn	0.03	0.03	0.03	0.03
left-turn	0.03	0.03	0.03	0.03
U-turn	0.005	0.005	0.005	0.005

For this ideal traffic pattern, we test the above four algorithms in a traffic episode with 900s, and the simulation results are shown in Figure 8.6, where the full lines are the medians with the shaded area representing the minimum and maximum of all agents. All agents in local learning, distributed learning, and centralized learning are with the same initialization to train their deep Q-network models, while interacting with the traffic environment as detailed in section 8.3.2.

We find from Figure 8.6 that, for the homogeneous traffic situation, distributed learning and centralized learning have similar performance in average waiting time, which is much lower than that of local learning and fixed-time control strategy. Besides, the median of local learning agents shows less average waiting time than the fixed-time control strategy in the last few episodes, while it exhibits remarkable fluctuations during the learning process, as a small batch size is taken for the gradient descent method in this simulation. As a comparison, the distributed learning and centralized learning agents have few fluctuations during the learning process, because the consensus process of multiple agents in distributed learning, to some extent, decreases this fluctuation

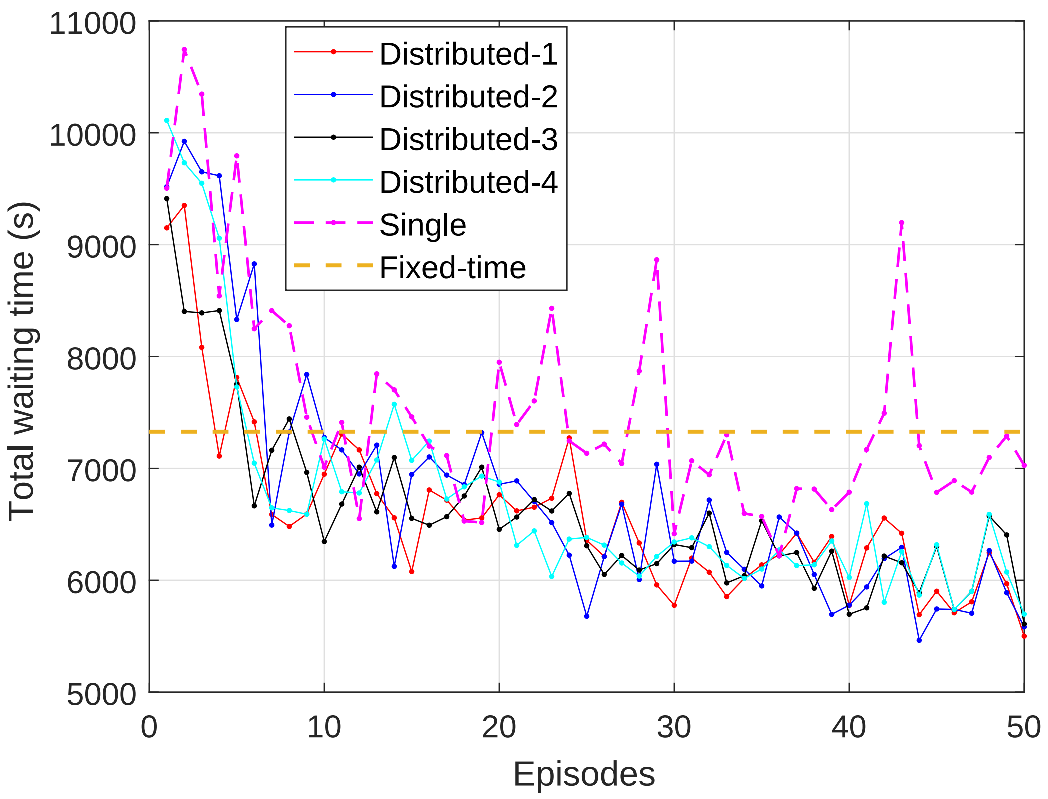


Figure 8.6: The performance of all algorithms on homogeneous traffic flow

introduced by the randomness.

8.5.2 Simulation Results on Heterogeneous Traffic Flow

As the traffic flow pattern in the above simulation is simple and ideal, we further verify the proposed distributed learning algorithm in a more practical traffic situation, where different intersections are under heterogeneous traffic flow patterns. Here, we consider the extreme case that there is only one major demand for each intersection, while the other demands are small. For example, the E-W straight is the major demand for the traffic pattern T_1, which thus has a larger vehicle generation probability, while the other demands are with a small probability. The detail of vehicle generation probabilities for the four heterogeneous traffic flow patterns are shown in Table 8.2.

In this scenario, different traffic light agents are in different traffic flow patterns and each agent only has access to the experience samples generated in its own environment. That is, the traffic light agent-k ($k \in \{1, 2, , 3, 4\}$) are under the corresponding

Table 8.2: The vehicle generation probability of the four traffic flow patterns.

Traffic pattern	From / To	east	west	south	north
T_1 (E-W straight major)	straight	0.08	0.08	0.01	0.01
	right-turn	0.02	0.02	0.02	0.02
	left-turn	0.02	0.02	0.02	0.02
	U-turn	0.005	0.005	0.005	0.005
T_2 (right-turn major)	straight	0.015	0.015	0.015	0.015
	right-turn	0.06	0.06	0.06	0.06
	left-turn	0.01	0.01	0.01	0.01
	U-turn	0.005	0.005	0.005	0.005
T_3 (left-turn major)	straight	0.015	0.015	0.015	0.015
	right-turn	0.01	0.01	0.01	0.01
	left-turn	0.06	0.06	0.06	0.06
	U-turn	0.005	0.005	0.005	0.005
T_4 (S-N straight major)	straight	0.01	0.01	0.08	0.08
	right-turn	0.02	0.02	0.02	0.02
	left-turn	0.02	0.02	0.02	0.02
	U-turn	0.005	0.005	0.005	0.005

T_k ($k \in \{1, 2, , 3, 4\}$) traffic flow pattern. In Figure 8.7, “Local-k”, “Distributed-k” and “Centralized-k” ($k \in \{1, 2, , 3, 4\}$) denote the simulation results of each agent on the training process for local learning, distributed learning, and centralized learning, respectively. It is seen from Figure 8.7 that the average waiting time of local learning, distributed learning, and centralized learning agents are similar during the training episodes on all of the four traffic patterns, which are lower than those of fixed-time strategy. This means that agents in all three learning methods achieve convergence on their own traffic flow patterns. However, these agents may not be robust enough to adapt to the traffic flow patterns at other intersections that they never learn in their experience.

To test the performance of local learning, distributed learning, and centralized learning models, we design a time-varying testing episode with $3600s$, which consists of four traffic flow patterns in the order of $T_1 - T_2 - T_3 - T_4$, and each pattern lasts for $900s$. Figure 8.8 describes the performance of the four methods on the designed testing episode, where the full lines represent the medians while the shaded area denotes the maximum and minimum of all the agents. It is seen from this figure that

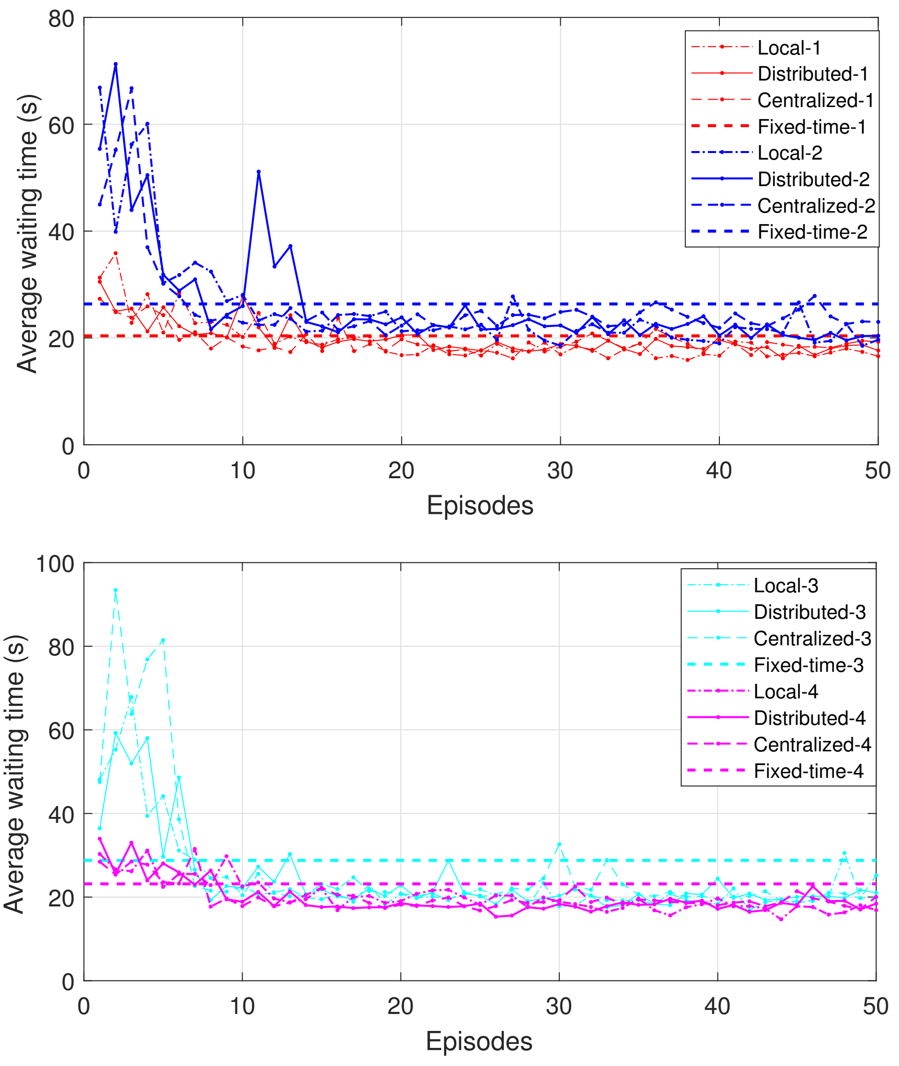


Figure 8.7: The training performance in heterogeneous traffic flows.

distributed agents and centralized learning agents have comparable good performance, which are much better than that of fixed-time strategy and local learning agents. In particular, the average waiting time of local learning agents are far higher than that of the other three methods. This means that local learning agents haven't achieved convergence on the testing episode and result in long average waiting time, even though they have nearly the same performance as distributed learning agents and centralized learning agents on the training episodes. This demonstrates that the distributed learning method is more efficient and robust as each agent in distributed learning only has access to the experience on a single traffic flow pattern but shows superior performance in all the four traffic patterns. Through the operation of the consensus process, each agent learns from other agents' experience without revealing its own data to other

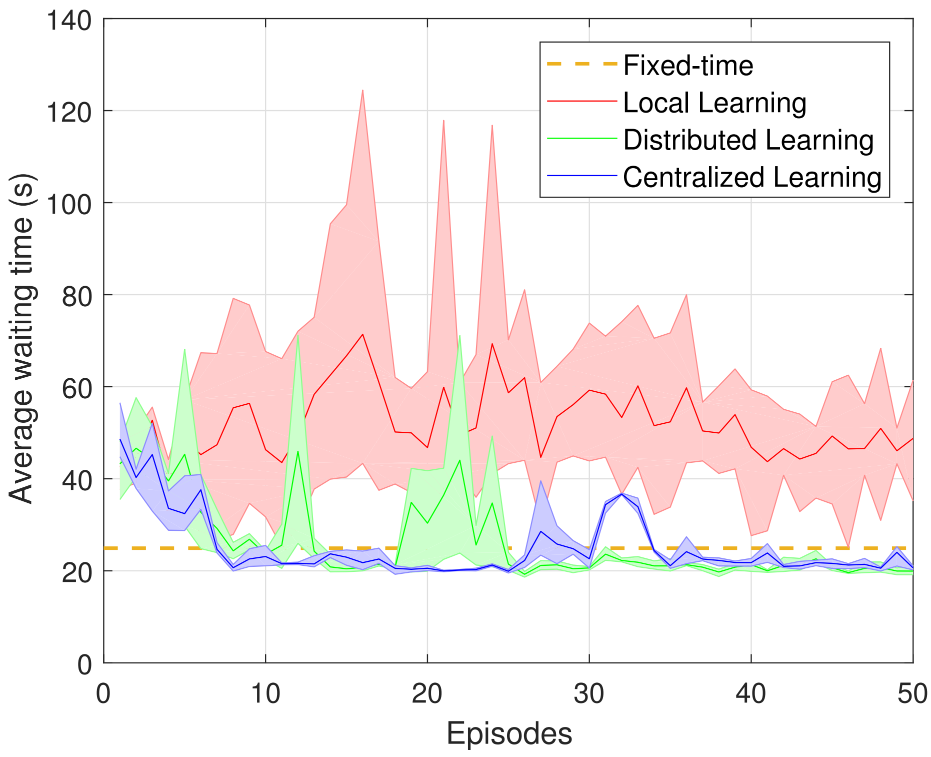


Figure 8.8: The testing performance in time-varying traffic flow.

agents, which greatly saves communication costs and computational time.

To further analyze the performance of all the four algorithms on each traffic pattern, we test the trained models on all of the four traffic patterns case by case, where each pattern represents an episode with 900s traffic flow under its vehicle generation probability as detailed in Table 8.2. It is seen from Figure 8.9 that the fixed-time control strategy exhibits fair performance in all of the four traffic pattern testing episodes, which is a baseline for the other three algorithms.

We can find from Figure 8.9 that distributed learning and centralized learning methods show great overall performance in all traffic flow patterns. For local learning, it only has acceptable performance in traffic patterns T_1 and T_4, even though it is still worse than those of distributed learning and centralized learning. As for the traffic patterns T_2 and T_3, only local learning agents that learn from this pattern (i.e., local-2 on T_2 and local-3 on T_3) show the performance as good as distributed learning agents, while the other situations of local learning agents even do not converge. Besides, for all the four patterns, the local learning agents show the best performance in the traffic pattern where it learns from (for example, local-1 on T_1) than the other situations (for example local-2 on T_1). This means that the local learning agents are more likely to fall into their local minimum on their own traffic patterns. As a comparison, the distributed learning agents exhibit similar and good performance in all the traffic patterns, no matter whether it learns from the traffic pattern or not.

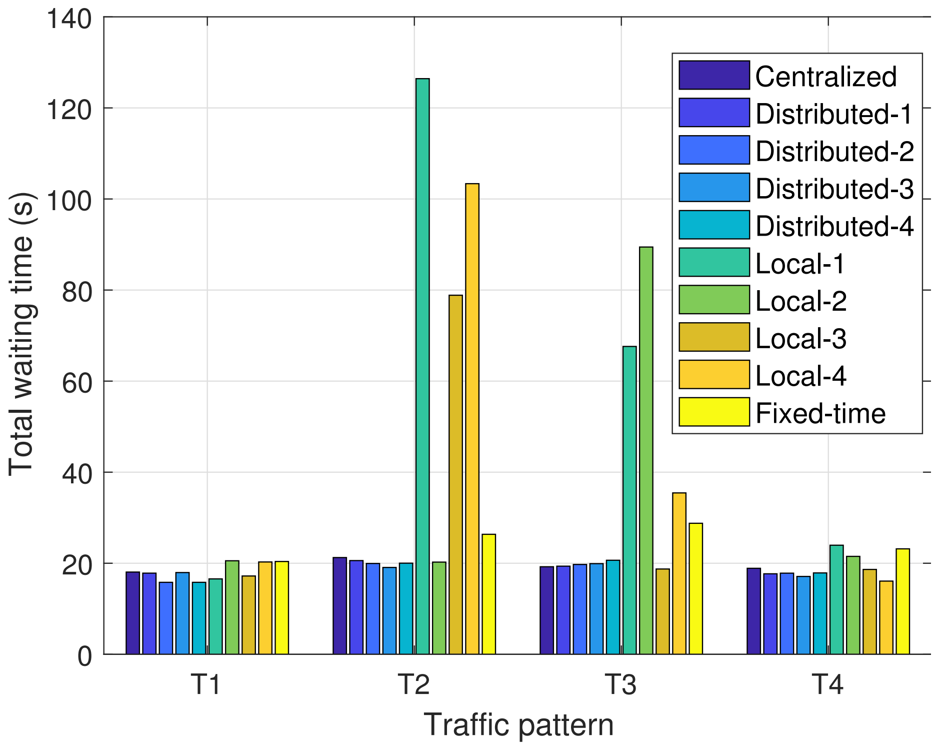


Figure 8.9: The performance of all algorithms on the four traffic patterns.

The above analysis demonstrates the superiority of the distributed learning on the traffic light control problems, especially in the more practical situation that different intersections are under heterogeneous traffic flow patterns.

8.6 Conclusion

In this chapter, we develop a distributed deep reinforcement learning method for the traffic light control problem. First, the three key elements of state, action, and reward of the reinforcement learning algorithm are defined to build the traffic environment. The traffic flow state is quantized by the position and velocity matrices, which is then processed by the convolutional layer. The processed traffic flow state is combined with the traffic light state, and then transmitted to a fully connected layer to obtain the state-action values. After the local optimization of the CNN-based deep Q-network based on each traffic light agent's experience of interacting with the environment, the consensus algorithm is applied to globally update the deep Q-networks of these traffic light agents connected over a decentralized communication topology.

At last, we design two simulation experiments to demonstrate the superiority of the proposed distributed learning algorithm with the comparison to the fixed-time control strategy, centralized learning, and local learning methods. The simulation shows that distributed learning method has similar performance with the centralized learning algorithm, and exhibits far better performance than the fixed-time strategy and local learning algorithms on both the homogeneous and heterogeneous traffic flow patterns.

Chapter 9

Conclusion and Future Work

In this chapter, we summarize the main contribution of this thesis, and discuss some possible future research work.

9.1 Conclusion

In this thesis, consensus-based distributed machine learning problems are systematically discussed, including theory, algorithm design, and engineering applications. With the explosive increase of data volume and more and more attention on privacy protection, traditional machine learning seems incapable of large-scale and privacy-related machine learning, as it needs to collect all data in a single agent (or computing node) for modeling. For this problem, Google proposed the federated learning in 2017, which allows all the agents to train a combined model, while keeping their private data locally stored. A potential bottleneck of this centralized training framework is the strong dependence on the central master, which causes a heavy communication burden. Decentralized topology is a promising alternative to the centralized topology, which does not require a central master but still allows all participants to approach each other and converge to their average. By deploying the machine learning algorithm in a decentralized communication topology, distributed learning is proposed, which allows all network-connected agents to train a combined model without the sharing of data samples. In this way, the central master is freed, which avoids the possible communication jam on the center, and the decentralized communication topology shows great robustness and expansibility. Based on this key idea, the main contributions of this thesis are summarized in the following two aspects.

1) Distributed supervised learning. We first propose a distributed training method based on the consensus algorithm for multi-layer neural networks over a decentralized communication topology, which only requires a single consensus step after every training step. The convergence analysis on empirical risk and model parameter gives the proof that distributed training allows all the agents connected in a decentralized communication topology to converge to the optimal model using the consensus algorithm. This is verified by the simulation experiment, which demonstrates that the proposed distributed training algorithm achieves comparable or even better performance than the centralized training model based on the entire dataset.

Furthermore, a heuristic adaptive consensus algorithm is proposed to adaptively adjust the weighted connectivity matrix, where a better agent is more likely to have a larger influence on its neighbors. Combining with the stochastic variance reduced gradient, the distributed training method is redesigned based on the SVRG and the heuristic adaptive consensus algorithm for neural networks connected in switching communication graphs. Theoretical analysis shows that SVRG reduces the variance introduced by SGD and improve its convergence rate with only a little extra computational cost, and all agents in switching graphs can still converge to the optimum with a single consensus communication at every training step, which verified the robustness of the distributed training framework. Simulations give the results that the heuristic adaptive consensus algorithm requires fewer iterations than the consensus algorithm with the fixed weighted connectivity matrix for all agents to reach the optimum, and SVRG greatly decreases the fluctuation caused by SGD and improves its performance.

To reduce the heavy communication costs during the distributed training process, the error-compensated compression method with bit-clipping is applied to compress the model parameter before sharing, which significantly saves communication costs. Then, a consensus-based distributed training framework with error-compensated communication compression is proposed to deal with both IID and non-IID datasets. The simulation study shows that distributed training with error-compensated communication compression is applicable on both IID and non-IID datasets, and shows comparable performance as distributed training and centralized training but saves a lot of communication costs.

In addition, a blockchain empowered distributed learning framework is proposed to realize the multi-vehicle intelligence in vehicular networks, which consists of the distributed learning framework and blockchain-based communication module. Besides, a new adaptive consensus algorithm is designed, which considers the model accuracy and computing contribution of each vehicle to adaptively adjust the weighted connectivity matrix during the consensus process. With comparison to local learning, federated learning, and distributed learning, the proposed blockchain empowered distributed adaptive learning algorithm is tested on a real-world traffic signal dataset, which shows that the proposed BADL algorithm has comparable performance with federated learning and distributed learning, and is immune to the malicious attack.

2) Distributed reinforcement learning. We extend the distributed training framework to reinforcement learning, where the classical reinforcement learning algorithm, deep Q-network, is taken as an example. In distributed deep Q-networks, all agents are connected over a decentralized communication topology to share their Q-network models, and the consensus algorithm is applied to make all agents approach each other. We change the learning process of deep Q-network into a two-phase update process, where the Q-network of each agent is locally updated based on its own experience first, and the Q-networks of all agents are then globally updated using the consensus algorithm. In this way, each agent learns from other agents' experiences without the actual sharing of experience data samples.

Moreover, we applied the distributed deep reinforcement learning framework in the traffic light control problem. The reinforcement learning environment for the traffic light control problem is built by defining the three key elements of state, action, and reward. After the local optimization of the CNN-based deep Q-network based on each traffic light agent's experience of interacting with the environment, the consensus algorithm is applied to globally update the deep Q-networks of these traffic light agents connected over a decentralized communication topology. The simulation experiments demonstrate the superiority of the proposed distributed learning algorithm with the comparison to the fixed-time control strategy, centralized learning, and local learning methods.

9.2 Future work

Distributed learning is a hot research topic for its great ability in large-scale and privacy-related machine learning problems. Even though that remarkable progress has been made in recent years, there still exists some open issues for both supervised learning and reinforcement learning in theory, algorithms, and applications. In terms of distributed supervised learning, potential future research directions are as follows:

- The convergence analysis for distributed training is based on some strong assumptions, such as convexity. A next plan is to prove the convergence of distributed training based on weaker assumptions, where specific neural networks can be considered for theoretical analysis, such as the over-parameterized neural networks [181].
- For algorithm design, more efficient model compression methods in distributed training should be considered to further reduce the communication costs. Meanwhile, the convergence of distributed training with communication compression should be carefully analyzed.
- During the proposed distributed training process, each agent needs to communicate with its neighbors synchronously after every learning step, which brings heavy communication costs. Future work can be to explore asynchronous communication strategy, where each agent shares its model after several learning steps. This strategy should be carefully designed to ensure convergence and model accuracy.

- For neural networks considered in this thesis, it is required to pre-define an appropriate structure before training, which is quite challenging. Other kinds of neural networks, such as the stochastic configuration networks [182], are worth careful study and extending to a distributed manner.

For distributed reinforcement learning, the following research topics are considered in future works:

- In the proposed distributed reinforcement learning setting, all agents are facing a similar task, this restricts the possible application of distributed reinforcement learning algorithms. A possible future work is to extend the distributed reinforcement learning for multi-agent reinforcement learning applications with diverse tasks.
- Notice that deep Q-network is designed for discrete reinforcement learning tasks. In future research, it is expected to extend the distributed reinforcement learning framework to other reinforcement learning algorithms for continuous tasks and apply the distributed reinforcement learning algorithms in practical engineering problems.

Bibliography

[1] W. Ertel, *Introduction to Artificial Intelligence.* Springer, 2018.

[2] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, “A survey on deep learning for big data,” *Information Fusion*, vol. 42, pp. 146–157, 2018.

[3] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[4] R. He, W. Zheng, B. Hu, and X. Kong, “Two-stage nonnegative sparse representation for large-scale face recognition,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 1, pp. 35–46, 2013.

[5] Y. Zhang, P. Li, Y. Jin, and Y. Choe, “A digital liquid state machine with biologically inspired learning and its application to speech recognition,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 11, pp. 2635–2649, 2015.

[6] Y. Miao, L. Yu, and P. Blunsom, “Neural variational inference for text processing,” in *International Conference on Machine Learning*, Jun 2016, pp. 1727–1736.

[7] N. R. Sabar, J. Abawajy, and J. Yearwood, “Heterogeneous cooperative co-evolution memetic differential evolution algorithm for big data optimization problems,” *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 2, pp. 315–327, 2017.

[8] Z. Wang, J. Liao, Q. Cao, H. Qi, and Z. Wang, “Friendbook: a semantic-based

friend recommendation system for social networks," *IEEE Transactions on Mobile Computing*, vol. 14, no. 3, pp. 538–551, 2015.

[9] J. B. Predd, S. B. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Processing Magazine*, vol. 23, no. 4, pp. 56–69, 2006.

[10] M. Fahimi and A. Ghasemi, "A distributed learning automata scheme for spectrum management in self-organized cognitive radio network," *IEEE Transactions on Mobile Computing*, vol. 16, no. 6, pp. 1490–1501, 2017.

[11] A. Broggi, A. Zelinsky, Ü. Özgüner, and C. Laugier, "Intelligent vehicles," in *Springer Handbook of Robotics.* Heidelberg, Springer, 2016, pp. 1627–1656.

[12] R. Gravina, P. Alinia, H. Ghasemzadeh, and G. Fortino, "Multi-sensor fusion in body sensor networks: state-of-the-art and research challenges," *Information Fusion*, vol. 35, pp. 68–80, 2017.

[13] N. Shaker, S. Asteriadis, G. N. Yannakakis, and K. Karpouzis, "Fusing visual and behavioral cues for modeling user experience in games," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1519–1531, 2013.

[14] L. Lin, W. Yang, C. Li, J. Tang, and X. Cao, "Inference with collaboratives model for interactive tumor segmentation in medical image sequences," *IEEE Transactions on Cybernetics*, vol. 46, no. 12, pp. 2796–2809, 2016.

[15] B. Lei, P. Yang, T. Wang, S. Chen, and D. Ni, "Relational-regularized discriminative sparse learning for alzheimer's disease diagnosis," *IEEE Transactions on Cybernetics*, vol. 47, no. 4, pp. 1102–1113, 2017.

[16] S. Li, L. Ding, H. Gao, C. Chen, Z. Liu, and Z. Deng, "Adaptive neural network tracking control-based reinforcement learning for wheeled mobile robots with skidding and slipping," *Neurocomputing*, vol. 283, pp. 20–30, 2018.

[17] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[18] J. Supancic III and D. Ramanan, “Tracking as online decision-making: Learning a policy from streaming videos with reinforcement learning,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 322–331.

[19] E. A. O. Diallo, A. Sugiyama, and T. Sugawara, “Coordinated behavior of cooperative agents using deep reinforcement learning,” *Neurocomputing*, vol. 396, pp. 230–240, 2020.

[20] F. Aznar, M. Pujol, and R. Rizo, “Obtaining fault tolerance avoidance behavior using deep reinforcement learning,” *Neurocomputing*, vol. 345, pp. 77–91, 2019.

[21] A. Botta, W. De Donato, V. Persico, and A. Pescapé, “Integration of cloud computing and internet of things: a survey,” *Future Generation Computer Systems*, vol. 56, pp. 684–700, 2016.

[22] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, p. 484, 2016.

[23] J. Pan and J. McElhannon, “Future edge cloud and edge computing for internet of things applications,” *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, Feb 2018.

[24] D. Ding, Z. Ding, G. Wei, and F. Han, “An improved reinforcement learning algorithm based on knowledge transfer and applications in autonomous vehicles,” *Neurocomputing*, vol. 361, pp. 243–255, 2019.

[25] H. Zhang, Y. Shu, P. Cheng, and J. Chen, “Privacy and performance trade-off in cyber-physical systems,” *IEEE Network*, vol. 30, no. 2, pp. 62–66, 2016.

[26] X. Liu, Y. Xia, W. Yang, and F. Yang, “Secure and efficient querying over personal health records in cloud computing,” *Neurocomputing*, vol. 274, pp. 99–105, 2018.

[27] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, “Chiron: Privacy-preserving machine learning as a service,” *arXiv preprint arXiv:1803.05961*, 2018.

[28] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[29] G. Skinner and T. Walmsley, "Artificial intelligence and deep learning in video games a brief review," in *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*. IEEE, 2019, pp. 404–408.

[30] A. Ferdowsi, U. Challita, W. Saad, and N. B. Mandayam, "Robust deep reinforcement learning for security and safety in autonomous vehicle systems," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 307–312.

[31] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines," *Journal of Machine Learning Research*, vol. 11, pp. 1663–1707, 2010.

[32] M. Stolpe, K. Bhaduri, and K. Das, "Distributed support vector machines: an overview," *Solving Large Scale Learning Tasks. Challenges and Algorithms*, pp. 109–138, 2016.

[33] L. Ho, J. Wu, and P. Liu, "Adaptive communication for distributed deep learning on commodity gpu cluster," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018.

[34] B. Zhang, J. Lam, and S. Xu, "Stability analysis of distributed delay neural networks based on relaxed lyapunov–krasovskii functionals," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 7, pp. 1480–1492, 2015.

[35] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[36] T. Chang, M. Hong, W. Liao, and X. Wang, "Asynchronous distributed ADMM for large-scale optimization – Part I: algorithm and convergence analysis," *IEEE Transactions on Signal Process.*, vol. 64, no. 12, pp. 3118–3130, 2016.

[37] X. Miao, Y. Liu, H. Zhao, and C. Li, "Distributed online one-class support vector machine for anomaly detection over networks," *IEEE Transactions on Cybernetics*, no. 99, pp. 1–14, 2018.

[38] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik, "Parallel support vector machines: the cascade SVM," in *Advances in Neural Information Processing Systems*, 2005, pp. 521–528.

[39] J. Zhang, Z. Li, and J. Yang, "A parallel SVM training algorithm on large-scale classification problems," in *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics.* IEEE, Aug 2005, pp. 1637–1641.

[40] E. Castillo, D. PeteiroBarral, B. G. Berdiñas, and O. Fontenla-Romero, "Distributed one-class support vector machine," *International Journal of Neural Systems*, vol. 25, no. 07, p. 1550029, 2015.

[41] W. Kim, M. S. Stanković, K. H. Johansson, and H. J. Kim, "A distributed support vector machine learning over wireless sensor networks," *IEEE Transactions on Cybernetics*, vol. 45, no. 11, pp. 2599–2611, 2015.

[42] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," *Google Research Blog*, vol. 3, 2017.

[43] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[44] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[45] T. Zhu, P. Xiong, G. Li, and W. Zhou, "Correlated differential privacy: Hiding information in non-iid data set," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 2, pp. 229–242, 2014.

[46] H. Zhu and Y. Jin, "Multi-objective evolutionary federated learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 4, pp. 1310–1322, 2019.

[47] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[48] Z. Yang, K. Yang, L. Lei, K. Zheng, and V. C. Leung, "Blockchain-based decentralized trust management in vehicular networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1495–1505, 2018.

[49] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 774–14 784.

[50] Y. Fu, F. R. Yu, C. Li, T. H. Luan, and Y. Zhang, "Vehicular blockchain-based collective learning for connected and autonomous vehicles," *IEEE Wireless Communications*, vol. 27, no. 2, pp. 197–203, 2020.

[51] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 4298–4311, 2020.

[52] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 4298–4311, 2020.

[53] H. Chai, S. Leng, Y. Chen, and K. Zhang, "A hierarchical blockchain-enabled federated learning algorithm for knowledge sharing in internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12, 2020.

[54] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, "Reliable federated learning for mobile networks," *IEEE Wireless Communications*, vol. 27, no. 2, pp. 72–80, 2020.

[55] S. Scardapane, D. Wang, M. Panella, and A. Uncini, "Distributed learning for random vector functional-link networks," *Information Sciences*, vol. 301, pp. 271–284, 2015.

[56] S. Scardapane, D. Wang, and M. Panella, "A decentralized training algorithm for echo state networks in distributed big data applications," *Neural Networks*, vol. 78, pp. 65–74, 2016.

[57] L. Georgopoulos and M. Hasler, "Distributed machine learning in networks by consensus," *Neurocomputing*, vol. 124, pp. 2–12, 2014.

[58] K. Yuan, Q. Ling, and W. Yin, "On the convergence of decentralized gradient descent," *SIAM Journal on Optimization*, vol. 26, no. 3, pp. 1835–1854, 2016.

[59] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 5336–5346.

[60] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, "D^2: Decentralized training over decentralized data," *arXiv preprint arXiv:1803.07068*, 2018.

[61] J. Chen, B. Chen, Z. Zeng, and P. Jiang, "Event-triggered synchronization strategy for multiple neural networks with time delay," *IEEE Transactions on Cybernetics*, vol. 50, no. 7, pp. 3271–3280, 2019.

[62] J. Wang, X. Zhang, H. Wu, T. Huang, and Q. Wang, "Finite-time passivity of adaptive coupled neural networks with undirected and directed topologies," *IEEE Transactions on Cybernetics*, vol. 50, no. 5, pp. 2014–2025, 2018.

[63] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *arXiv preprint arXiv:1903.02891*, 2019.

[64] H. Tang, S. Gan, C. Zhang, T. Zhang, and J. Liu, "Communication compression for decentralized training," in *Advances in Neural Information Processing Systems*, 2018, pp. 7652–7662.

[65] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[66] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[67] D. Ye, G. Chen, W. Zhang, S. Chen, B. Yuan, B. Liu, J. Chen, Z. Liu, F. Qiu, H. Yu *et al.*, "Towards playing full moba games with deep reinforcement learning," *arXiv preprint arXiv:2011.12692*, 2020.

[68] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A survey of deep reinforcement learning in video games," *arXiv preprint arXiv:1912.10944*, 2019.

[69] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6023–6029.

[70] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[71] C. You, J. Lu, D. Filev, and P. Tsiotras, "Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning," *Robotics and Autonomous Systems*, vol. 114, pp. 1–18, 2019.

[72] S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–20, 2020.

[73] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *International Conference on Learning Representations*, 2018.

[74] T. De Bruin, J. Kober, K. Tuyls, and R. Babuska, "Experience selection in deep reinforcement learning for control," *Journal of Machine Learning Research*, vol. 19, pp. 1–56, 2018.

[75] H. H. Zhuo, W. Feng, Q. Xu, Q. Yang, and Y. Lin, “Federated reinforcement learning,” *arXiv preprint arXiv:1901.08277*, vol. 1, 2019.

[76] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, “Federated learning in mobile edge networks: A comprehensive survey,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[77] A. Anwar and A. Raychowdhury, “Multi-task federated reinforcement learning with adversaries,” *arXiv preprint arXiv:2103.06473*, 2021.

[78] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen *et al.*, “Massively parallel methods for deep reinforcement learning,” *arXiv preprint arXiv:1507.04296*, 2015.

[79] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[80] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.

[81] D. Horgan, J. Quan, D. Budden, G. BarthMaron, M. Hessel, H. Van Hasselt, and D. Silver, “Distributed prioritized experience replay,” *arXiv preprint arXiv:1803.00933*, 2018.

[82] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.

[83] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, A. Muldal, N. Heess, and T. Lillicrap, “Distributed distributional deterministic policy gradients,” *arXiv preprint arXiv:1804.08617*, 2018.

[84] G. Sartoretti, Y. Wu, W. Paivine, T. S. Kumar, S. Koenig, and H. Choset, "Distributed reinforcement learning for multi-robot decentralized collective construction," in *Distributed Autonomous Robotic Systems.* Springer, 2019, pp. 35–49.

[85] D. Lee, H. Yoon, and N. Hovakimyan, "Primal-dual algorithm for distributed reinforcement learning: distributed gtd," in *2018 IEEE Conference on Decision and Control (CDC).* IEEE, 2018, pp. 1967–1972.

[86] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," in *International Conference on Machine Learning.* PMLR, 2018, pp. 3053–3062.

[87] T. Chen, K. Zhang, G. B. Giannakis, and T. Başar, "Communication-efficient distributed reinforcement learning," *arXiv preprint arXiv:1812.03239*, 2018.

[88] B. Liu, Z. Ding, and C. Lv, "Distributed training for multi-layer neural networks by consensus," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 5, pp. 1771–1778, 2020.

[89] B. Liu and Z. Ding, "Distributed heuristic adaptive neural networks with variance reduction in switching graphs," *IEEE Transactions on Cybernetics*, pp. 1–9, 2019.

[90] B. Liu and Z. Ding, "A consensus-based decentralized training algorithm for deep neural networks with communication compression," *Neurocomputing*, vol. 440, pp. 287–296, 2021.

[91] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[92] M. Toulouse, B. Q. Minh, and P. Curtis, "A consensus based network intrusion detection system," in *IT Convergence and Security (ICITCS), 2015 5th International Conference on.* IEEE, Aug 2015, pp. 1–6.

[93] S. Boyd, P. Diaconis, and L. Xiao, “Fastest mixing markov chain on a graph,” *SIAM Review*, vol. 46, no. 4, pp. 667–689, 2004.

[94] F. L. Lewis, H. Zhang, K. Hengster-Movric, and A. Das, *Cooperative Control of Multi-Agent Systems: Optimal and Adaptive Design Approaches.* Springer Science & Business Media, 2013.

[95] R. Fierimonte, S. Scardapane, M. Panella, and A. Uncini, “A comparison of consensus strategies for distributed learning of random vector functional-link networks,” in *International Workshop on Neural Networks.* Springer, 2015, pp. 143–152.

[96] G. Bandyopadhyay and S. Chattopadhyay, “Single hidden layer artificial neural network models versus multiple linear regression model in forecasting the time series of total ozone,” *International Journal of Environmental Science & Technology*, vol. 4, no. 1, pp. 141–149, 2007.

[97] Y. Song, W. Chen, and H. Dai, “Fast convergent distributed cooperative learning algorithms over networks,” *Neurocomputing*, vol. 275, pp. 2191–2199, 2018.

[98] Y. Yang, “Multi-tier computing networks for intelligent iot,” *Nature Electronics*, vol. 2, no. 1, p. 4, 2019.

[99] A. V. Dastjerdi and R. Buyya, “Fog computing: Helping the internet of things realize its potential,” *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[100] J. Liang, K. Li, C. Liu, and K. Li, “Joint offloading and scheduling decisions for dag applications in mobile edge computing,” *Neurocomputing*, vol. 424, pp. 160–171, 2021.

[101] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2017, pp. 1175–1191.

[102] A. Kalantari, A. Kamsin, S. Shamshirband, A. Gani, H. Alinejad-Rokny, and A. T. Chronopoulos, “Computational intelligence approaches for classification

of medical data: State-of-the-art, future challenges and research directions," *Neurocomputing*, vol. 276, pp. 2–22, 2018.

[103] W. Dai, M. Qiu, L. Qiu, L. Chen, and A. Wu, "Who moved my data? privacy protection in smartphones," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 20–25, 2017.

[104] L. Shao, D. Wu, and X. Li, "Learning deep and wide: A spectral method for learning deep networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 12, pp. 2303–2308, 2014.

[105] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, 2014.

[106] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1996–2018, 2014.

[107] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server." in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, vol. 14, Oct 2014, pp. 583–598.

[108] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[109] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Advances in Neural Information Processing Systems*, 2015, pp. 2737–2745.

[110] A. T. Suresh, F. X. Yu, S. Kumar, and H. B. McMahan, "Distributed mean estimation with limited communication," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, Aug 2017, pp. 3329–3337.

[111] G. Li, J. Wang, J. Wu, and J. Song, "Data processing delay optimization in mobile edge computing," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[112] F. Xiao, "Multi-sensor data fusion based on the belief divergence measure of evidences and the belief entropy," *Information Fusion*, vol. 46, pp. 23–32, 2019.

[113] X. Li, X. Liu, and L. Motiwalla, "Valuing personal data with privacy consideration," *Decision Sciences*, 2020.

[114] K. D. Martin and P. E. Murphy, "The role of data privacy in marketing," *Journal of the Academy of Marketing Science*, vol. 45, no. 2, pp. 135–155, 2017.

[115] M. S. Ali, K. Dolui, and F. Antonelli, "IoT data privacy via blockchains and ipfs," in *Proceedings of the seventh international conference on the internet of things*, 2017, pp. 1–7.

[116] S. S. Du, X. Zhai, B. Poczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," *arXiv preprint arXiv:1810.02054*, 2018.

[117] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," *arXiv preprint arXiv:1811.03804*, 2018.

[118] Y. Li and Y. Liang, "Learning overparameterized neural networks via stochastic gradient descent on structured data," in *Advances in Neural Information Processing Systems*, 2018, pp. 8168–8177.

[119] G. Lan, S. Lee, and Y. Zhou, "Communication-efficient algorithms for decentralized and stochastic optimization," *arXiv preprint arXiv:1701.03961*, 2017.

[120] B. Sirb and X. Ye, "Consensus optimization with delayed and stochastic gradients on decentralized networks," in *Big Data (Big Data), 2016 IEEE International Conference on.* IEEE, Dec 2016, pp. 76–85.

[121] G. Lan and Y. Zhou, "Asynchronous decentralized accelerated stochastic gradient descent," *arXiv preprint arXiv:1809.09258*, 2018.

[122] Q. Ma, F. L. Lewis, and S. Xu, "Cooperative containment of discrete-time linear multi-agent systems," *International Journal of Robust and Nonlinear Control*, vol. 25, no. 7, pp. 1007–1018, 2015.

[123] Z. Li and Z. Duan, *Cooperative Control of Multi-agent Systems: A Consensus Region Approach.* CRC Press, 2014.

[124] Z. Ding, "Consensus control of a class of lipschitz nonlinear systems," *International Journal of Control*, vol. 87, no. 11, pp. 2372–2382, 2014.

[125] W. Ren, R. W. Beard, and D. B. Kingston, "Multi-agent kalman consensus with relative uncertainty," in *American Control Conference, 2005. Proceedings of the 2005.* IEEE, Jun 2005, pp. 1865–1870.

[126] H. Jiang, Q. Bi, and S. Zheng, "Impulsive consensus in directed networks of identical nonlinear oscillators with switching topologies," *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, no. 1, pp. 378–387, 2012.

[127] A. Y. Ng, "Feature selection, L1 vs. L2 regularization, and rotational invariance," in *Proceedings of the twenty-first International Conference on Machine Learning.* ACM, Jun 2004, p. 78.

[128] Y. Gong, J. Li, Y. Zhou, Y. Li, H. S. Chung, Y. Shi, and J. Zhang, "Genetic learning particle swarm optimization," *IEEE Transactions on Cybernetics*, vol. 46, no. 10, pp. 2277–2290, 2016.

[129] X. Zeng, W. Wang, C. Chen, and G. G. Yen, "A consensus community-based particle swarm optimization for dynamic community detection," *IEEE Transactions on Cybernetics*, vol. 50, no. 6, pp. 2502–2513, 2019.

[130] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1223–1231.

[131] Y. Ming, Y. Zhao, C. Wu, K. Li, and J. Yin, "Distributed and asynchronous stochastic gradient descent with variance reduction," *Neurocomputing*, vol. 281, pp. 27–36, 2018.

[132] N. L. Roux, M. Schmidt, and F. R. Bach, "A stochastic gradient method with an exponential convergence _rate for finite training sets," in *Advances in Neural Information Processing Systems*, 2012, pp. 2663–2671.

[133] S. Shalev-Shwartz and T. Zhang, "Stochastic dual coordinate ascent methods for regularized loss minimization," *Journal of Machine Learning Research*, vol. 14, no. Feb, pp. 567–599, 2013.

[134] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, 2013, pp. 315–323.

[135] S. Zhao and W. Li, "Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[136] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. J. Smola, "On variance reduction in stochastic gradient descent and its asynchronous variants," in *Advances in Neural Information Processing Systems*, 2015, pp. 2647–2655.

[137] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[138] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010.* France, Springer, 2010, pp. 177–186.

[139] K. Buza, "Feedback prediction for blogs," in *Data analysis, machine learning and knowledge discovery.* Switzerland, Springer, 2014, pp. 145–152.

[140] J. Zeng and W. Yin, "On nonconvex decentralized gradient descent," *IEEE Transactions on Signal Processing*, vol. 66, no. 11, pp. 2834–2848, 2018.

[141] J. Wu, W. Huang, J. Huang, and T. Zhang, "Error compensated quantized sgd and its applications to large-scale distributed optimization," *arXiv preprint arXiv:1806.08054*, 2018.

[142] H. Tang, X. Lian, T. Zhang, and J. Liu, “Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression,” *arXiv preprint arXiv:1905.05957*, 2019.

[143] L. Deng, “The mnist database of handwritten digit images for machine learning research [best of the web],” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[144] B. Xiao, Y. Wei, X. Bi, W. Li, and J. Ma, “Image splicing forgery detection combining coarse to refined convolutional neural network and adaptive clustering,” *Information Sciences*, vol. 511, pp. 172–191, 2020.

[145] M. Whaiduzzaman, M. Sookhak, A. Gani, and R. Buyya, “A survey on vehicular cloud computing,” *Journal of Network and Computer applications*, vol. 40, pp. 325–344, 2014.

[146] Y. Wang, S. Zhao, R. Zhang, X. Cheng, and L. Yang, “Multi-vehicle collaborative learning for trajectory prediction with spatio-temporal tensor fusion,” *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2020.

[147] G. Loukas, T. Vuong, R. Heartfield, G. Sakellari, Y. Yoon, and D. Gan, “Cloud-based cyber-physical intrusion detection for vehicles using deep learning,” *IEEE Access*, vol. 6, pp. 3491–3508, 2017.

[148] B. Cao, Y. Li, L. Zhang, L. Zhang, S. Mumtaz, Z. Zhou, and M. Peng, “When internet of things meets blockchain: Challenges in distributed consensus,” *IEEE Network*, vol. 33, no. 6, pp. 133–139, 2019.

[149] H. Dai, Z. Zheng, and Y. Zhang, “Blockchain for internet of things: A survey,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8076–8094, 2019.

[150] X. Zheng, M. Li, Y. Chen, J. Guo, M. Alam, and W. Hu, “Blockchain-based secure computation offloading in vehicular networks,” *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–15, 2020.

[151] N. Z. Aitzhan and D. Svetinovic, “Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging

streams," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 5, pp. 840–852, 2016.

[152] M. Sumagita, I. Riadi, J. P. D. S. Sh, and U. Warungboto, "Analysis of secure hash algorithm (SHA) 512 for encryption process on web based application," *International Journal of Cyber-Security and Digital Forensics (IJCSDF)*, vol. 7, no. 4, pp. 373–381, 2018.

[153] Y. Wanjun and W. Yuan, "Research on network trading system using blockchain technology," in *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, vol. 3. IEEE, 2018, pp. 93–97.

[154] T. Duong, L. Fan, J. Katz, P. Thai, and H. Zhou, "2-hop blockchain: Combining proof-of-work and proof-of-stake securely," in *European Symposium on Research in Computer Security.* Springer, 2020, pp. 697–712.

[155] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[156] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT press, 2018.

[157] J. Zhou, H. Qian, X. Lu, Z. Duan, H. Huang, and Z. Shao, "Polynomial activation neural networks: Modeling, stability analysis and coverage BP-training," *Neurocomputing*, vol. 359, pp. 227–240, 2019.

[158] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[159] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[160] H. Y. Ong, K. Chavez, and A. Hong, "Distributed deep q-learning," *arXiv preprint arXiv:1508.04186*, 2015.

[161] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[162] H. Wei, G. Zheng, H. Yao, and Z. Li, "Intellilight: A reinforcement learning approach for intelligent traffic light control," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2496–2505.

[163] P. Koonce and L. Rodegerdts, "Traffic signal timing manual." United States. Federal Highway Administration, Tech. Rep., 2008.

[164] Y. Wang, X. Yang, H. Liang, and Y. Liu, "A review of the self-adaptive traffic signal control system based on future traffic environment," *Journal of Advanced Transportation*, vol. 2018, pp. 1–12, 2018.

[165] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 8050–8062, 2019.

[166] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[167] I. Haque, M. Nurujjaman, J. Harms, and N. Abu-Ghazaleh, "Sdsense: An agile and flexible sdn-based framework for wireless sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1866–1876, 2019.

[168] D. Garg, M. Chli, and G. Vogiatzis, "Deep reinforcement learning for autonomous traffic light control," in *2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*. IEEE, 2018, pp. 214–218.

[169] J. Zeng, J. Hu, and Y. Zhang, "Adaptive traffic signal control with deep recurrent q-learning," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2018, pp. 1215–1220.

[170] X. Liang, X. Du, G. Wang, and Z. Han, “A deep reinforcement learning network for traffic light cycle control,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1243–1253, 2019.

[171] T. Ming, W. Fang, C. Peng, C. Cai, R. K. De Richter, M. H. Ahmadi, and Y. Wen, “Impacts of traffic tidal flow on pollutant dispersion in a non-uniform urban street canyon,” *Atmosphere*, vol. 9, no. 3, p. 82, 2018.

[172] Y. Liu, L. Liu, and W.-P. Chen, “Intelligent traffic light control using distributed multi-agent q learning,” in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, Oct. 2017, pp. 1–8.

[173] H. Ge, Y. Song, C. Wu, J. Ren, and G. Tan, “Cooperative deep q-learning with q-value transfer for multi-intersection signal control,” *IEEE Access*, vol. 7, pp. 40 797–40 809, 2019.

[174] T. Chu, J. Wang, L. Codecà, and Z. Li, “Multi-agent deep reinforcement learning for large-scale traffic signal control,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1086–1095, 2019.

[175] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel, “A unified game-theoretic approach to multiagent reinforcement learning,” in *Advances in Neural Information Processing Systems*, Dec. 2017, pp. 4190–4203.

[176] Z. Wang, “The applications of deep learning on traffic identification,” *BlackHat USA*, vol. 24, no. 11, pp. 1–10, 2015.

[177] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, “Advances and open problems in federated learning,” *arXiv preprint arXiv:1912.04977*, 2019.

[178] V. Builenko, A. Pakhomova, and S. Pakhomov, “Optimization of the method for collecting source data to calculate the length of the traffic light control cycle,” *Transportation research procedia*, vol. 36, pp. 90–94, 2018.

[179] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, Dec. 2012, pp. 1097–1105.

[180] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, 2012.

[181] C. Fang, H. Dong, and T. Zhang, "Mathematical models of overparameterized neural networks," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 683–703, 2021.

[182] D. Wang and M. Li, "Stochastic configuration networks: Fundamentals and algorithms," *IEEE Transactions on Cybernetics*, vol. 47, no. 10, pp. 3466–3479, 2017.