Byzantine Resilient Consensus, Learning, and Optimization in Distributed Multi-Agent Systems

By

Jiani Li

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

May 31, 2021

Nashville, Tennessee

Approved:

Xenofon Koutsoukos, Ph.D.

Abhishek Dubey, Ph.D.

D. Mitchell Wilkes, Ph.D.

Gautam Biswas, Ph.D.

Waseem Abbas, Ph.D.

To my grandparents.

Acknowledgments

My deep gratitude goes first to my PhD advisor, Dr. Xenofon Koutsoukos, who supported my study and research for the last five years. The accomplishment of this thesis would not have been possible without his motivation, expertise, and guidance.

Besides my advisor, I would like to thank Dr. Gautam Biswas, Dr. Abhishek Dubey, Dr. D. Mitchell Wilkes, and Dr. Waseem Abbas, for being members of my thesis committee. In addition, I would like to thank my coauthors for the help and insights they provided when conducting research together. In particular, I would like to thank Dr. Waseem Abbas, Feiyang Cai, Chandreyee Bhowmick, and Weihan Wang. My sincere thank also goes to my other colleagues at Vanderbilt University and Institute for Software Integrated Systems for their insightful comments and useful feedback during group meetings.

Finally, thank you Weihan, for taking risks to travel back to the U.S. during Covid-19, for the patience and company, and for bringing me love and happiness.

Table of Contents

List of Tables

List of Figures

$\mathbb{R}$	Field of real numbers.
$\mathbb{1}$	Column vector with all its entries equal to one.
I_M	Identity matrix of size $M \times M$.
$\lvert x \rvert$	Absolute value of a real number x or cardinality of a set x.
$\lVert x \rVert$	Euclidean norm or ℓ_2-norm of a vector x.
$\mathbb{E}[\cdot \lvert \xi]$	Expected value of a random variable ξ; $\mathbb{E}[\cdot]$ is used when context is clear.
$\binom{n}{m}$	The number of possible combinations of m objects from a set of n objects.
$[n]$	Set $\{1, 2, \ldots, n\}$.
$[a_{lk}]$	A matrix with (l, k)-th entry a_{lk}.
$\mathrm{col}\{\mathrm{a}, \mathrm{b}\}$	Column vector with entries a and b.
$\mathrm{diag}\{\mathrm{a}, \mathrm{b}\}$	Diagonal matrix with entries a and b.
$X \cap Y$	Intersection of sets X and Y.
$X \backslash Y$	Relative complement of set Y in set X.
$(M)^\top$	Transpose of matrix M.
$\mathrm{Tr}(M)$	Trace of matrix M.
$(M)^{-1}$	Inverse of matrix M.
$A \bigotimes B$	Kronecker product of matrices A and B.
$\nabla_x F$	Gradient vector of function F relative to x.
$\alpha = O(\mu)$	$\lvert\alpha\rvert \leqslant c \lvert\mu\rvert$ for some constant $c > 0$.
$\limsup_{n \to \infty} a(n)$	Limit superior of the sequence $a(n)$.

Chapter 1

Introduction

1.1 Motivation

With the ever-growing technological explosion of the world, distributed systems are becoming more and more widespread and spawning numerous applications in multi-agent systems including sensor networks, cloud computing, swarm robotics, and intelligent systems [10–14]. In particular, distributed learning and optimization techniques have attracted increasing attention due to the rapid growth of machine learning applications in multi-agent networks, such as mobile phones, wearable devices, and smart homes [15]. In such methods, multiple agents operate in a distributed and cooperative manner to achieve a common learning task. Typically, agents communicate their model parameters with their local neighbors without the sharing of user data. Such cooperation has been demonstrated to help improve the learning performance over the network while not compromising data privacy [16]. Consider a concrete example of learning the behavior of users in a cellular network based on data generated using various mobile applications. Each user may generate data that follows a distinct distribution and it is common to learn separate models for each user. However, people may exhibit similar behaviors and relatedness among models commonly exists [15]. Hence, cooperation among agents could be leveraged to promote the learning performance over the network. Compared to a centralized approach, distributed methods offer multiple advantages that include robustness to drifts in the statistical properties of the data, scalability, relying on local data and fast response among others.

Although cooperation among agents benefits the operation of the system, it is also susceptible to attacks where non-cooperative or adversarial neighbors sharing wrong information can disrupt the normal operation of the entire network. In particular, it has been shown that cooperation through averaging is not resilient to even one non-cooperative agent. For example, in both distributed consensus and distributed learning systems, a single non-cooperative agent sending a constant value to its neighbors can lead the entire network to converge to this value being transmitted by the non-cooperative agent [2, 17, 18]. Multiple detection approaches based on a statistical analysis of normal states have been presented in the literature [19–21] to cope with attacks. However, detection methods are susceptible to false alarms and the attacker can always change its attacking strategy to adapt to such detection rules, thus making these methods not able to provide strong resilience guarantees. As a result, it is important to come up with resilient cooperation mechanisms that can eliminate the possible damage caused by non-cooperative agents and ensure the resilience of the

information aggregation in distributed systems.

1.2 Challenges

The notion of resilience is broad and relates to the concepts of fault-tolerance, reliability, safety, and robustness. In most of the cases, it refers to the ability of the system to withstand adversaries and bounce back to a state of normal operation. A typical practice in the literature dealing with the design and analysis of resilient distributed algorithms is to obtain bounds on the number of adversarial agents in the network and achieve the desired objective as long as the number of adversaries in the network is less than the obtained bounds. Such a setup poses some limitation as we need to assume that the number of adversarial agents cannot be greater than the values indicated in the bounds. However, despite this limitation, these bounds could be useful as they provide insight into the resilience or the ability of the network to perform normally even in the presence of adversarial agents. On the other hand, it is indeed a challenge to develop methods and techniques that do not rely on the knowledge and assumption on the maximum number of adversaries in the network to achieve the desired performance in a resilient manner.

The resilience of distributed consensus, learning and optimization algorithms has been widely studied in the literature. It has been well established that methods such as using the Tverberg partitions and the safe area can achieve resilient vector consensus, yet with a high computational time complexity growing exponentially in the dimension of the states [22, 23]. When applying resilient consensus algorithms into distributed learning and optimization, one immediately faces the dilemma that 1) resilient scalar consensus algorithms cannot guarantee resilient vector consensus for high-dimensional model parameters, and 2) it is not practical to use resilient vector consensus algorithms given its exponential computational complexity in the dimension. As a compromise, resilient consensus algorithms for scalar states are frequently used for achieving resilient aggregation in distributed learning and optimization with high-dimensional model parameters, sacrificing the strong resilience to adversaries. Hence, there is a major challenge in balancing the trade-off between the resilience and the computational complexity in achieving resilient aggregation in distributed learning and optimization algorithms.

Finally, to design resilient consensus, learning and optimization algorithms in distributed multi-agent systems and analyze their performance, realistic attack methods being used to model the damage of cyber-attacks on the system are needed. Therefore, it poses a challenge in obtaining the attack strategy that is not restrictive and can adequately represent the worst-case attack one could realize in the real world to examine the resilience of the system.

1.3 Contributions

The focus of our contributions is on carefully examining the vulnerabilities of a variety of distributed consensus, learning, and optimization systems, and designing resilient cooperation mechanisms in such distributed systems to eliminate the damage caused by non-cooperative or adversarial agents. Our results are supported by theoretical analysis as well as numerical implementations. Codes for the numerical implementations can be found at https://github.com/JianiLi. The detailed contributions of this thesis are listed below.

Chapter 3

In this chapter, we study the resilient vector consensus problem and a recently proposed solution referred to as the Approximate Distributed Robust Convergence (ADRC) algorithm. The main contributions are:

- We show that the resilience of ADRC algorithm can be improved by using the notion of centerpoint instead of Tverberg partition. We discuss these improvements in two, three and higher dimensions separately.
- Using centerpoints, we show that $|\mathcal{N}_i| \geq n_{f_i}(d+1)+1$ is not only sufficient but also necessary to compute a safe point, which is a key step in the ADRC algorithm. Here n_{f_i} is the number of adversaries in the neighborhood of a normal node i. We also provide an overview of various algorithms reported in the literature to compute centerpoints in different dimensions.
- We compare and numerically evaluate our results with the existing algorithm by simulating resilient vector consensus in multirobot networks.

Chapter 4

In this chapter, we study the problem of the resilient convergence of distributed learning algorithms for multi-robot systems in the presence of Byzantine adversaries. We make the following contributions:

- We analyze the sufficient condition to achieve Byzantine resilient convergence of distributed learning algorithms, which is to guarantee that the aggregated state lies inside the convex hull of the normal states. When the sufficient condition is satisfied, we show that normal agents are guaranteed to converge towards the global optimum state with $O(1/i)$ convergence rate using appropriate stepsize, where i is the time iteration.
- We propose a centerpoint-based aggregation rule and show that it guarantees the resilient convergence of the distributed learning algorithms whenever each normal agent k in the network has $f \leq \lceil \frac{n_k}{d+1} \rceil - 1$ Byzantine neighbors.

- We evaluate our results using the examples of target pursuit and pattern recognition in multi-robot systems. We compare the proposed centerpoint-based aggregation rule with the average, coordinate-wise median, and geometric median-based rules. The simulation results show that our approach is resilient to $\lceil \frac{n_k}{d+1} \rceil - 1$ Byzantine neighbors, and the cooperation improves the average learning performance over the network than the non-cooperative case, while the other approaches are not resilient in the same scenarios.

Chapter 5

In this chapter, we study the resilient distributed diffusion Least-Mean-Square (LMS) problem in the multi-task networks. Specifically, we make the following contributions:

- By exploiting the adaptive weights that aim to cope with fixed-value Byzantine attacks, we develop time-dependent attack models that drive normal agents to converge to states selected by an attacker. The attack models can be used to deceive a specific node or the entire network and are applicable to both stationary and non-stationary state estimation. Although the attack models are based on a strong knowledge of the system, we show the attack can also succeed without such knowledge to demonstrate the practicality of the proposed attack. This for the first time dispels the myth that baseline diffusion multi-task LMS by itself is resilient to adversarial attack.

- We develop a resilient distributed diffusion algorithm that can achieve resilience to up to F compromised nodes in one normal agent's neighborhood. By selecting an appropriate F, the proposed algorithm guarantees normal agents will converge to their actual targets under any data falsification attacks. If the parameter F selected by the normal agents is large, the resilient distributed diffusion algorithm degenerates to non-cooperative estimation. Thus, we also analyze trade-off between the resilience of distributed diffusion and its performance degradation in terms of MSD.

- We evaluate the proposed attack models and the resilient estimation algorithm using both stationary and non-stationary multi-target localization. The simulation results are consistent with our theoretical analysis and show that the approach provides resilience to attacks while incurring performance degradation which depends on the assumption about the number of compromised nodes.

Chapter 6

In this chapter, we extend the work in Chapter 5 and study the resilient distributed multi-task learning problem for general convex loss functions. In particular, we make the following contributions:

- We propose an efficient Byzantine resilient online weight adjustment rule for distributed multi-task learning. We measure similarities among agents based on the accumulated loss of an agent's data and the models of its neighbors. In each iteration, a normal agent computes the weights assigned to its neighbors in time that is linear in the size of its neighborhood and the dimension of the data.
- We show that aggregation with the proposed weight assignment rule always results in an improved expected regret than the non-cooperative case, and normal agents converge resiliently towards the global minimum. Even when all the neighbors are Byzantine, a normal agent can still resiliently converge to the global minimum bounded by the same expected regret as without any cooperation with other agents, achieving resilience to an arbitrary number of Byzantine agents.
- We conduct three experiments for both regression and classification problems and demonstrate that our approach yields good empirical performance for non-convex models, such as convolutional neural networks.

Chapter 7

In this chapter, we extend the work in Chapter 6 and study the distributed clustered multi-task learning problem. In particular, we make the following contributions:

- We propose an adaptive clustering method in distributed multi-task learning networks that allows agents to perform the optimization task while simultaneously learn which neighbors are suitable for cooperation.
- We analyze the convergence and learning performance for the proposed method.
- We propose the optimal combination weights for aggregating the neighbors' model parameters after recognizing which neighbors to cooperate with using the proposed clustering method that optimize the network learning performance.
- We evaluate the proposed method for both linear regression and classification problems and compare the results with existing distributed clustering methods. The evaluation results show that the proposed method significantly improves the learning performance compared to the non-cooperative approach by correctly estimating the clustering structure. In contrast, other methods fail to achieve the clustering information and exhibit inferior learning performance.

Chapter 8

In this chapter, we study the resilient distributed reinforcement learning problem in the presence of Byzantine agents. In particular, we make the following contributions:

- We propose a Byzantine resilient aggregation method for distributed reinforcement learning algorithms and analyze the convergence of RL algorithms towards the optimal solution in the presence of an arbitrary number of Byzantine agents when linear approximations are applied.
- We evaluate the proposed method using multiple RL tasks for both value-based and policy-based RL with non-linear function approximations, such as Deep Q-learning and Deep Deterministic Policy Gradient (DDPG). The evaluation results show that the proposed method exhibits better or similar learning performance (measured by the accumulated reward over the network) than no-cooperation with or without the presence of Byzantine agents.

Chapter 9

In this chapter, we extend the work in Chapter 8 and propose efficient adaptive learning methods for distributed actor-critic algorithms that have linear time complexity and are resilient in the presence of attacks. We make the following contributions:

- We propose efficient adaptive learning methods for distributed actor-critic algorithms that allow agents to learn which neighbors to cooperate with in a fully-decentralized network.
- We analyze the convergence of distributed actor-critic algorithms with the proposed method when linear function approximations are applied.
- We evaluate the proposed method for multiple actor-critic algorithms including Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC), Twin Delayed DDPG (TD3). The evaluation results show that the proposed method greatly improves the learning performance than the non-cooperative case in all the scenarios. Besides, when all the agents share the common learning task, the efficiency and performance of the proposed method matches that of averaging method. Moreover, the proposed method results in superior performance in multi-task networks and in the presence of attacked agents – it is resilient even when all one's neighbors are attacked, which matches the performance of the non-cooperative case.

1.4 Organization

The technical contributions of this thesis are in Chapter 3-9. This thesis is organized as follows.

- Chapter 2 reviews the related work in resilient distributed consensus, learning, and optimization algorithms.
- Chapter 3 studies the problem of resilient approximate vector consensus.

- Chapter 4 applies the results in Chapter 3 to distributed learning algorithms for resilient convergence of networked agents in the learning systems.
- Chapter 5 studies the resilient distributed diffusion LMS problem in multi-task networks.
- Chapter 6 generalizes the results of Chapter 5 to general convex loss functions for resilient distributed multi-task learning.
- Chapter 7 extends the results of Chapter 6 to the distributed clustered multi-task learning problem.
- Chapter 8 studies the resilient distributed reinforcement learning problem.
- Chapter 9 proposes the adaptive learning methods for distributed actor-critic algorithms that enable networked agents to learn resiliently in the presence of attacks.
- Chapter 10 concludes this thesis.

1.5 Main Concepts

We introduce the following concepts used throughout this thesis.

- **Capacity and knowledge of normal agents.** *Normal* agents are the ones that always update their states (estimates) according to a prescribed updating rule. For a normal agent k, all agents in its neighborhood are indistinguishable, that is, k cannot identify which of its neighbors are adversaries.
- **Synchronicity of the system.** We consider two types of systems: synchronous and asynchronous. In *synchronous* systems, there is a global clock assumed to produce time reference for all the agents. And it is assumed that every agent starts and operates simultaneously. In *asynchronous* systems, every agent operates according to its internal clock. Clocks may arbitrarily differ from each other.
- **Distributed systems.** Generally, two types of distributed systems are considered in the literature. One is composed of a central server and multiple distributed workers and the workers communicate only with the central node. The other is the peer-to-peer system where agents are fully decentralized and they communicate with their one-hop neighbors. Unless otherwise noted, the second model is considered in this thesis.
- **Scope of threats.** Other than the normal agents, agents that could be subverted and controlled by an adversary is usually referred to as non-cooperative or adversarial agents. The scope of threats confines the behavior of the adversary. There are several threat models considered in the literature including the crash [24], non-colluding [19], malicious [17, 19, 25] and Byzantine [18, 22, 26–29] nodes. Crash nodes are

those could stop their updates and possibly communication. Non-colluding nodes do not know the network topology, the identity of other misbehaving nodes, or the states of non-neighboring nodes. Malicious nodes, however, have full knowledge of the networked system. Byzantine nodes differ from malicious nodes in the way that it can transmit different information to different neighbors, whereas malicious nodes must transmit the same information to each neighbor. Byzantine nodes are the most powerful threats that are usually assumed in the literature and is of particular interest in this thesis.

- **Effect of attacks.** False data injection attacks are usually employed to result in the following outcomes on the distributed consensus, learning and optimization networks: 1) preventing the network from reaching a consensus, 2) slowing down the convergence of the network, and 3) making the network converges to a wrong state.

Chapter 2

Related Work

In this chapter, we review the related works of distributed consensus and learning algorithms in cooperative multi-agent systems, with a special focus on the vulnerabilities, attacks and resilience design of the cooperation in such systems. We begin with the problem of resilient distributed (approximate) consensus in multi-agent systems in Section 2.1. In Section 2.2, we review the resilient distributed learning and optimization problems. Then, in Section 2.3, we discuss the problem of distributed reinforcement learning (RL) and review existing work on resilient distributed RL. Finally, in Section 2.4, we make a comparison between our contributions to the existing work in the literature.

2.1 Resilient Distributed Approximate Consensus

A fundamental problem in distributed systems is to have networked agents reach a state of consensus, where the state can be either a scalar or a vector. The resilient distributed (approximate) consensus problem initiated in [30] addresses such a problem when the states are composed of reals and is widely studied in the robotics and control systems community [17, 22, 28, 31–34]. The main objective is to ensure that all normal agents in a network satisfy the *safety* and *agreement* conditions in the presence of Byzantine agents [22, 28] by exchanging and updating their states via local interaction. The safety condition requires normal agents to update their states such that they are always inside the convex hull of normal agents' initial states. Agreement means that eventually, all normal agents' states are very close to each other, that is, within an arbitrary $\epsilon > 0$ distance from one another. Although it is often assumed that agents can exchange messages with all of their one-hop neighbors, the gossip-based randomized consensus algorithms are also considered in the literature where an agent chosen randomly wakes up, contacts a neighbor randomly within its connectivity radius, and exchanges the state variable for an update of the state [35]. In this thesis, we focus on the first cooperation mechanism where agents exchange information with all of their neighbors. In the following, we first discuss the studies on resilient distributed consensus over scalar states, followed by the related work on resilient distributed consensus over vector states.

2.1.1 Resilient Distributed Consensus over Scalar States

For scalar sates, the goal is to guarantee the convergence of normal agents to a common state $\bar{x} \in [x_{\min}(0),\, x_{\max}(0)]$ where $x_{\min}(0)$ and $x_{\max}(0)$ are the minimum and maximum of the initial values of normal nodes respectively. The Weighted-Mean Subsequence Reduced (W-MSR) algorithm proposed in [17] guarantees convergence in the presence of adversaries under certain robustness conditions on the underlying network graph. It is assumed in W-MSR that at most f of a normal agent's neighbors may be misbehaving. To achieve resilient consensus, each agent removes the f smallest and f largest extreme values with respect to its own value when updating its state. After removing the extreme values, the normal agent uses the average of the remaining values to be its updated state. This method requires an assumption of the number f of Byzantine agents. And by increasing f, the W-MSR algorithm is resilient to more Byzantine agents. The maximum value of $f = \lceil \frac{n}{2} \rceil - 1$, where n is the total number of states to be aggregated with. In other words, when setting $f = \lceil \frac{n}{2} \rceil - 1$, the algorithm can be resilient when less than half of the total number of agents are Byzantine. Different variations of the W-MSR algorithms for scalar consensus have also been proposed [36–38]. Further, the median-based consensus protocol was proposed [31, 39, 40] such that instead of updating the state as an (weighted) average of neighbors' states, it uses the median of the neighbors' states to be the aggregated state. Compared to the average value, the median is inherently robust to the presence of outliers as a statistical measure. The nice robustness property the median has guarantees that as long as more than $\lceil \frac{n}{2} \rceil$ points are in $[-r, r]$ for $r \in \mathbb{R}$, then the median must be in $[-r, r]$ no matter where the other points locate. This ensures that as long as the majority of the points are benign, the median is sure to be inside the bounds of the benign points. The median-based updating rule is resilient to up to $\lceil \frac{n}{2} \rceil - 1$ Byzantine nodes. Note that by setting $f = \lceil \frac{n}{2} \rceil - 1$, the W-MSR algorithm is equivalent to the median.

Resilience can also be achieved via fault detection and isolation (FDI). For instance, [19] studies the FDI problem for linear consensus networks via high connectivity networks and global knowledge of the network structure by each agent. [20] considers a similar FDI problem for second-order systems. Authors in [21] presents a distributed detection method for consensus+innovation algorithms via local observations of agents only. However, detection approaches are usually deficient in providing strong resilience guarantees given that they are susceptible to false alarms.

2.1.2 Resilient Distributed Consensus over Vector States

The problem is more challenging when the state vectors are in $\mathbb{R}^d$ where $d \geq 2$. And the objective of resilient consensus for vector states is to ensure that normal agents converge at some point in the convex hull of their initial states. A simple approach could be to run d instances of resilient consensus algorithms over

scalar states (such as the W-MSR or median-based updating rule), one for each dimension. For instance, the approach of running the W-MSR algorithm multiple times for d dimensions has been applied to mobile multi-robot systems for resilient formation control [41] and resilient flocking [42]. However, as a result of this approach, normal agents might converge at a point outside of the convex hull of their initial states, as discussed in [22]. The trade-off between resilience and accuracy has been studied in [6]. Resilient vector consensus based on Tverberg partition has been proposed in recent years that guarantees the safety and agreement conditions given the number of Byzantine agents is bounded by a fraction of the total number of agents [22, 33]. The idea is to partition the points into a series of point sets (Tverberg partitions) each containing $n - f$ points, where n is the total number of points and f is the upper bound of adversarial points. Then, there must be at least one partition containing only the normal points since at most f points can be adversarial. As a result, the intersection of the partitions must lies inside the convex hull of the normal points. And agents move towards somewhere inside the intersection of the Tverberg partitions thus ensuring the safety condition. It should be noted that the computational complexity for computing a Tverberg point (a point inside the intersection of the Tverberg partitions) is high, especially for large dimensions, which requires $O(n^d)$ where d is the dimension [43]. Further, a similar idea based on the concept of "safe area" that is the intersection of any partition of size $n - f$ was proposed in [28]. The safe area can be can computed in $O(n^d)$ time using linear programming since the safe area is the intersection of $O(n^d)$ half-spaces. Notably, there exists a trade-off between resilience and computational time. A recent work [44] provides a more efficient algorithm following the idea of Tverberg partition with an assumption of a set of known normal agents and an upper bound of the number of Byzantine agents. Moreover, an approximate solution for computing an approximate Tverberg point was used in [32] that reduces the computational time from $O(n^{d-1})$ to $O(n)$, yet also reducing the resilience bound from $\lceil \frac{n}{d+1} \rceil - 1$ to $\lceil \frac{n}{2^d} \rceil - 1$.

A summary of the resilient distributed approximate consensus algorithms we review is given in Table 2.1. And Table 2.2 provides a summary of the resilient consensus algorithms.

Table 2.1: Resilient distributed approximate consensus algorithms.

Algorithm	State Type	Resilience Condition	Computational Complexity
W-MSR [17]	Scalar	$n \geq 2f + 1$	$O(nd)$
Median [31]	Scalar	$n \geq 2f + 1$	$O(nd)$
Tverberg point [22]	Vector	$n \geq (d + 1)f + 1$	$O(n^d)$
Safe area [23]	Vector	$n \geq (d + 1)f + 1$	$O(n^d)$
Approximate Tverberg point [32]	Vector	$n \geq 2^d f + 1$	$O(nd)$

Table 2.2: Summary of related work on resilient consensus.

	Communication Protocol		State Type		Exchanging Messages With	
Paper	Synchronous	Asynchronous	Scalar	Vector	All Neighbors	Randomized Neighbor
[17, 31, 36–40]	✓	-	✓	-	✓	-
[32, 41, 42]	✓	-	-	✓	✓	-
[22]	✓	✓	-	✓	✓	-
[33, 44]	✓	-	-	✓	✓	-
[28]	-	✓	-	✓	✓	-
[45]	-	✓	-	✓	-	✓
[46, 47]	-	✓	✓	-	-	✓

2.2 Resilient Distributed Learning and Optimization

Distributed learning and optimization has attracted increasing attention due to the growth of machine learning (ML) applications in distributed devices within multi-agent networks, such as mobile phones, wearable devices, and smart homes [48–51]. In such networks, multiple agents operate in a distributed and cooperative manner to achieve a learning task. For example, consider learning the behavior of users in a cellular network based on data generated using various mobile applications. Each user may generate data that follows a distinct distribution and it is common to learn separate models for each user. However, people may exhibit similar behaviors and similarities among models commonly exist [15]. In this case, cooperation among agents could be leveraged to promote the learning performance over the network.

Given data privacy concerns, cooperation among agents in a network relies typically on exchanging model parameters instead of data. In a distributed learning network, an agent communicates model parameters with its local neighbors and also updates these parameters by incorporating the neighbors' information [16]. A variety of frameworks has been proposed in the literature for distributed learning and optimization. For example, [18, 52–54] considers a network consisting of one central server with multiple distributed learners (agents) connected to the server. At each iteration, learners sample data and compute the gradient of the local empirical loss. And the parameter server collects and aggregates the gradients from the learners and updates the model that will be sent to the learners for the next iteration's gradient computation. In contrast, a fully-decentralized network was considered in [2, 55–57] such that each agent communicates only with their one-hop neighbors for information exchanging. Different underlying models being learned by the network are also studied in the literature. In most cases [52, 58–60], discriminative models, such as linear regression, SVM, or neural networks are used, whereas a generative model based on Bayesian learning is considered in [61]. Besides, there are different messages to be aggregated, including gradients [54, 59, 62–64] and model parameters [55–58]. In addition, it is usually assumed in the literature that every agent in the network is access

to data from the same distribution [18, 52–54], which is also referred to as the single-task model; however, there is a particular field in distributed learning [55, 56, 59, 60] adopts the multi-task setting where agents learn from distinct but related data distributions. Further, both synchronous and asynchronous communication among agents are considered in the literature [65–68].

Although cooperation among agents helps improve the overall learning performance [69], it is also susceptible to attacks where non-cooperative or adversarial neighbors sharing wrong information can disrupt the convergence of the algorithm. Average-based information aggregation rules have been widely used in distributed learning [69–71], yet it has been shown that a single misbehaving agent can adversely impact the convergence of the average-based aggregation rules [1, 72]. Therefore, it is crucial to design resilience in distributed learning that can ensure the convergence of normal agents towards their target models. The problem of achieving resilient cooperation in distributed learning networks is a hot topic in recent years and there are extensive works in this field. In general, approaches solving this problem fall into two categories. One is by resilient consensus algorithms that assumes a majority of the agents are normal and computes the resilient aggregated result as a linear combination of the normal messages. The other approach tries to measure the similarities among agents and aggregates the information by how much agents are related. While the second approach can be used in either the single-task or the multi-task setup, the consensus-based (majority-based) resilient aggregation rules are not directly applicable to the multi-task networks since each agent fits a distinct model and agents might not form a majority group in such a network. Later in this section, we will review some classic and widely used methods for the two major approaches.

2.2.1 Consensus-based Resilient Cooperation

The objective in resilient distributed learning is to ensure that every normal agent converges to its actual target model. In a typical setup, agents are deployed in a fully-decentralized network, and they solve a network optimization problem with access to local data sets and exchanging model parameters with neighbors [73]. To achieve resilient distributed learning, one needs to ensure that the aggregated model parameter is not influenced by any adversarial agent. One approach is to discard cooperation with possible Byzantine neighbors using the idea of trimming. In such an approach, it is assumed that a maximum of f Byzantine agents can be present in the neighborhood of a normal agent. Algorithms are then designed for a normal agent to rank its neighbors based on some trust criteria and a normal agent discards the values from its f least trusted neighbors. Such a setup poses some limitation as one needs to assume that the number of adversarial agents cannot be greater than the a given bound. Despite this limitation, these bounds could be useful as they provide insight into the resilience or the ability of the network to perform normally even in the presence of

adversarial agents. Various metrics have been proposed in the literature to evaluate a agent's trustworthiness, including metrics based on the product of the weight and the loss [1, 2], model parameters [57], gradients and their norms [54, 74], and a combination of gradient and model parameters [75]. For example, [57] proposes a trimming algorithm where agents reject received values that are too large or too small, which follows the idea of the W-MSR algorithm; and [54] presents a screening algorithm called Zeno that ranks the scores of the aggregated gradients as the measurements of their trustworthiness and discards f largest scores.

Moreover, various majority-based aggregation rules have been proposed, similar to the median-based approach used in resilient scalar consensus, that preclude states far away from the cluster of the normal agents' states to achieve the resilience of distributed learning. Well-known majority-based aggregation rules include the coordinate-wise median [52, 63, 64], coordinate-wise trimmed mean [52, 57], geometric median [58, 62], Krum and multi-Krum [72], Bulyan and multi-Bulyan [76, 77]. For instance, coordinate-wise median [52] computes the medians for each dimension of the aggregated parameters. It is designed to be resilient when less than half of the agents are adversarial. Similarly, coordinate-wise trimmed mean [52] follows the idea of the W-MSR algorithm that trims f largest and smallest values and computes the mean of the remaining values for each dimension, which is designed to be resilient to up to f adversarial agents. [62] uses the geometric median and approximate geometric median as the aggregation result, which is a generalization of the median in higher dimensions. The geometric median guarantees the aggregated state to lie inside the Euclidean ball of radius r blowed up by a constant factor only as long as there are the majority of points inside the Euclidean ball, thus guarantees that the geometric median is close to the majority of the normal points when less than half of the agents are adversarial. Similarly, [18] proposes a method called Krum that tries to preclude the vectors that are too far away. To find the points that are far away from the majority of the points, they compute the sum of the squared distances between each point and its $n - f - 2$ closest points, where n is the number of aggregated states and f is the upper bound of the quantity of the adversarial agents. It then computes the aggregation result to be the point that has the minimum value of the summed squared distance. This guarantees that all the points that are far from the cluster of the majority of the points are discarded. Krum is considered to be resilient if $n > 2f + 2$. A variant of Krum, namely multi-Krum, selects m points instead of only one point that score the best, and computes the average of these points as the aggregation result [18]. It can be easily found that the cases $m = 1$ and $m = n$ correspond to Krum and averaging respectively. When selecting $m = n - f$, multi-Krum achieves the same resilience as Krum does. Further, Bulyan and multi-Bulyan are proposed on top of Krum [76, 77]. Instead of ranking and filtering out the suspicious messages, these rules differ from the ranking methods in the way that they do not need the ranking step and the aggregation result directly precludes states far away from the cluster of the majority of the states by the intrinsic properties of the aggregation rules. However, studies have already

reported all of the above-mentioned methods are not resilient to attacks under certain conditions [78–80]. We summarize the consensus-based resilient distributed learning algorithms in Table 2.3.

Note that instead of manipulating the aggregation rules, one can also use computation redundancy to realize resilient convergence for distributed learning algorithms, which typically involves coding theory and algorithmic redundancy [81–83]. An example of such a framework is DRACO [81] in which the parameter server uses redundant gradients received from agents to eliminate the effects of adversarial updates. Another algorithm proposed recently is the RSA algorithm that introduces an ℓ_p-norm regularization term into the objective function for the resilience purpose [84]. It eliminates the effect caused by the magnitudes of malicious messages sent by the Byzantine agents, as a result, only the number of Byzantine agents, but not the magnitude, influence the model update, making the algorithm robust to large outliers.

Table 2.3: Consensus-based resilient aggregation rules for distributed learning algorithms.

Aggregation Algorithms	Resilience Condition	Computational Complexity
W-MSR [57]	$n \geq 2f+1$	$O(nd)$
Coordinate-Wise Trimmed Mean [52]	$n \geq 2f+1$	$O(nd)$
Coordinate-Wise Median [52, 63, 64]	$n \geq 2f+1$	$O(nd)$
Zeno [54]	$n \geq f+1$	$O(nd)$
Geometric Median [58, 62]	$n \geq 2f+1$	$O(n^2d)$
Krum [18]	$n \geq 2f+3$	$O(n^2d)$
Multi-Krum [18]	$n \geq 2f+3$	$O(n^2d)$
Bulyan [76]	$n \geq 4f+3$	$O(n^2d)$
Multi-Bulyan [77]	$n \geq 4f+3$	$O(n^2d)$

2.2.2 Distributed Multi-Task Learning and Clustering

Multi-task learning (MTL) deals with the problem of learning multiple related tasks simultaneously to improve the generalization performance of the models learned by each task with the help of the other auxiliary tasks [85, 86]. The extensive literature in MTL can be broadly categorized into two categories based on how the data is collected. The *centralized* approach assumes the data is collected beforehand at a centralized entity. Many successful MTL applications with deep networks, such as in natural language processing and computer vision, fall into this category [87–90]. This approach usually learns multiple objectives from a shared representation by sharing layers and splitting architecture in the deep networks. An example is to learn the depth and semantics from RGB images simultaneously [86]. On the other hand, the *distributed* approach assumes data is collected separately by each task in a distributed manner. This approach is naturally suited to model distributed learning in multi-agent systems such as mobile phones, autonomous vehicles, and smart

cities [49–51].

Although it is often assumed that a clustered, sparse, or low-rank structure among tasks is known *a priori* [91–93], such information may not be available in many real-world applications. Learning the relatedness among tasks online from data to promote effective cooperation is a principle approach in MTL when the relationships among tasks are not known *a priori*. There has been extensive work in online relationship learning that can be broadly categorized into centralized and distributed methods. The first group assumes that a centralized server collects the task models and utilizes a convex formulation of the regularized MTL optimization problem over the relationship matrix, which is learned by solving the convex optimization problem [94–96]. The second group relies on a distributed architecture in which agents learn relationships with their neighbors based on the similarities of their models and accordingly adjust weights assigned to neighbors [60, 97–99]. Typical similarity metrics, such as $\mathcal{H}$ divergence [55, 56, 100] and Wasserstein distance [56, 101], can be used in MTL in the same way they are used in domain adaptation, transfer learning, and adversarial learning. However, such metrics are mainly designed for measuring the divergence in data distributions and are not suitable for online relationship learning due to efficiency and privacy concerns in data sharing.

Distributed clustering is a relevant problem in the scope of MTL. Clustering is a well-known unsupervised learning technique for grouping a set of data points [102]. Compared to the traditional clustering problem, distributed clustering deals with the problem of grouping a set of networked agents in a multi-task network running individual optimization tasks into clusters by their tasks relatedness [60, 97, 103, 104]. In a distributed clustering network, agents perform local tasks while simultaneously learning which neighbors they should cooperate with by measuring their similarities. Compared to traditional clustering, distributed clustering is more challenging since any clustering error could lead an agent towards an undesired model. Measuring the Euclidean distance between the model parameters of agents is a principle method used in distributed clustering. For example, in [103], if the Euclidean distance is less than a pre-defined threshold, then the two agents will be clustered into the same group. Similarly, in [104], an accumulated Euclidean distance within a time-based sliding window is used. In addition, adaptive weights based on Euclidean distance are used in [97] and [60] for distributed clustering.

2.2.3 Task Similarity-based Resilient Cooperation

Distributed learning and optimization over networks rely on in-network processing and cooperation among neighboring agents. However, cooperation is beneficial only when agents share a common objective. Under the MTL setup, agents may belong to different clusters that pursue different objectives. And as a result, indiscriminate cooperation will damage the learning of agents and lead to an undesired outcome.

Thus, an effective clustering scheme is needed that enables agents to identify which cluster of neighbors to cooperate with and which other neighbors they should ignore, in order to benefit from the cooperation with neighbors. For instance, [60] proposes a method that assigns adaptive weights based on the similarity between models measured by Euclidean distance. Neighbors estimating a similar model with small Euclidean distance are assigned large weights while those with large Euclidean distance are assigned small weights. By doing so, only agents pursuing similar objectives cooperate, which the cooperation to be beneficial. Several variants focusing on adaptive weights using Euclidean distance in multi-task networks can be found in [97, 103–105]. However, it was studied in [1, 2] that measuring model similarity using Euclidean distance actually introduces other vulnerabilities that can be exploited by adversaries. Byzantine agents can employ the property of adaptive weights to send similar models to normal agents in order to gather a large weight. And by doing so, Byzantine agents can lead normal agents to converge to anywhere desired by the adversary via gradient-based attacks. Other similarity metrics, such as $\mathcal{H}$ divergence [55, 56, 100], Wasserstein distance [56, 101], cosine similarity [59] and Kullback–Leibler divergence [106] were proposed in the literature as a measurement of the similarity between two data distributions, which are widely used in multi-task learning, robust learning, domain adaptation, transfer learning, privacy preserving, and so on. In particular, the $\mathcal{H}$ divergence between the two distributions is measured to be large, if a predictor that performs well on one and badly on the other exists, whereas if all functions in the hypothesis class perform similarly on both of the distributions, then the $\mathcal{H}$ divergence is measured to be small [56]. The intuition behind Wasserstein distance is the optimal transport problem that for a distribution of mass on a space, to transport the mass in such a way that it is transformed into another distribution on the same space with the minimum cost, which is the amount of mass it needs to transport times the mean distance it has to be moved [56]. The cosine similarity measures the relative direction between two gradients. Consider the simple case when a server aggregates two workers' gradients to update its model. In a stationary solution, the server has a zero aggregated gradient. This is either when the two gradients are zero or the two gradients are of the same norm and point into opposite directions. Cosine similarity is used to check if the two points to opposite directions and therefore could resiliently cluster the gradients [59]. A summary of the resilient distributed learning algorithms we review in this section is given in Figure 2.1 and Table 2.4.

2.3 Resilient Distributed Reinforcement Learning

With recent exciting achievements of deep learning benefiting from big data, powerful computation, and new algorithmic techniques, we have been witnessing the renaissance of reinforcement learning (RL), especially combined with deep neural networks. As a data-driven method, RL provides solutions for controlling

Table 2.4: Summary of related work on resilient distributed learning.

	Param. Exchanged		Task Type		Network Topology	
Paper	Model	Gradient	Single	Multiple	Central Server	Fully-decentralized
[18, 52–54, 62–64]	-	✓	✓	-	✓	-
[58]	✓	-	✓	-	✓	-
[57]	✓	-	✓	-	-	✓
[1, 2, 55, 56, 60, 97, 103–105]	✓	-	-	✓	-	✓
[59]	-	✓	-	✓	✓	-

agents in complex environment without the need of developing high-fidelity physical models, and is capable of dealing with subtle considerations.

Markov decision processes (MDPs) are widely used for modeling RL problems. MDPs can be described formally as a tuple $M = \langle S, A, P, R \rangle$, where S is the state space, A is the set of action, $P : S \times A \to S$ is the transition function and $R : S \times A \times S \to \mathbb{R}$ represents the reward function [107]. In RL, a learning agent interacts with an unknown environment modeled by a MDP in discrete time step. At each step t, the agent receives an observation s_t from the environment, and takes an action $a_t \in A$, resulting in the environment moves to next state s_{t+1} with probability $P(s_t, a_t, s_{t+1})$, and the agent gets an immediate reward $r_{t+1} = R(s_t, a_t, s_{t+1})$ associated with the transition. The return (discounted long-term reward) $R = \sum_{t=0}^{\infty} \gamma^t r_t$, where γ is the discounted factor that determines how much future rewards are counted. The general objective of RL algorithms is to learn an optimal policy $\pi \in \Pi : S \to A$ that maximize the expectation of the return: $\max_{\pi \in \Pi} \mathbb{E}_\pi(R)$.

The technique of training multiple RL agents distributedly for a common objective has been extensively studied in recent years. Related work in distributed RL can be broadly grouped into two categories. In the first category, multiple agents operate in similar but independent MDPs whose actions do not affect each other [108, 109]. Such an approach is widely used in recent RL techniques for parallel exploration and computation to accelerate exploration and speed up learning, especially combined with deep neural networks. It is also naturally suited to be used in multi-agent networks where networked agents perform similar RL tasks in independent environments. We will discuss this approach in detail in Section 2.3.1. The second category considers training RL algorithms for multiple agents in a single MDP, which can be further divided into team learning and concurrent learning [110]. Team learning, using a single learner to learn the behaviors for the entire team, is not a distributed learning approach and is out of the scope of this thesis. In contrast, concurrent learning deals with the problem of making each agent learn in a shared MDP concurrently. There are limitations to both of the two approaches. Team learning suffers from the scalability issues when the number of agents is large; yet for concurrent learning, multiple learners taking actions in the same MDP

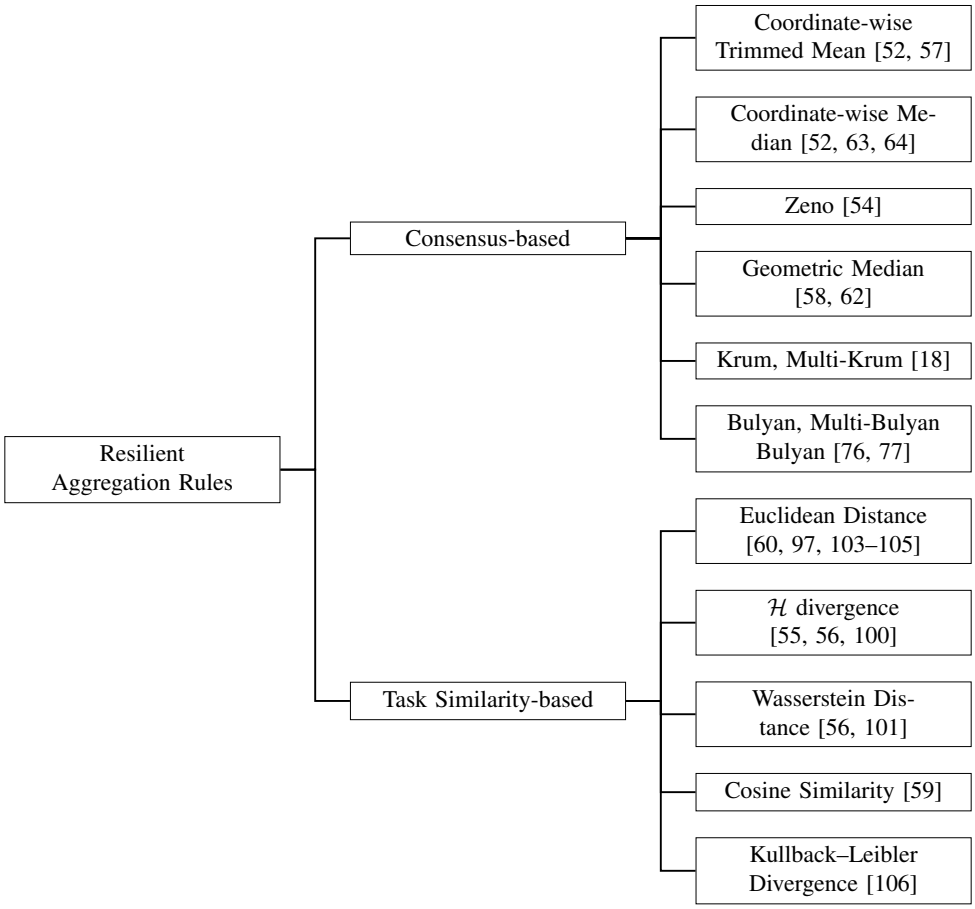


Figure 2.1: Resilient aggregation rules in distributed learning algorithms.

will cause the environment to be non-stationary, thus violating the assumptions behind the traditional RL algorithms. Many emerging techniques have been proposed for concurrent learning which we will discuss in detail in Section 2.3.2. A taxonomy of the distributed RL problems considered in the literature is given in Figure 2.2.

2.3.1 Distributed Reinforcement Learning in Independent MDPs

The technique of training multiple RL workers in independent MDPs in parallel to expand experience memory and improve learning efficiency has been widely used, especially combined with deep neural networks. A major body of such works considers using a centralized parameter server for model updates and multiple actors to execute in multiple instances of the environment in parallel to collect state-action pairs. For example, [109] considers the parallel training of a distributed Deep Q Network (DQN) where a single parameter server collects the gradients of the Q network from distributed actors and updates the parameters of the Q network; and the distributed actors explore in independent MDPs and store the collected state-action

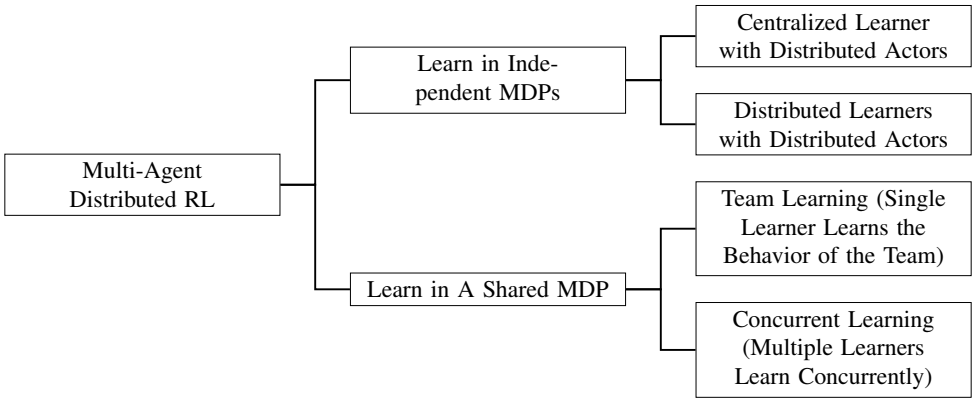


Figure 2.2: Distributed reinforcement learning algorithms.

pairs in the same experience replay memory and compute the gradients of the model. Another well-known instance of such a paradigm termed A3C (Asynchronous Advantage Actor-Critic) is proposed in [111] where they present an asynchronous variant of the actor-critic algorithm using asynchronous actors. Distributed RL in fully-decentralized networks has also been studied in the literature [108, 112–114]. For example, [112] proposes a distributed implementation of Q-learning called QD-learning where every normal agent collaboratively updates tabular Q-values that being shared with their neighbors. Further, [108] proposes a distributed parallel RL method for policy evaluation with linear value function approximation. Moreover, [113] proposes a distributed actor-critic framework that aims to learn a policy that performs well on average for the whole set of tasks. Most of these works assume a single-task network, whereas [114] considers multi-task RL where each agent is trained to solve its own task while constrained to stay close to the shared policy.

Besides the advantages of expanding the replay memory and speed up learning, training distributed RL algorithms distributedly in independent MDPs has many practical applications in multi-agent networks, especially for recommender systems [115–117]. Although personalizing customer interactions at scale through the data analysis of users' online behavior patterns has been realized by machine learning, recommending special content types such as news and online short videos is still challenging. Reasons are that user preferences in topics change frequently and the features of those contents are dynamic by nature and become rapidly irrelevant. RL, in particular, can be fit into such scenarios in real-time. For example, consider learning the behavior of users using RL algorithms in a cellular network based on data generated using various mobile applications. Each user may generate data that follows a distinct distribution and it is common to learn separate models for each user. However, people may exhibit similar behaviors and similarities among models commonly exist [15]. In this case, cooperation among agents could be leveraged. However, given data privacy concerns, cooperation among agents in a network relies typically on exchanging model parameters instead of data. As a result, sharing experience memory by storing the state-action pairs in a common

memory accessible to each agent [109] is not applicable to such privacy-sensitive real-world applications. In a distributed learning network, an agent communicates model parameters with its local neighbors and also updates these parameters by incorporating the neighbors' information [16]. It has been demonstrated that such cooperation enables improved learning performance over the network [73].

2.3.2 Distributed Reinforcement Learning in A Shared MDP

Another line of research in distributed RL is the concurrent learning problem, also known as the multi-agent reinforcement learning (MARL) problem, where multiple interactive agents work collaboratively or competitively in the same environment modeled by a single MDP. One straightforward method in solving this problem is to learn an individual policy for each agent by their individual actions without considering the other agents' actions, which has been successfully applied to the two-player pong game [118]. However, multiple agents taking actions in the environment rendering the environment non-stationary, making the experience buffer obsolete frequently and leading to convergence problems. To address such problems, several methods have been proposed, such as by taking other agents' actions into account when updating one's critic network [119], or using importance sampling to naturally decay obsolete data in experience buffer [120]. For example, [119] proposes a multi-agent policy gradient algorithm where agents learn a centralized critic network based on the observations and actions of all agents. Their method has better empirical results than traditional RL algorithms, such as DQN, Actor-Critic, a first-order implementation of Trust Region Policy Optimization (TRPO), and Deep Deterministic Policy Gradient (DDPG), in a variety of cooperative and competitive multi-agent environments. In addition, [121] proposes a two-stage multi-goal multi-agent policy gradient approach where at the first stage, a single agent learns for their goal attainment; and at the second stage, a credit assignment method is used to learn a centralized policy by the cooperation among agents. Moreover, algorithms that use parameter sharing among agents in MARL can be found in [122–125], where communication among agents contributes to improved overall performance.

2.3.3 Resilience in Distributed Reinforcement Learning

Although research in Byzantine resilient aggregation for distributed learning algorithms is very broad, studies focusing on resilient distributed RL are limited. A recent work presented in [126] uses trimmed mean to achieve resilience, where a centralized server exists in the network. In addition, a resilient version of QD-learning in a full-decentralized network has been proposed in [127], which is based on the W-MSR algorithm.

2.4 Comparison to This Dissertation

The work presented in this thesis addresses multiple resilient consensus, learning and optimization problems in distributed multi-agent systems through designing resilient cooperation mechanisms and aggregation rules. Specially, we seek to develop efficient aggregation rules for distributed consensus, learning and optimization systems with strong resilience guarantees that can secure the system from Byzantine attacks. In particular, compared to the previous work that directly applies consensus algorithms into distributed learning and optimization systems for resilient cooperation purpose, which faces the dilemma that 1) resilient scalar consensus algorithms cannot guarantee resilient vector consensus, and 2) it is not practical to use resilient vector consensus algorithms for high-dimensional model parameters given its exponential computational complexity in the dimension, we propose resilient aggregation methods for distributed learning and optimization by measuring the similarities among agents, which addresses the dilemma between resilience and computational complexity. In addition, we study the cooperation in fully-decentralized multi-agent learning and optimization systems and provide analytical and empirical evidence to demonstrate the benefits from such cooperation, in distributed supervised learning, clustering, and reinforcement learning networks.

Chapter 3

Resilient Vector Consensus in Multi-Agent Networks Using Centerpoints [1]

In this chapter, we study the resilient vector consensus problem in multi-agent networks and improve resilience guarantees of existing algorithms. In resilient vector consensus, agents update their states, which are vectors in $\mathbb{R}^d$, by locally interacting with other agents some of which might be adversarial. The main objective is to ensure that normal (non-adversarial) agents converge at a common state that lies in the convex hull of their initial states. Currently, resilient vector consensus algorithms, such as approximate distributed robust convergence (ADRC) are based on the idea that to update states in each time step, every normal node needs to compute a point that lies in the convex hull of its normal neighbors' states. To compute such a point, the idea of Tverberg partition is typically used, which is computationally hard. Approximation algorithms for Tverberg partition negatively impact the resilience guarantees of consensus algorithm. To deal with this issue, we propose to use the idea of centerpoint, which is an extension of median in higher dimensions, instead of Tverberg partition. We show that centerpoint provides a better characterization of the necessary and sufficient conditions guaranteeing resilient vector consensus and is computationally more efficient. We analyze these conditions in two, three, and higher dimensions separately. We also numerically evaluate the performance of our approach.

3.1 Introduction

Resilient consensus in a network of agents, some of which might be adversarial or faulty, has several applications in multirobot networks, distributed computing, estimation, learning and optimization (for instance, see [2, 27, 32, 128, 129]). The main goal of resilient consensus is to ensure that all normal agents in a network agree on a common state despite the presence of some adversarial agents, which aim to prevent normal nodes from consensus and whose identities are unknown to normal agents. Resilient consensus is achieved if appropriate state update laws are designed for normal agents and the underlying network topology satisfies certain connectivity and robustness conditions. For instance, when agents' states are scalars, [130] presents a resilient distributed algorithm guaranteeing convergence of normal nodes to a common state.

If agents' states are vectors or points in $\mathbb{R}^d$, $d \geq 2$, then the resilient consensus objective is to ensure that

[1] ©2020 IEEE. Adapted with permission, from [Mudassir Shabbir, Jiani Li, Waseem Abbas and Xenofon Koutsoukos, "Resilient Vector Consensus in Multi-Agent Networks Using Centerpoints," 2020 American Control Conference (ACC), 2020, pp. 4387-4392, doi: 10.23919/ACC45564.2020.9147441].

normal agents converge at some point in the convex hull of their initial states. A simple approach could be to run d instances of scalar resilient consensus, one for each dimension. However, as a result of this approach, normal agents might converge at a point outside of the convex hull of their initial states, as discussed in [22]. Thus, we cannot rely on resilient scalar consensus algorithms to achieve resilient vector consensus. Various solutions have been proposed to achieve resilient vector consensus, which has been an active research topic, for instance see [22, 28, 32–34].

In this chapter, we study the resilient vector consensus problem and a recently proposed solution referred to as the *Approximate Distributed Robust Convergence (ADRC)* algorithm in [32]. We show that the resilience of the algorithm, in terms of the number of adversarial agents whose presence does not prevent normal agents from converging to a common state in the desired convex hull, is improved with some simple modification. In particular, if normal agents implement ADRC as in [32], then consensus is guaranteed if the number of adversarial agents in the neighborhood of a normal agent i is $n_{f_i} \leq \left\lceil \frac{|\mathcal{N}_i|}{2^d} \right\rceil - 1$, where $|\mathcal{N}_i|$ is the number of nodes in the neighborhood of i, and d is the dimension of state vector. We show that in the case of $d = 2, 3$, consensus is guaranteed if $n_{f_i} \leq \left\lceil \frac{|\mathcal{N}_i|}{d+1} \right\rceil - 1$, and for $d > 3$ if $n_{f_i} \leq \left\lceil \frac{|\mathcal{N}_i|}{d^{\frac{r}{r-1}}} \right\rceil - 1$ where r can be any integer.

ADRC is an iterative algorithm, and in each iteration, a normal node needs to compute some point in the convex hull of points corresponding to its normal neighbors' states. To compute such a point, which is referred to as the *safe point*, authors in [32] utilize the idea of *Tverberg partition* of points in $\mathbb{R}^d$ (discussed in Section 3.3). We argue that instead of computing Tverberg partition, it is much better to use the notion of *centerpoint* in $\mathbb{R}^d$ to compute safe points. The notion of centerpoint and its properties have been an active research topic in discrete geometry [131, 132]. A centerpoint essentially extends the notion of median in higher dimensions. We show that safe points, as used in ADRC algorithm, are essentially the interior centerpoints. This perspective provides a complete characterization of safe points, and hence allows us to improve the resilience bound of the algorithm.

- We show that the resilience of ADRC algorithm can be improved by using the notion of centerpoint instead of Tverberg partition. We discuss these improvements in two, three and higher dimensions separately.

- Using centerpoints, we show that $|\mathcal{N}_i| \geq n_{f_i}(d+1)+1$ is not only sufficient but also necessary to compute a safe point, which is a key step in the ADRC algorithm. Here n_{f_i} is the number of adversaries in the neighborhood of a normal node i. We also provide an overview of various algorithms reported in the literature to compute centerpoints in different dimensions.

- We compare and numerically evaluate our results with the existing algorithm by simulating resilient vector consensus in multirobot networks.

The rest of the chapter is organized as follows: Section 3.2 introduces notations and preliminaries. Section 3.3 provides an overview of the ADRC algorithm. Section 3.4 discusses the notion of centerpoint for ADRC and presents main results in the chapter. Section 3.5 gives a numerical evaluation of our results, and Section 3.7 concludes the chapter.

3.2 Notations and Preliminaries

We consider a network of agents modeled by a *directed graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with self-loops allowed, where $\mathcal{V}$ represents agents and $\mathcal{E}$ represents interactions between agents. Each agent $i \in \mathcal{V}$ has a d-dimensional state vector whose value is updated over time. The state of each agent i at time t is represented by a point $x_i(t) \in \mathbb{R}^d$. An edge (j, i) means that i can observe the state value of j. The *neighborhood* of i is the set of nodes $\mathcal{N}_i = \{j \in \mathcal{V} | (j, i) \in \mathcal{E}\}$. For a given set of points $X \subset \mathbb{R}^d$, we denote its *convex hull* by conv(X). A set of points in $\mathbb{R}^d$ is said to be in *general positions* if no hyperplane of dimension $d-1$ or less contains more than d points. A point $x \in \mathbb{R}^d$ is an *interior point* of a set $X \subset \mathbb{R}^d$ if there exists an open ball centered at x which is completely contained in X. We use terms agents and nodes interchangeably, and similarly use terms points and states interchangeably.

Normal and Adversarial Agents. There are two types of agents in the network, normal and adversarial. *Normal* agents are the ones that interact with their neighbors synchronously and always update their states according to a pre-defined state update rule, that is the consensus algorithm. *Adversarial* agents are the ones that can change their states arbitrarily and do not follow the pre-defined state update rule. Moreover, an adversarial node can transmit different values to its different neighbors, which is referred to as the *Byzantine* model. The number of adversarial nodes in the neighborhood of a normal node i is denoted by n_{f_i}. For a normal node i, all nodes in its neighborhood are indistinguishable, that is, i cannot identify which of its neighbors are adversarial.

Resilient Approximate Vector Consensus. The goal of the resilient vector consensus is to ensure the following two conditions:

- *Safety* – Let $X(0) = \{x_1(0), x_2(0), \cdots, x_n(0)\} \subset \mathbb{R}^d$ be the set of initial states of normal nodes, then at each time step t, and for any normal node i, the state value of i, denoted by $x_i(t)$ should be in the conv$(X(0))$.

- *Agreement* – For every $\epsilon > 0$, there exists some t_ϵ, such that for any normal node pair i, j, $||x_i(t) - x_j(t)|| < \epsilon, \forall t > t_\epsilon$.

3.3 Background and Approximate Distributed Robust Convergence (ADRC) Algorithm

In this section, first we provide an overview of a resilient vector consensus algorithm known as the approximate distributed robust convergence, recently proposed in [32]. Then, we discuss improvement in resilience guarantees of the algorithm by reconsidering its computational aspects.

The ADRC is an iterative algorithm, in which a normal node i gathers the state values of its neighbors in each iteration t, and then computes a point that lies in the interior of the convex hull of its normal neighbors' states. After computing this point, which is referred to as the *safe point* $s_i(t)$, node i updates its state as follows:

$$x_i(t+1) = \alpha_i(t)s_i(t) \ + \ (1-\alpha_i(t))x_i(t), \tag{3.1}$$

where, $\alpha_i(t)$ is a dynamically chosen parameter in the range $(0 \quad 1]$.[2] It is shown in [32] that if all normal nodes follow this procedure, they converge at a common point and achieve resilient consensus (satisfying safety and agreement conditions stated in the previous section).

Computation of a safe point in each iteration is the key step in the algorithm. For this, [32] utilizes results from discrete geometry, in particular the idea of *Tverberg partitions* [133] and related results. We first state the main result regarding the partitioning of points in $\mathbb{R}^d$, and then discuss the application of this result, as adapted in [32], for computing safe points.

Proposition 3.1. *([134–136]) If we have a set X of n points in general positions in $\mathbb{R}^d$, where $n \geq (r-1)(d+1)+1$ and $d \leq 8$, then it is possible to partition X into r pairwise disjoint subsets $X_1, X_2, \cdots, X_r$ such that the intersection of convex hulls of these r subsets is non-empty and is at least d-dimensional.*

Such a partition is a *Tverberg partition*. Now, consider a normal node i in our network having n neighbors in its neighborhood out of which at most n_f are adversarial.[3] Each node corresponds to a point in $\mathbb{R}^d$. The goal for a node i is to compute an interior point in the convex hull of $n - n_f$ normal points. If $n \geq n_f(d+1)+1$, then by Proposition 3.1, we will have a partition of n points into $n_f + 1$ subsets such that the intersection of convex hulls of these subsets is non-empty and is d-dimensional. We call this intersection region as *Tverberg region*. Since there are at most n_f adversarial nodes and we have $n_f + 1$ subsets in the partition, one of these subsets consists of points corresponding to normal nodes only. Let us denote this subset by X^*. Note that the Tverberg region lies in the convex hull of X^*, and conv(X^*) itself lies in the convex hull of all normal nodes points. Consequently, every interior point in the Tverberg region is a safe point. Thus, to compute a safe point, a normal node i computes a Tverberg partition, which is possible if

[2]The choice of $\alpha_i(t)$ depends on applications, for instance, in multirobot systems, it is selected such that the physical constraints including maximum allowable displacement by a robot is not violated.

[3]For the ease of notation, we drop the subscript i from n_i and n_{f_i} denoting the total number of neighbors and the number of adversarial neighbors of node i respectively from here onward. Note that, in general these values can be different for different nodes.

$n \geq n_f(d+1)+1$. In other words, a normal node can compute a safe point in the presence of n_f adversarial neighbors if $n_f \leq \lceil \frac{n}{d+1} \rceil - 1$. Figure 3.1 gives an illustration of these ideas.

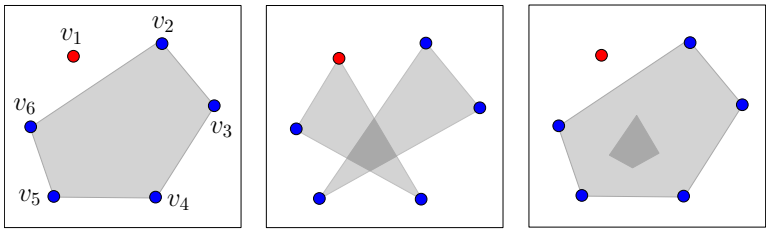


Figure 3.1: (a) Five normal (blue) and a single adversarial node. Shaded area is the convex hull of normal nodes. (b) Tverberg partition consisting of two subsets, out of which one contains only normal nodes. Convex hulls of both subsets have a non-empty intersection, corresponding to a Tverberg region. (c) Intersection of Tverberg region and the convex hull of normal nodes.

However, computing a Tverberg partition in general is an NP-hard problem. The best known algorithm that computes it in a reasonable run time is an approximate algorithm [137], which has a time complexity of $d^{O(1)}n$. The algorithm is approximate in a sense that to have a partition of n points into r subsets, $n \geq 2^d r$ (as compared to $n \geq (r-1)(d+1)+1$ in Proposition 3.1). Consequently, to compute a safe point in the presence of n_f adversarial neighbors, a normal node needs to have at least $n \geq (n_f+1)2^d$ nodes in its neighborhood. In other words, with a total of n neighbors, a node i can compute a safe point, and hence achieve resilient consensus (using ADRC) if there are n_f adversarial nodes in its neighborhood, where

$$n_f \leq \left\lceil \frac{n}{2^d} \right\rceil - 1. \tag{3.2}$$

Note that (3.2) indicates *resilience* of the ADRC algorithm that relies on approximate Tverberg partitions to compute safe points. For instance, the algorithm guarantees resilient consensus in $\mathbb{R}^2$ if for every normal node, less than 25% of its neighbors are adversarial.

3.3.1 How Can We Improve the Resilience of ADRC?

Here, we ask if it is possible to improve the resilience of the ADRC algorithm? What modifications will allow us to guarantee consensus even if the number of adversarial nodes in the neighborhood of a normal node is greater than $\lceil \frac{n}{2^d} \rceil - 1$? Next, we show that it is possible to achieve a better resilience bound if we use a slightly different way of computing safe points, that is by using the notion of *centerpoint* instead of Tverberg partition. Moreover, centerpoint provides a better characterization of necessary and sufficient conditions for computing safe points.

3.4 ADRC Using Centerpoints

In this section, we explain the notion of a *centerpoint* and its relation to safe point. Then, we discuss that computing a safe point through centerpoint is more desirable as it results in improving the resilience of the ADRC algorithm.

3.4.1 Safe Point and the Interior Centerpoint

The notion of safe point is pivotal in the ADRC algorithm, so we define n_f-safe point as in [32] below.

Definition 3.1. (n_f-Safe point) *Given a set of n points in d dimensions, of which at most n_f correspond to adversarial nodes, an n_f-safe point is a point that lies in the relative interior of the convex hull of $(n - n_f)$ normal points. We refer to a $(\lceil \frac{n}{d+1} \rceil - 1)$-safe point in $\mathbb{R}^d$ as an optimal safe point or just safe point.*

As we discussed in the last section, a point that lies in the interior of the Tverberg region of (n_f+1) subsets is always an n_f-safe point Here, we provide a better characterization of n_f-safe point using centerpoint, which is defined below.

Definition 3.2. (Centerpoint) *Given a set S of n points in $\mathbb{R}^d$ in general positions, a centerpoint p is a point, not necessarily from S, such that any closed half-space[4] of $\mathbb{R}^d$ that contains p also contains at least $\lceil \frac{n}{d+1} \rceil$ points from S.*

Intuitively, a centerpoint lies in the "center region" of the point cloud, in the sense that there are enough points of S on each side of a centerpoint. A centerpoint, essentially, extends the notion median to higher dimensions. A related notion of centerpoint depth is defined as follows:

Definition 3.3. *For a given pointset, centerpoint depth or simply depth of a point q is the maximum number α such that every closed halfspace containing q contains at least α points.*

Thus, a centerpoint has depth at least $\lceil \frac{n}{d+1} \rceil$. The existence of such a point for any given set S is guaranteed by the famous Centerpoint Theorem (see [138, 139]).

Theorem 3.1. (Centerpoint Theorem) *For any given point set in general positions in an arbitrary dimension, a centerpoint always exists.*

A centerpoint doesn't need to be unique, in fact, there can be infinitely many centerpoints. The set of all centerpoints constitutes the *centerpoint region* or simply the *center region*. It is known that center region is closed and convex. We observe that the safe point from [32] is actually an interior centerpoint.

[4]Recall that closed half-space in $\mathbb{R}^d$ is a set of the form $\{x \in \mathbb{R}^d : a^T x \geq b\}$ for some $a \in \mathbb{R}^d \setminus \{0\}$.

Theorem 3.2. *For a given set of points S in $\mathbb{R}^d$, an n_f-safe point is equivalent to an interior centerpoint for $n_f = \lceil \frac{n}{d+1} \rceil - 1$.*

Proof: See [3].

Theorem 3.2 provides a complete characterization of a safe point in the presence of n_f adversarial nodes. Here, we would also like to note that $n_f = \frac{1}{d+1} - 1$ is the best possible fraction, that is, there exist general node positions where allowing more adversary nodes would mean that there is no safe point at all.

Proposition 3.2. *For a set of n nodes in general positions, if $n_f \geq \lceil \frac{n}{d+1} \rceil$, then there exist general examples in which an n_f-safe point does not exist.*

Proof. Imagine $d + 1$ copies of $\frac{n}{d+1}$ points at the vertices of a non-degenerate d-simplex. If there are $\frac{n}{d+1}$ adversarial nodes whose identities are unknown, then there is no point that lies in the convex hull of remaining points. Note that points in this examples can be arbitrarily perturbed to ensure that general positions condition is not violated. □

Figure 3.2 demonstrates Proposition 3.2 for the planar case. Further, we note that every point in the intersection of an appropriate Tverberg partition is a centerpoint and thus, also a safe point. However, the converse is not true; a centerpoint or a safe point need not be a Tverberg point in general.

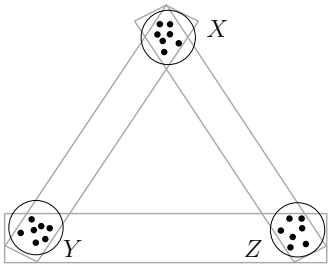


Figure 3.2: S is partitioned into X, Y, Z each of which contains $n/3$ points. If there are $n/3$ adversarial nodes then points in either of these three sets can all be adversarial. We require that an n_f-safe point must lie in the convex hull of normal nodes for all three possibilities. This is not possible because intersection of three possible sets of normal nodes $X \cup Y$, $Y \cup Z$, $Z \cup X$ is empty. Therefore, there is no n_f-safe point in this case.

In [32], normal nodes compute safe points using approximate Tverberg partitions [137], which deteriorate the resilience of ADRC algorithm from $n_f \leq \lceil \frac{n}{d+1} \rceil - 1$ to $n_f \leq \lceil \frac{n}{2^d} \rceil - 1$. However, with this new characterization, we can use centerpoints to compute safe points. Thus, if we are able to compute centerpoint exactly (in a reasonable run time), then we are able to improve the resilience of ADRC algorithm, that is,

$$n_f \leq \left\lceil \frac{n}{d+1} \right\rceil - 1 \tag{3.3}$$

as compared to (3.2).

Next, we discuss the existence and computation of interior centerpoints in two, three and higher dimensions separately.

3.4.2 Centerpoint-based Resilient Consensus in 2-D

In [32], authors show that an n_f-safe point can be found in $\mathbb{R}^2$ when the number of adversarial nodes n_f is at most $\min\left(\lceil\frac{n}{4}\rceil, \lfloor\frac{n}{3}\rfloor\right) - 1$ where n is total number of nodes in the neighborhood of a normal node. As is evident, this is a loose bound on the resilience of such a consensus algorithm. Unfortunately, that is the best that can be hoped for if one is to seek a safe-point using an approximate Tverberg partition. Tverberg partitions are, in general, thought to be computationally expensive. However, we have showed in Theorem 3.2 that safe points and the interior centerpoints always coincide, and in the following subsection, we summarize a well-known linear time algorithm to find a centerpoint in the plane. It should be pointed out that complexity of finding a centerpoint in general dimensions is unknown, although computing centerpoint depth of a given point is coNP-Complete. In this section, we propose the following result to compute a safe point in the plane:

Theorem 3.3. *Given a set of n points in two dimensions in general positions, an n_f-safe point exists whenever the number of adversarial nodes is $n_f \leq \lceil\frac{n}{3}\rceil - 1$. Moreover, such a safe point can be computed in linear time.*

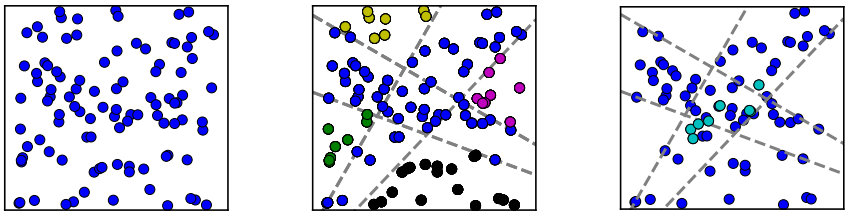
Proof: See [3].

Remark 3.4. *The set of Tverberg points is a subset of the centerpoint region. Thus, the existence of an interior centerpoint in the plane, as shown above, is also implied by the existence of an interior Tverberg point in dimensions two to eight. We hope that the alternative proof above may help extend this result to dimensions greater than 8 for which the existence of an interior centerpoint and interior Tverberg point are unknown.*

3.4.3 Computing Centerpoint in 2-D

Here we address the computational aspects of the centerpoint in two dimensions. Due to a seminal result in [140], it is possible to find a centerpoint for a non-degenerate pointset in the optimal $O(n)$ time. We remark that this result is also significant because it makes finding a centerpoint in linear time possible even when checking whether a point is a centerpoint can not be done in better than $\Omega(n \log n)$ time. Here, we briefly outline this method to compute a centerpoint of a set of points; the details can be found in [140].

The algorithm is based on the idea that by pruning or replacing some of the "marginal points", a centerpoint of the remaining points is still a centerpoint of the original pointset. In each iteration one can compute the points that are to be discarded or replaced, which will reduce the size of the set by a fraction. We continue the pruning procedure until the size of the set becomes smaller than a fixed constant, one can then compute a centerpoint by any straightforward brute-force method. Pruning of points is a pivotal step in the algorithm. Given a set of n points $\mathcal{P}$, we start by defining four half-planes, named L, U, R, and D (representing Left, Up, Right, and Down, respectively), such that each of them contains less than $\lceil \frac{n}{3} \rceil - 1$ points (this ensures that they don't contain any centerpoint) and their closures contain at least $\lceil \frac{n}{3} \rceil$ points. And the closure of each of the sets $L \cap U$, $L \cap D$, $R \cap U$, and $R \cap D$ contains at least $\left(\lceil \frac{n}{3} \rceil - \lceil \frac{n}{4} \rceil \right)$ points. It is, then, argued that either one can discard the points of a triangle on three points from three of the four intersections or substitute four points from the four intersection sets by their Radon point[5]. This reduces the size of $\mathcal{P}$ by a significant fraction and a centerpoint of the remaining point set is also a centerpoint of the original point set. The pruning process, as illustrated in Figure 3.3, is repeated until the number of points is less than a small constant, and then one can compute the centerpoint by a brute-force method. The construction of halfplanes with the prescribed number of points in their intersection is achieved by the famous ham-sandwich cut algorithm [141].



(a) (b) (c)

Figure 3.3: One iteration illustration of replacing points in $L \cap U$, $L \cap D$, $R \cap U$, and $R \cap D$ by their Radon points: (a) point set of 100 points, (b) intersections of the four half-planes, and (c) replacement of points in the intersections by their Radon point.

3.4.4 Centerpoint-based Resilient Consensus in 3-D

The resilience bound that we get from the results in [32] guarantee an n_f-safe point in three dimensions whenever $n_f \leq \lceil \frac{n}{8} \rceil$-1 adversarial nodes. From the centerpoint theorem, we know that a safe point exists in the interior of centerpoint region in 3-D even in the presence of $(n/4) - 1$ adversarial nodes. In context of Theorem 3.2, this property can be leveraged to present a better resilience guarantee.

[5] Any set of 4 points in $\mathbb{R}^2$ can be partitioned into two disjoint sets whose convex hulls intersect. A point in the intersection of these convex hulls is called a Radon point of the set.

Theorem 3.5. *An* $\left(\lceil \frac{n}{4} \rceil - 1\right)$*-safe point exists for every pointset in general positions in* $\mathbb{R}^3$*. Such a point can be computed in* $O(n^2)$ *expected time.*

Proof. We know that an $\left(\lceil \frac{n}{4} \rceil - 1\right)$-safe point is an interior centerpoint by Theorem 3.2, and [135] implies that an interior centerpoint must exist in three dimensions. A randomized algorithm by Chan can be used to compute a centerpoint in three dimensions in $O(n^2)$ expected time [142]. We proceed by running Chan's algorithm four times, and compute the centroid of the four centerpoints returned to get an interior centerpoint. □

Next, we provide a brief overview of Chan's algorithm in which he computes a centerpoint of a non-degenerate pointset in $O(n^{d-1})$ time [142]. He first solves the decision version of the problem: does there exist a point of depth k for a given k? If the answer is yes, then a point of given depth is reported as well. This decision version is solved using a randomized Linear Program solver by dualizing the pointset: given points S are dualized to a set $S*$ of hyperplane and a point of given depth dualizes to special hyperplane that has at least k hyperplanes from $S*$ above or below it. The problem of finding this hyperplane is solved by partitioning the space and solving sub-problems in each smaller region. The partitioning is done by the famous Cutting Lemma [143]. For further details, we refer to the chapter [142].

3.4.5 Centerpoint-based Resilient Consensus in d-dimensions for $d > 3$

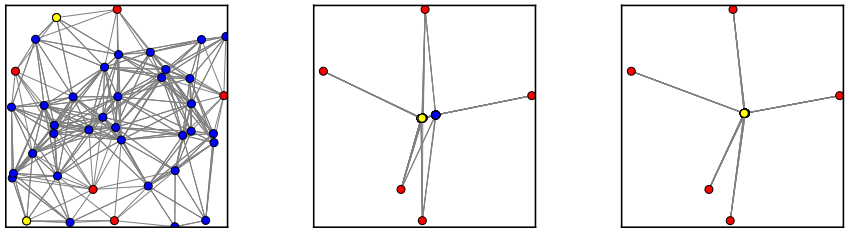
In higher dimensions, current methods to compute either a desirable Tverberg partition or a centerpoint for a given pointset become computationally impractical. It is known deciding whether a point lies in the intersection of a Tverberg partition is NP-Complete and deciding whether a point is centerpoint is coNP-Complete. Various approximations are employed to compute these points in practice. In [32], authors use a "lifting-based" approximation that finds an n_f-safe point in presence of $n_f \leq \frac{n}{2^d} - 1$ adversarial nodes. In the following, we outline an algorithm by Miller and Sheehy to compute an approximate centerpoint [144]. The point returned by this algorithm has a centerpoint depth of $\frac{n}{d^{r/r-1}}$ for any integer $r \geq 2$. For $r = 3$, this gives an n_f-safe point when the number of adversarial nodes is at most $\frac{n}{d^{3/2}}$. By increasing r, the quality of approximation, and hence the bound on the number of adversarial nodes improves. However, it comes at the cost of increasing time complexity as the runtime of the algorithm is $O(rd^d)$ for an integer $r > 1$.

Miller and Sheehy centerpoint-approximation algorithm is based on the technique of Radon's theorem which states that for any given set of at least $d+2$ points, there exists a partition into two sets with intersecting convex hulls; a point in the intersection of the two said sets is called a Radon point. They improve upon a classic algorithm that starts by partitioning a given pointset S into groups of $d + 2$ points and computing Radon point for each group. The set of $\lceil \frac{|S|}{d+2} \rceil$ Radon points returned in the previous iteration are assumed to

be the new pointset and centerpoint for these points is recursively computed. An approximate centerpoint is, thus, found in at most $\log_{d+2}|S|$ iterations. Miller and Sheehy showed that they can create groups of larger sizes (of multiples of $d+2$ points) and reduce the number of iterations. Details of their algorithm, anyalsis and proof of correctness is available in [144]. As a consequence, we have the following result:

Theorem 3.6. *For a given pointset in $\mathbb{R}^d$ in general positions, a $\left(\frac{n}{d^{r/r-1}}\right)$-safe point exists and can be computed in time $O(rd^d)$ for any integer $r > 1$.*

Thus, using approximate centerpoint in dimension d, consensus is guaranteed if the number of adversarial nodes in the neighborhood of every normal node is $n_f \leq \frac{n}{d^{r/r-1}}$ for an integer $r > 1$, which is better than the resilience achieved by using approximate Tverberg partition, where $n_f \leq \frac{n}{2^d} - 1$.



(a) (b) (c)

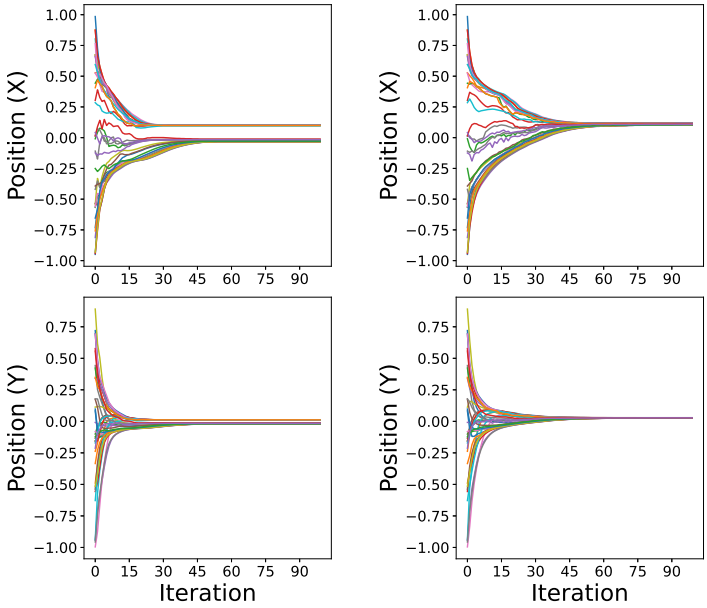
Figure 3.4: (a) Initial positions of robots. (b) Final positions of robots using approximate Tverberg partition based algorithm. (c) Final positions using centerpoint based algorithm.

3.5 Evaluation

We perform simulations[6] to compare resilient consensus in multirobot systems in two dimensions using centerpoint and approximate Tverberg partition [32]. At each iteration t of the multi-robot consensus algorithm, a normal robot i computes a safe point $s_i(t)$ of its neighbors' positions (using centerpoint or approximate Tverberg partition), and calculates its new position using (3.1). In our experiments, we set $\alpha_i(t) = 0.8$. We consider stationary adversarial nodes, and assume that the network graph is undirected and fixed.[7] A group of 45 robots of which 5 are adversarial is distributed in a planar region $\mathcal{W} = [-1,1] \times [-1,1] \in \mathbb{R}^2$ as shown in Figure 3.4(a). All normal robots have at most $\left(\lceil \frac{|\mathcal{N}_i|}{3} \rceil - 1\right)$ adversaries in their neighborhood, which means resilient consensus is guaranteed by the centerpoint based algorithm. However, a couple of normal robots (depicted in yellow color) have n_{f_i} adversaries in their neighborhood, where $\left(\lceil \frac{|\mathcal{N}_i|}{4} \rceil - 1\right) < n_{f_i} \leq \left(\lceil \frac{|\mathcal{N}_i|}{3} \rceil - 1\right)$. For the two robots in yellow, they have 7 and 8 neighbors, and both of them have 2

[6] Our code is available at https://github.com/JianiLi/MultiRobotsRendezvous

[7] For additional experiments with disk graphs and moving adversaries, see [3].



(a) (b)

Figure 3.5: Positions of normal robots as a function of iterations using (a) approximate Tverberg partition based, and (b) centerpoint based algorithms.

adversarial neighbors in their respective neighborhoods. Consequently, the resilient consensus condition for the approximate Tverberg partition based algorithm is not satisfied, and consensus is not guaranteed. Figures 3.4(b) and (c) show final positions of robots for both algorithms. It is clear that consensus is achieved with the centerpoint based algorithm, whereas robots fail to converge at a common point using the approximate Tverberg partition based algorithm. Figure 3.5 illustrates positions of robots as a function of iterations and demonstrates the same results.

3.6 Resilient Asynchronous Approximate Vector Consensus Using Centerpoints

The approximate vector consensus problem in Byzantine asynchronous systems has been extensively studied, e.g., in [22, 23, 28]. In this section, we extend the proposed resilient synchronous approximate consensus method using centerpoints to the *asynchronous systems* in *complete graphs* based on [28]. In a synchronous system, operations are coordinated by one, or more, centralized clock signals. In contrast, there is no global clock in an asynchronous system and no upper bound on relative speeds of messages or on message delay is required. As a result, it is not possible to distinguish between a faulty agent that has halted and a normal one that is simply slow to respond [28]. For incomplete graphs, agents connected to different Byzantine agents may end up in different consensus states. As a result, in the case of incomplete graphs, resilient asynchronous approximate vector consensus relies on some robustness conditions on the

connectivity graphs [33], which is not considered in this section.

3.6.1 Iterative Algorithms and Resilience Bounds

It has been well established in [28] that the iterative algorithm given in Algorithm 1 can be used for any normal agent in multiple discrete iterations to achieve resilient asynchronous approximate vector consensus in a complete graph.

Algorithm 1: Resilient Asynchronous Approximate Vector Consensus

1 **for** *iteration* $t \geq 0$ **do**
2 **for** *every agent* i **do**
3 1. Broadcast its current state;
4 2. Receive multiple states from the other agents (including its own), never waiting for more than $n - n_f$ states since n_f states might have crashed, where n is the total number of agents and n_f is the upper bound for the number of Byzantine agents in the network;
5 3. Update its current state to a particular point inside a "safe area" in $\mathbb{R}^d$, guaranteed to be in the convex hull of the normal inputs, where d is the dimension of the states.

It needs to be assumed in Algorithm 1 that communication channels are point-to-point reliable (all messages are eventually delivered), complete (any pairwise communication is possible), and FIFO (first-in, first-out). The agents can reliably identify the sender of any message. Besides, agents use two communication primitives, namely the reliable broadcast and the witness technique [28], to support the communication among agents, such that

- the *reliable broadcast* technique eliminates the situation when Byzantine agents convey different messages to different normal agents, and
- the *witness technique* results in normal agents having $n - n_f$ common states.

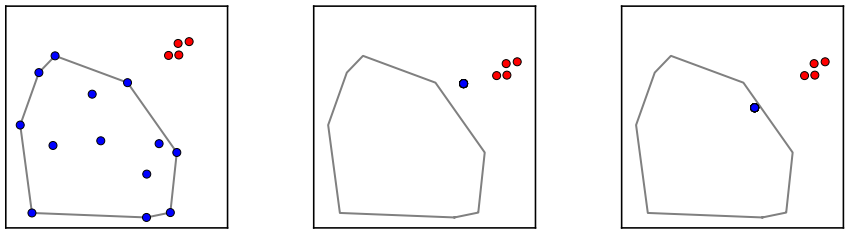
The two techniques are essential for the correctness of the algorithm to ensure the unique consensus point. The details of the two techniques as well as how they guarantee the correctness of the resilient asynchronous approximate vector consensus algorithm in Algorithm 1 are thoroughly discussed in [28].

The only difference between the above algorithm for asynchronous approximate vector consensus and the iterative algorithm for synchronous approximate vector consensus is in the second step where agents never wait for more than $n - n_f$ states in the asynchronous system yet wait for n total states in the synchronous system, which results in a different resilience bound between the two systems [23]. In particular, with complete communication graphs, the necessary and sufficient condition to solve the resilient approximate vector consensus problem is (1) $n > (d + 1)n_f$, in synchronous systems; and (2) $n > (d + 2)n_f$ in asynchronous systems [23].

3.6.2 Iterative Algorithms Using Centerpoints

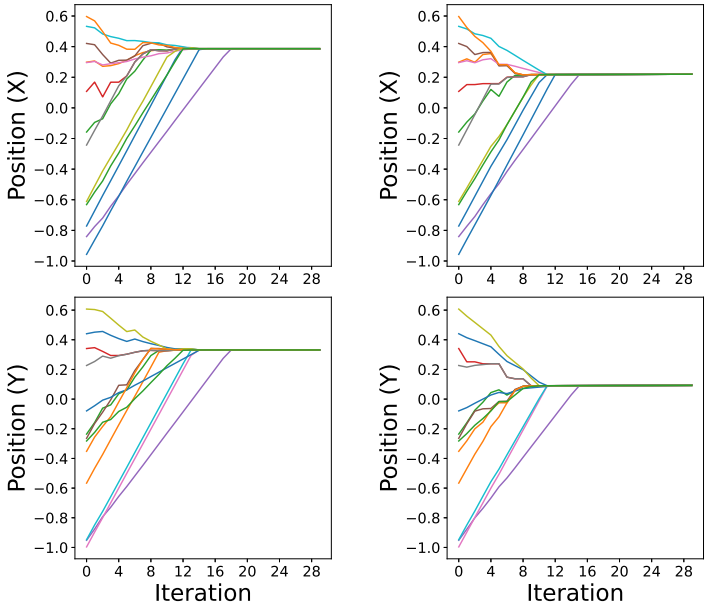
The insight of our method lies in step 3 of Algorithm 1, where we compute a centerpoint of $n_{acp} = n - n_f$ accepted states as a point in the "safe area". As we discuss in Section 3.4, this point is guaranteed to be inside the convex hull of the normal inputs (safety condition) if the number of Byzantine points in n_{acp} accepted points is upper bounded by $\lceil \frac{n_{acp}}{d+1} \rceil - 1$. Since n_{acp} accepted points may contain all the Byzantine points, it follows that the safety condition is guaranteed if $n_f \leq \lceil \frac{n-n_f}{d+1} \rceil - 1$, which is equivalent to $n - n_f \geq (n_f + 1 - 1)(d+1) + 1$. Rearranging the terms and it yields that $n \geq (d+2)n_f + 1$, which is consistent to the results in [23]. Therefore, using centerpoint for computing a point in the "safe area" in step 3 of Algorithm 1, we could achieve resilient approximate vector consensus for asynchronous systems if $n \geq (d+2)n_f + 1$. Note that the difference between the resilience bounds for synchronous and asynchronous systems results from the different number of accepted states for aggregation.

Compare our method with the method used in [23, 28] for computing a point in the safe area. In [23, 28], they use linear programming for deterministically obtain a point inside the safe area. The linear program uses a total of $\binom{n}{n-n_f}(d+1+n-n_f)$ constraints in $d + \binom{n}{n-n_f}(n-n_f)$ variables. It can be found that the linear program cannot be solved in polynomial time for $n_f = \lceil \frac{n}{d+2} \rceil - 1$ with the number of variables and constraints that are not polynomial in n. However, using centerpoints, we observe $O(n)$ and $O(n^2)$ computation time for $d = 2$ and $d = 3$ respectively as we discussed in 3.4. Note that $d \leq 3$ in many practical applications. Hence, the centerpoint-based method for computing a point in the safe area offers more advantages in terms of computational complexity and characterization.



(a) (b) (c)

Figure 3.6: Asynchronous approximate vector consensus: (a) Initial positions of robots. (b) Final positions of robots using approximate Tverberg partition based algorithm. (c) Final positions using centerpoint based algorithm.



(a) (b)

Figure 3.7: Asynchronous approximate vector consensus: positions of normal robots as a function of iterations using (a) approximate Tverberg partition based, and (b) centerpoint based algorithms.

3.6.3 Evaluation

We perform simulations[8] to compare resilient asynchronous approximate vector consensus in multirobot systems using Algorithm 1 in two dimensions with centerpoint and approximate Tverberg partition [32] for computing a point in the "safe area". To realize asynchronous communication, we simulate each agent as a thread with no global clock. In our simulation, we do not implement the reliable broadcast and witness technique in order to make the Byzantine agents convey the same messages to all the normal agents and all the normal agents have $n - n_f$ common messages. However, we achieve the same purpose by forcing the Byzantine agents broadcast the same messages to all the other agents and assigning time sequence of agents for broadcasting and receiving messages at each iteration.

Following Algorithm 1, at each iteration t of the multi-robot consensus algorithm, a normal robot i broadcasts its current position, computes a safe point $s_i(t)$ of the first $n - n_f$ positions they receive from the other agents (using centerpoint or approximate Tverberg partition), and calculates its new position using (3.1). In our experiments, we set $\alpha_i(t) = 1$. We consider non-stationary Byzantine nodes, such that each Byzantine node changes its position by moving right, down, left, and up with length 0.2 constantly for four consecutive iterations starting from the first iteration. We assume that the network graph is modeled by a complete graph. A group of 17 robots of which 4 are adversarial is distributed in a planar region $\mathcal{W} = [-1, 1] \times [-1, 1] \in \mathbb{R}^2$

[8]Our code is available at https://github.com/JianiLi/asyncConsensus

as shown in Figure 3.6(a), where blue nodes are normal agents and red nodes are Byzantine agents, and the gray polygon is the convex hull of the initial positions of the normal agents. From the discussion in Section 3.6.2, using centerpoints in the third step of Algorithm 1, we achieve resilient asynchronous approximate vector consensus if $n \geq (d+2)n_f+1$. Since $17 \geq (2+2)\times 4+1$, the resilient asynchronous approximate vector consensus is guaranteed by the centerpoint based algorithm. Figures 3.6(b) and (c) show the final positions of robots for consensus using approximate Tverberg partition and centerpoint, respectively. It is clear that the resilient consensus is achieved with the centerpoint based algorithm, where agents end up in a common position within the convex hull of their initial positions. However, robots fail to converge at a common point in the convex hull of their initial positions using the approximate Tverberg partition based algorithm. Figure 3.7 illustrates positions of robots as a function of iterations and demonstrates the same results.

3.7 Conclusion

The task of ensuring consensus in the convex hull of vector states for a set of nodes, a fraction of which may be under the influence of an adversary, is a challenging problem. In this chapter, we present a geometric characterization of an optimal point, in the form of a centerpoint, in the convex hull of normal nodes when the number of adversarial nodes is limited to a $1/(d+1)$ fraction of the size of the neighborhood of a node. It also follows that the upper bound on the number of adversarial nodes is best possible in the worst case. We propose to use well-known efficient algorithms to compute exact centerpoints in two and three dimensions. For higher dimensions, we use an approximate centerpoint algorithm proposed in [144], and improve previous bound on the number of adversarial nodes.

It follows from our results that if the fraction of adversarial nodes in the neighborhood of every normal node is at least $d/(d+1)$, then adversarial nodes can make normal nodes converge to any point they desire in the synchronous systems. At the same time, if their fraction is less than $1/(d+1)$, then the resilient consensus is guaranteed. In addition, in the asynchronous systems, this bound is reduced to $1/(d+2)$. In future, we aim to study and characterize the effect of adversarial nodes if their fraction in the neighborhood of a normal node is between the above two numbers.

Chapter 4

Byzantine Resilient Distributed Learning in Multi-Robot Systems Using Centerpoints [1]

Distributed machine learning algorithms are increasingly used in multi-robot systems and are prone to Byzantine attacks. In this chapter, we consider a distributed implementation of the Stochastic Gradient Descent (SGD) algorithm in a cooperative network, where networked agents optimize a global loss function using SGD on the local data and by aggregating the estimates of immediate neighbors. Byzantine agents can send arbitrary estimates to their neighbors, which may disrupt the convergence of normal agents to the optimum state. We show that if every normal agent combines its neighbors' estimates (states) such that the aggregated state is in the convex hull of its normal neighbors' states, then the resilient convergence is guaranteed. To assure this sufficient condition, we propose a resilient aggregation rule based on the notion of *centerpoint*, which is a generalization of the median in the higher-dimensional Euclidean space. We evaluate our results using examples of target pursuit and pattern recognition in multi-robot systems. The evaluation results demonstrate the cases where distributed learning with the average, coordinate-wise median, and geometric median-based aggregation rules fail to converge to the optimum state, whereas the centerpoint-based aggregation rule is resilient in the same scenario.

4.1 Introduction

There is a growing trend towards collaboratively training machine learning models on distributed devices to deal with the rapid increase of data and privacy and security concerns. In this chapter, we consider the problem of distributed machine learning (DML) in a fully decentralized network [69, 70, 145]. In such a network, agents interact with each other without a centralized server and leverage the shared information to benefit their learning performance. Such a decentralized framework also addresses the single point of failure problem as well as scalability issues and is naturally suited for applications in multi-robot systems, including spectrum sensing in cognitive networks [146], target localization and tracking [147], distributed clustering [148], and biologically inspired designs for mobile networks [149].

Although cooperation among agents helps improve the overall learning performance [69], it is also susceptible to attacks where non-cooperative or adversarial neighbors sharing wrong information can disrupt the

[1] ©2020 RSS. Adapted with permission, from [Jiani Li, Waseem Abbas, Mudassir Shabbir, and Xenofon Koutsoukos. "Resilient Distributed Diffusion for Multi-Robot Systems Using Centerpoint". In: Proceedings of Robotics: Science and Systems (RSS). Corvalis, Oregon, USA, July 2020. DOI: 10.15607/RSS.2020.XVI.021].

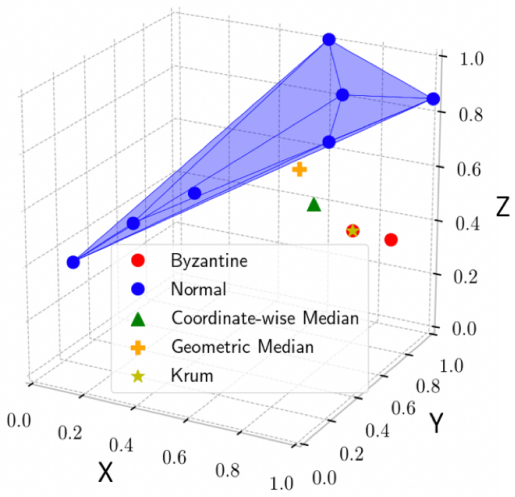


Figure 4.1: Aggregating 9 points including 2 Byzantine points using different aggregation rules: all the aggregated results fall outside of the convex hull (blue polygon) of the normal points.

convergence of the algorithm. Average-based information aggregation rules have been widely used in DML [69–71]; however, a single misbehaving agent can adversely impact the convergence of the average-based aggregation rules [1, 72]. Many robust aggregation rules have also been proposed to cope with outliers in data or Byzantine adversaries, including the coordinate-wise median [52, 63, 64], coordinate-wise trimmed mean [52, 57], geometric median [58, 62], Krum and multi-Krum [72], Bulyan, and multi-Bulyan [76, 77]. However, studies have already reported successful attacks in Byzantine systems using the rules mentioned above [78–80].

In this chapter, we study the problem of the resilient convergence of DML for multi-robot systems in the presence of Byzantine adversaries. We show that in the aggregation step, if every normal agent in the network combines its neighbors' states such that the aggregated result is in the convex hull of *normal* neighbors' states, then the resilient convergence of DML is guaranteed. We observe that most of the aggregation rules used in the literature do not satisfy this condition, as illustrated in Figure 4.1. The primary challenge here is that an agent cannot distinguish between its normal or Byzantine neighbors; hence, it cannot simply discard the information from Byzantine neighbors to satisfy the above condition. To address this, we propose a resilient aggregation rule based on the notion of *safe region*. For a normal agent k with n_k neighbors, of which *any* f can be Byzantine agents, a safe region is the set of states that always lie in the convex hull of agent k's normal neighbors' states. We show that a normal agent can always find a state in the safe region by computing a *centerpoint* of its neighbors' states if $f \leq \lceil \frac{n_k}{d+1} \rceil - 1$, where d is the dimension of states. Thus, the centerpoint-based aggregation guarantees the resilient convergence of DML in the Byzantine system.

Our main contributions are:

- We analyze the sufficient condition to achieve Byzantine resilient convergence in distributed learning algo-

rithms. The condition guarantees that the state obtained due to the aggregation step lies inside the convex hull of the normal agents' states. When the sufficient condition is satisfied, we show that normal agents converge to the global optimum state with $O(1/i)$ convergence rate using appropriate stepsizes, where i denotes the time index.

- We propose a centerpoint-based aggregation rule and show that it guarantees the resilient convergence of the distributed learning algorithms whenever each normal agent k in the network has $f \leq \lceil \frac{n_k}{d+1} \rceil - 1$ Byzantine neighbors.

- We evaluate our results using the examples of target pursuit and pattern recognition in multi-robot systems. We compare the proposed centerpoint-based aggregation rule with the average, coordinate-wise median, and geometric median-based rules. The simulation results show that our approach is resilient to $\lceil \frac{n_k}{d+1} \rceil - 1$ Byzantine neighbors, and the cooperation improves the average learning performance over the network than the non-cooperative case, while the other approaches are not resilient in the same scenarios.

The rest of the chapter is organized as follows: Section 4.2 discusses the related work. Section 4.3 formulates the resilient distributed learning problem. Section 4.4 analyzes the sufficient condition to achieve resilient convergence in distributed learning. Section 4.5 introduces the resilient aggregation rule based on the centerpoint, which satisfies the sufficient condition and further guarantees the resilient convergence in distributed learning. Section 4.6 gives an evaluation of the results. Finally, Section 4.7 provides a discussion of the proposed method and concludes the chapter.

4.2 Related Work

Approximate Byzantine Consensus. The approximate Byzantine consensus problem initiated in [30] is widely studied in the robotics and control systems community and is very relevant to resilient distributed learning and optimization. The main objective is to ensure that all normal agents in a network satisfy the *safety* and *agreement* conditions in the presence of Byzantine agents [22, 28]. Safety condition requires normal agents to update their states such that they are always inside the convex hull of normal agents' initial states. Agreement means that eventually, all normal agents' states are very close to each other, that is, within an arbitrary $\epsilon > 0$ distance from one another. The Mean Subsequence Reduced (MSR) algorithms [17, 150, 151] and the median-based algorithms [31, 39, 40] were proposed for the approximate Byzantine consensus over scalar states. The problem is more challenging when the state vectors are in $\mathbb{R}^d$ where $d \geq 2$. For vector consensus, Tverberg partition [22, 33, 44], safe area [28], and centerpoint-based [3] approaches have been proposed. The resilient vector consensus has various applications in multi-robot systems for fault-tolerant

rendezvous [152], formation control [41], flocking [42], secure localization [153, 154], and target pursuit [4]. Some of these works rely on coordinate-wise resilient scalar consensus, which does not necessarily achieve resilient vector consensus. Note that in the resilient consensus problem, the connectivity and robustness of the underlying network play an important role in the convergence of the iterative algorithms. In order to achieve resilient consensus, the underlying network should satisfy certain robustness conditions that in turn guarantee the redundant information needed by agents to ensure consensus in the presence of adversarial agents [17, 151]. Different than the consensus problem, in distributed learning, agents learn to converge to their target using the adaptation step in addition to the aggregation of the neighbors' states. We will show in Section 4.4 that the normal agents converge towards the optimum state as long as the safety condition is satisfied in the aggregation step of distributed learning, regardless of the robustness of the underlying, which is different from the case of resilient consensus.

Resilient Aggregation in DML. To achieve resilient convergence in DML, one approach is to discard cooperation with possible Byzantine neighbors using the idea of trimming, similar to the MSR approach used in resilient scalar consensus. In such an approach, it is assumed that a maximum of f Byzantine agents can be present in the neighborhood of a normal agent. Algorithms are then designed for a normal agent to rank its neighbors based on some trust criteria and a normal agent discards the values from its f least trusted neighbors. Various metrics have been proposed in the literature to evaluate a agent's trustworthiness, including metrics based on the product of the weight and the loss [1], model parameters [57], gradients and their norms [54, 74], and a combination of gradient and model parameters [75]. Moreover, various majority-based aggregation rules have been proposed, similar to the median-based approach used in resilient scalar consensus. Well-known majority-based aggregation rules include the coordinate-wise median [52, 63, 64], coordinate-wise trimmed mean [52, 57], geometric median [58, 62], Krum and multi-Krum [72], Bulyan and multi-Bulyan [76, 77]. Instead of ranking and filtering out the suspicious messages, these rules differ from the ranking methods in the way that they do not need the ranking step and the aggregation result directly precludes states far away from the cluster of the majority of the states by the intrinsic properties of the aggregation rules. However, studies have already reported all of the above-mentioned methods are not resilient to attacks under certain conditions [78–80].

Resilient DML via Computation Redundancy. One can also use computation redundancy to achieve resilient convergence in DML, which typically involves coding theory and algorithmic redundancy [81–83]. An example of such a framework is DRACO [81] in which the parameter server uses redundant gradients received from agents to eliminate the effects of adversarial updates. Another algorithm proposed recently is the RSA [84] that introduces an ℓ_p-norm regularization term into the objective function for the resilience pur-

pose. It eliminates the effect caused by the magnitudes of malicious messages sent by the Byzantine agents, as a result, only the number of Byzantine agents, but not the magnitude, influence the model update, making the algorithm robust to large outliers.

4.3 Problem Formulation

In this section, we describe the problem of distributed learning in networks in an adversarial setting.

4.3.1 Distributed Learning

Consider a network of n agents[2] modeled by an *undirected graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents agents and $\mathcal{E}$ represents interactions between agents. A bi-directional edge $(l, k) \in \mathcal{E}$ means that agents k and l can exchange information with each other. Since each agent also has its own information, we have $(k, k) \in \mathcal{E}, \forall k \in \mathcal{V}$. The *neighborhood* of k is the set $\mathcal{N}_k = \{l \in \mathcal{V} | (l, k) \in \mathcal{E}\}$. Each agent k has data $\left\{(x_k^i, y_k^i)\right\}_{i \in \mathcal{S}_k}$ sampled randomly from the distribution generated by the random variable ξ_k, where $x_k^i \in \mathbb{R}^d$, $y_k^i \in \mathbb{R}$, and $\mathcal{S}_k$ is the sample set. We consider a convex *prediction function* (model) $\varphi_k(x_k^i) = \theta_k^\top x_k^i$, where $\theta_k \in \mathbb{R}^d$ is the model parameter (or state). We use $\ell_k(\cdot)$ to denote a convex *loss function* associated with the prediction function for agent k, and $f_k(\cdot)$ to denote the convex (expected) *risk function* $f_k(\theta_k) = \mathbb{E}\left[\ell_k(\theta_k; \xi_k)\right]$.

The *objective* of the network of n agents is to estimate the parameter vector θ^* in a distributed and cooperative manner, that minimizes a global cost function of the following form:

$$\min_{\theta} \left\{ J(\theta) \triangleq \frac{1}{n} \sum_{k=1}^{n} f_k(\theta) \right\}. \tag{4.1}$$

Stochastic gradient descent (SGD) can be used to optimize the global cost function (4.1) given the stochastic gradient of $J(\theta)$. Since such a value is not available, we consider a distributed solution for each agent, known as *cooperative SGD*, which takes the following two steps in synchronized rounds of communication between agents [69]:

$$\hat{\theta}_{k,i} = \theta_{k,i-1} - \alpha_{k,i-1} \nabla \ell_k(\theta_{k,i-1}; \xi_k^{i-1}), \quad \text{(SGD)} \tag{4.2}$$

$$\theta_{k,i} = Aggr\left(\left\{\hat{\theta}_{l,i} : l \in \mathcal{N}_k\right\}\right). \quad \text{(Aggregation)} \tag{4.3}$$

In cooperative SGD, at each iteration i, agent k minimizes the individual risk using SGD given local data, followed by an aggregation step that aggregates neighboring estimates. Here, $\alpha_{k,i}$ is the stepsize, $\nabla \ell_k(\theta_{k,i-1}; \xi_k^{i-1})$ is the gradient using the instantaneous realization ξ_k^{i-1} of the random variable ξ_k, $\mathcal{N}_k$ is the neighborhood set of agent k, and $Aggr(\cdot)$ denotes an aggregation function. An example of aggregation functions is the convex

[2] We use terms *agent* and *robot* interchangeably.

combination of the neighbors' states, i.e.,

$$Aggr\left(\left\{\hat{\theta}_{l,i} : l \in \mathcal{N}_k\right\}\right) \triangleq \sum_{l \in \mathcal{N}_k} a_{lk}(i)\hat{\theta}_{l,i},$$

where $a_{lk}(i)$ denotes the weight assigned by agent k to l at iteration i, and satisfies the following:

$$a_{lk}(i) \geq 0, \quad \sum_{l \in \mathcal{N}_k} a_{lk}(i) = 1, \quad a_{lk}(i) = 0 \text{ if } l \notin \mathcal{N}_k. \tag{4.4}$$

4.3.2 Byzantine Attacks and Resilient Distributed Learning

In distributed learning, the issue of resilience against Byzantine agents has received much attention recently [29, 52, 58, 62, 155]. Since Byzantine agents can send incorrect information (state values) to their neighbors, the aggregation step is susceptible to cyber-attacks. In particular, normal agents communicating with Byzantine neighbors and updating their states using the Byzantine messages in the aggregation step may converge to a point desired by the attacker [1].

We assume two types of agents in the network, normal and Byzantine. *Normal* agents are the ones that interact with their neighbors synchronously and always update their estimates according to the prescribed update rule. *Byzantine* agents are the ones that can change their states arbitrarily and do not follow the prescribed update rule. Moreover, a Byzantine agent can transmit different values to its different neighbors. Further, we assume that the identities of normal and Byzantine agents are not changing. Since Byzantine agents that stop sending messages can be easily identified in a synchronous network, we assume Byzantine agents always send messages during communication. For a normal agent k, all agents in its neighborhood are indistinguishable, that is, k cannot identify which of its neighbors are Byzantine. Further, we use the following notation:

$\mathcal{N}$	set of normal agents;
$\mathcal{F}$	set of Byzantine agents ($\lvert\mathcal{N}\rvert + \lvert\mathcal{F}\rvert = n$);
$\mathcal{N}_k$	set of neighbors of agent k;
$\mathcal{N}_k^+$	set of normal neighbors of agent k;
$\mathcal{N}_k^-$	set of Byzantine neighbors of agent k, such that $\lvert\mathcal{N}_k^+\rvert + \lvert\mathcal{N}_k^-\rvert = \lvert\mathcal{N}_k\rvert$;
f	upper bound on the number of Byzantine neighbors of a normal agent, i.e., $\lvert\mathcal{N}_k^-\rvert \leq f$.

In the presence of Byzantine agents, the objective of the normal agents should be rewritten as follows:

$$\min_{\theta} \left\{ F(\theta) \triangleq \frac{1}{|\mathcal{N}|} \sum_{k \in \mathcal{N}} f_k(\theta) \right\}. \tag{4.5}$$

We make the following assumption about the global objective function in (4.5).

Assumption 4.1. (Strong convexity) The global objective function F is strongly convex in that there exists a constant $c > 0$ such that

$$F(y) \geq F(x) + \nabla F(x)^\top (y - x) + \frac{c}{2} \|y - x\|^2, \forall x, y.$$

Hence, F has a unique minimizer, denoted as θ^* with $F^* \triangleq F(\theta^*)$.

This chapter aims to address the problem of resilient distributed learning in the presence of Byzantine agents. The goal is to ensure that all the normal agents in the network using the cooperative SGD algorithm to optimize (4.5) achieve *resilient convergence*, formally stated below.

Definition 4.1. (Resilient Convergence) *The network is said to achieve resilient convergence if*

$$\lim_{i \to \infty} \mathbb{E} \left[\|\theta_{k,i} - \theta^*\|^2 \right] = 0, \forall k \in \mathcal{N}, \tag{4.6}$$

thereby ensuring that all normal agents converge to the globally optimum state in expectation. Here, θ^* is the minimizer of (4.5).

4.4 Resilient Distributed Learning

In this section, we propose a sufficient condition to guarantee the resilient convergence of the cooperative SGD algorithms. We also discuss the possible outcome when this condition is not satisfied. Later in Section 4.5, we propose an aggregation rule that satisfies the sufficient condition, which further guarantees the resilient convergence of the cooperative SGD.

Sufficient condition. *At each iteration $i \in \mathbb{N}$ and for every normal agent $k \in \mathcal{N}$, the outcome of the aggregation step $\theta_{k,i}$ is a convex combination of the estimates of the normal neighbors of k, i.e.,*

$$\begin{aligned} \theta_{k,i} &= \sum_{l \in \mathcal{N}_k^+} a_{lk}(i) \hat{\theta}_{l,i}, \forall i \in \mathbb{N}, k \in \mathcal{N}, \\ \text{s.t. } a_{lk}(i) &\geq 0, \forall l \in \mathcal{N}_k^+, \text{ and } \sum_{l \in \mathcal{N}_k^+} a_{lk}(i) = 1. \end{aligned} \tag{4.7}$$

In other words, at each iteration i, a normal agent k aggregates its neighbors' estimates such that the output of the aggregation $\theta_{k,i}$ is in the *convex hull of normal neighbors' state estimates* regardless of the estimates from Byzantine neighbors[3].

Next, we prove the resilient convergence when the sufficient condition is satisfied (Theorem 4.1). To facilitate the analysis, we list the following assumptions and lemma.

Assumption 4.2. (Lipschitz-continuous gradients) The global objective function F is continuously differentiable and ∇F is Lipschitz continuous with Lipschitz constant $L > 0$, i.e.,

$$\|\nabla F(x) - \nabla F(y)\| \leq L\|x - y\|, \forall x, y.$$

Assumption 4.3. (First and second moment limits) The objective function F and the sequence of $\theta_{k,i}$ for $k \in \mathcal{N}$ and $i \in \mathbb{N}$, obtained by implementing the cooperative SGD algorithm satisfy the following:

(i) $F(\theta_{k,i}) \leq F_{\text{inf}}$ for some scalar F_{inf}.

(ii) There exist scalars $\mu_G \geq \mu > 0$ such that,

$$\nabla F(\theta_{k,i})^\top \mathbb{E}_{\xi_k^i}[\nabla \ell_k(\theta_{k,i}; \xi_k^i)] \geq \mu \|\nabla F(\theta_{k,i})\|^2 \text{ and}$$

$$\|\mathbb{E}_{\xi_k^i}[\nabla \ell_k(\theta_{k,i}; \xi_k^i)]\| \leq \mu_g \|\nabla F(\theta_{k,i})\|, \forall k \in \mathcal{N} \text{ and } i \in \mathbb{N}.$$

(iii) There exist scalars $M_k \geq 0$ and $V_k \geq 0$ such that

$$\mathbb{E}_{\xi_k^i}[\|\nabla \ell_k(\theta_{k,i}; \xi_k^i)\|^2] - \|\mathbb{E}_{\xi_k^i}[\nabla \ell_k(\theta_{k,i}; \xi_k^i)]\|^2 \leq M_k + V_k \|\nabla F(\theta_{k,i})\|^2, \forall k \in \mathcal{N} \text{ and } i \in \mathbb{N}.$$

Lemma 4.1. *Under Assumptions 1-3 and stepsize* $0 < \alpha_{k,i} \leq \frac{\mu}{LG_k}$*, where* $G_k \triangleq V_k + \mu_g^2$*, for all* $k \in \mathcal{N}$*,* $i \in \mathbb{N}$*, the iterates of SGD (step* (4.2)*) satisfy the following inequalities:*

$$\mathbb{E}[F(\hat{\theta}_{k,i+1}) - F^*] \leq (1 - \alpha_{k,i} c\mu)\mathbb{E}[F(\theta_{k,i}) - F^*] + \frac{1}{2}\alpha_{k,i}^2 L M_k.$$

Proof. Our proof is based on the convergence proof of SGD (Theorem 4.6) in [156]. Since F has an L-

[3]The convex hull of a set of points $S = \{p_1, p_2, \ldots, p_n\}$ in $\mathbb{R}^d$ is the smallest convex set containing S. Any point p inside the convex hull of S has the property that $p = \sum_{k=1}^n \lambda_k p_k$, where $0 \leq \lambda_k \leq 1$ and $\sum_{k=1}^n \lambda_k = 1$. And no point outside of the convex hull has such representation.

Lipschitz continuous gradient, it holds that

$$F\left(\hat{\theta}_{k,i+1}\right) - F(\theta_{k,i}) \leq \nabla F(\theta_{k,i})^\top (\hat{\theta}_{k,i+1} - \theta_{k,i}) + \frac{1}{2} L \|\hat{\theta}_{k,i+1} - \theta_{k,i}\|^2.$$

Given the SGD step (4.2), we have

$$F\left(\hat{\theta}_{k,i+1}\right) - F(\theta_{k,i}) \leq -\alpha_{k,i} \nabla F(\theta_{k,i})^\top \nabla \ell_k(\theta_{k,i}; \xi_k^i) + \frac{1}{2} \alpha_{k,i}^2 L \|\nabla \ell_k(\theta_{k,i}; \xi_k^i)\|^2. \tag{4.8}$$

Take the expected value of the above equation with respect to the random variable ξ_k^i. Since $\hat{\theta}_{k,i+1}$ depends on ξ_k^i, whereas $\theta_{k,i}$ does not, we obtain

$$\mathbb{E}_{\xi_k^i}\left[F\left(\hat{\theta}_{k,i+1}\right)\right] - F(\theta_{k,i}) \leq \quad -\alpha_{k,i} \nabla F(\theta_{k,i})^\top \mathbb{E}_{\xi_k^i}[\nabla \ell_k(\theta_{k,i}; \xi_k^i)] + \frac{1}{2} \alpha_{k,i}^2 L \mathbb{E}_{\xi_k^i}[\|\nabla \ell_k(\theta_{k,i}; \xi_k^i)\|^2].$$

Given Assumption 4.3, it follows that

$$\begin{aligned}
&\mathbb{E}_{\xi_k^i}\left[F\left(\hat{\theta}_{k,i+1}\right)\right] - F(\theta_{k,i}) \\
\leq& -\mu \alpha_{k,i} \|\nabla F(\theta_{k,i})\|^2 + \frac{1}{2} \alpha_{k,i}^2 L \mathbb{E}_{\xi_k^i}[\|\nabla \ell_k(\theta_{k,i}; \xi_k^i)\|^2] \\
\leq& -\mu \alpha_{k,i} \|\nabla F(\theta_{k,i})\|^2 + \frac{1}{2} \alpha_{k,i}^2 L \left(M_k + (V_k + \mu_g^2) \|\nabla F(\theta_{k,i})\|^2\right) \\
=& -\left(\mu - \frac{1}{2} \alpha_{k,i} L G_k\right) \alpha_{k,i} \|\nabla F(\theta_{k,i})\|^2 + \frac{1}{2} \alpha_{k,i}^2 L M_k,
\end{aligned} \tag{4.9}$$

where $G_k \triangleq V_k + \mu_g^2$.

Given that F is strongly convex, there exists $0 < c \leq L$ such that

$$\|\nabla F(\theta)\|^2 \geq 2c\,(F(\theta) - F^*) \text{ for all } \theta.$$

Also, since $\alpha_{k,i} \leq \frac{\mu}{L G_k}$, it holds that $\alpha_{k,i} L G_k \leq \mu$. Following (4.9), We have

$$\mathbb{E}_{\xi_k^i}\left[F\left(\hat{\theta}_{k,i+1}\right)\right] - F(\theta_{k,i}) \leq -\frac{1}{2} \alpha_{k,i} \mu \|\nabla F(\theta_{k,i})\|^2 + \frac{1}{2} \alpha_{k,i}^2 L M_k \leq -\alpha_{k,i} c \mu \left(F(\theta_{k,i}) - F^*\right) + \frac{1}{2} \alpha_{k,i}^2 L M_k$$

Subtracting F^* from both sides of (4.9) and taking total expectations over the joint distribution $\xi_{k,i}$ for all $k \in \mathcal{N}$, $i \in \mathbb{N}$, we have

$$\mathbb{E}[F(\hat{\theta}_{k,i+1}) - F^*] \leq (1 - \alpha_{k,i} c \mu) \mathbb{E}[F(\theta_{k,i}) - F^*] + \frac{1}{2} \alpha_{k,i}^2 L M_k,$$

which completes the proof. □

Denote $\Delta_{k,i} \triangleq \mathbb{E}\left[F\left(\theta_{k,i}\right) - F(\theta^*)\right]$ as the expected optimality gap. Using the cooperative SGD algorithm satisfying the sufficient condition, $\Delta_{k,i}$ can be bounded as given in Theorem 4.1, which guarantees its resilient convergence as given in Proposition 4.1.

Theorem 4.1. *If the sufficient condition* (4.7) *and Assumptions 1-3 are satisfied, and all normal agents implement the cooperative SGD algorithm with the same diminishing stepsize sequence $\alpha_{k,i} = \alpha_i$ given by*

$$\alpha_i = \frac{\beta}{\gamma + i} \text{ for some } \beta > \frac{1}{c\mu} \text{ and } \gamma > 0 \text{ s.t. } 0 < \alpha_1 \leq \frac{\mu}{L\bar{G}_k}, \tag{4.10}$$

where $\bar{G}_k \triangleq \max_{k\in\mathcal{N}} G_k$, then for all $i \in \mathbb{N}, k \in \mathcal{N}$, the expected optimality gap satisfies

$$\Delta_{k,i} \leq \frac{\nu}{\gamma + i}, \quad \text{where } \nu = \max\left\{\frac{\beta^2 L}{2(\beta c\mu - 1)} \max_{l\in\mathcal{N}} M_l, (\gamma+1)\Delta_{k,1}\right\}. \tag{4.11}$$

Proof. Given (4.7) and the convexity of F, using Jensen's inequality, we have

$$F(\theta_{k,i}) \leq \sum_{l\in\mathcal{N}_k^+} a_{lk}(i) F(\hat{\theta}_{l,i}). \tag{4.12}$$

Given (4.10), $\alpha_{k,i} = \alpha_i \leq \alpha_1 \leq \frac{\mu}{L\bar{G}_k} \leq \frac{\mu}{LG_k}$, for all $k \in \mathcal{N}$. Thus, following Lemma 4.1, it holds that

$$\Delta_{k,i+1} \leq \sum_{l\in\mathcal{N}_k} a_{lk}(i)\left[(1 - \alpha_i c\mu)\Delta_{l,i} + \frac{1}{2}\alpha_i^2 L M_l\right]. \tag{4.13}$$

The following proof is based on the convergence proof of SGD (Theorem 4.7) in [156]. When $i = 1$, it is obvious that (4.11) holds given the definition of ν. We now prove (4.11) by induction. Assume (4.11) holds for some $i \geq 1$, then it follows from (4.13) that

$$\begin{aligned}
\Delta_{k,i+1} \leq & \sum_{l\in\mathcal{N}_k} a_{lk}(i)\left[(1 - \frac{\beta c\mu}{\hat{i}})\frac{\nu}{\hat{i}} + \frac{\beta^2 L M_l}{2\hat{i}^2}\right] \\
= & \sum_{l\in\mathcal{N}_k} a_{lk}(i)\left[\left(\frac{\hat{i} - \beta c\mu}{\hat{i}^2}\right)\nu + \frac{\beta^2 L M_l}{2\hat{i}^2}\right] \\
= & \frac{\hat{i} - 1}{\hat{i}^2}\nu - \sum_{l\in\mathcal{N}_k} a_{lk}(i) \underbrace{\left[\frac{\beta c\mu - 1}{\hat{i}^2}\nu - \frac{\beta^2 L M_l}{2\hat{i}^2}\right]}_{\text{nonnegative by the definition of } \nu} \\
\leq & \frac{\nu}{\hat{i} + 1},
\end{aligned}$$

where $\hat{i} :\triangleq \gamma + i$, and the last inequality follows since $\hat{i}^2 \geq (\hat{i}+1)(\hat{i}-1)$, which completes our proof. □

Remark 4.2. *It immediately follows from Theorem 4.1 that normal agents achieve $O(1/i)$ convergence rate since $\mathbb{E}[F(\theta_{k,i}) - F(\theta^*)] \leq \frac{\nu}{\gamma+i}$ with constants $\nu > 0$ and $\gamma > 0$.*

Proposition 4.1. *If the sufficient condition* (4.7) *and Assumptions 1-3 are satisfied, and all normal agents implement the cooperative SGD with the same diminishing stepsize sequence defined in* (4.10)*, then the network achieves resilient convergence as defined in Definition 4.1.*

Proof. Given the expected optimality gap (4.11) obtained in Theorem 4.1, and the fact that $\lim_{i\to\infty} \frac{\nu}{\gamma+i} = 0$ with constants $\nu > 0$ and $\gamma > 0$, it follows that

$$\lim_{i\to\infty} \mathbb{E}\left[F\left(\theta_{k,i}\right) - F(\theta^*)\right] = 0, \forall k \in \mathcal{N}.$$

Using the strong convexity of F, it yields that

$$\lim_{i\to\infty} \mathbb{E}\left[\|\theta_{k,i} - \theta^*\|^2\right] \leq \frac{c}{2} \lim_{i\to\infty} \mathbb{E}\left[F\left(\theta_{k,i}\right) - F(\theta^*)\right] = 0,$$

$\forall k \in \mathcal{N}$, which completes the proof. □

The above discussion establishes that if the aggregation result is a convex combination of the normal neighbors' states, then the resilient convergence is guaranteed. Now, we consider a scenario in which the condition (4.7) is not satisfied and $\theta_{k,i}$ in an aggregation step of a normal node k lies outside the convex hull of $\hat{\theta}_{j,i}$, where $j \in \mathcal{N}_k^+$. Then, it is possible that after aggregation the distance from $\theta_{k,i}$ to θ^* is larger than the distance of any of the normal neighbors' states to θ^*. This means the aggregation step indeed makes the state of the normal agent k deviate from θ^*. As the number of iterations increases, the deviation from the target state might also increase and normal agents might converge to a wrong state. We further demonstrate this in Section 4.6, where the aggregation rules, such as average, coordinate-wise median, and geometric median, do not satisfy the sufficient condition, and normal agents fail to achieve resilient convergence using such rules.

4.5 Resilient Aggregation

The sufficient condition in Section 4.4 requires that in the aggregation step, a normal agent computes a point that lies in the convex hull of points corresponding to *normal* neighbors' states. Computing such a point is a challenging task as a normal agent cannot distinguish between its normal and Byzantine neighbors. Our goal in this section will be to propose an efficient way to compute such a point in the aggregation step. For this, we will utilize the notion of *safe region* as defined below.

4.5.1 The Safe Region

Let S denote a set of n_k points, and S' be a subset of S containing exactly $(n_k - f)$ points. There are $\binom{n_k}{n_k - f}$ possibilities of S' and one such possibility corresponds to the set of normal points. Let $\mathcal{S}$ be the family of all $\binom{n_k}{n_k - f}$ possibilities of S'. If one could somehow show that the intersection of convex hulls of members of $\mathcal{S}$ is nonempty, then every point in this intersection is guaranteed to lie inside the convex hull of normal points. We call this intersection set a *safe region*.

Definition 4.2. (Safe Region) *For a set S of n_k points in $\mathbb{R}^d$, of which any f points can be Byzantine, the safe region of S is*

$$\text{Safe}_f(S) = \bigcap_{S' \subset S, |S'| = n_k - f} \text{Conv}(S'),$$

where $\text{Conv}(S')$ *denotes the convex hull of* S'.

It is immediately implied from the above definition that $\text{Safe}_f(S)$, if it exists, is always in the convex hull of the $(n_k - f)$ normal points, regardless of the selection of the f Byzantine points, as illustrated by the example in Figure 4.2. There are $n_k = 5$ points and $f = 1$, which means there are five possibilities to choose a point corresponding to a Byzantine agent. The gray shaded area is the safe region $\text{Safe}_1(S)$. We note that the safe region always lies in the convex hull of four normal points regardless of the selection of the Byzantine point, as illustrated in Figures 4.2(b)–4.2(f).

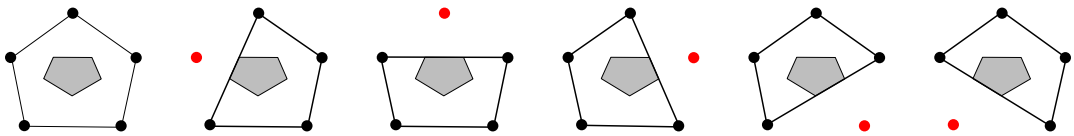


Figure 4.2: Illustration of $\text{Safe}_1(S)$ (shaded region) for a set of five points in (a). In (b)–(f), black nodes are normal, red nodes are Byzantine, and the area spanned by thick lines is the convex hull of the normal nodes.

The existence of the safe region depends on the number of Byzantine agents (points) f. For example, if $f > n_k/2$, then a safe region may not exist even in dimension $d = 1$, because two potential sets of normal points, (leftmost, and rightmost intervals of $n_k - f$ points) are non-overlapping, and the intersection will be empty. For dimension $d = 2$, if there are $n_k/3$ Byzantine points, then all possibilities for $2n_k/3$ normal points might not have a common point. In other words, no matter which point we choose for aggregation, there is always a chance that it lies outside the convex hull of normal points in $\mathbb{R}^2$. Thus, the number of allowed Byzantine points can not be more than $n/3$ in a plane. The condition of the existence of a nonempty safe region has been studied in [28] (Lemma 3.6, 3.10), which is given in the following.

Lemma 4.2. *For a set S of n_k points in $\mathbb{R}^d$, of which any f points could be Byzantine, if $f < \frac{n_k}{d+1}$, then* $\text{Safe}_f(S)$ *is necessarily nonempty; and if $f \geq \frac{n_k}{d+1}$, then* $\text{Safe}_f(S)$ *might be empty.*

Based on Lemma 4.2, the maximum number of Byzantine agents the system is resilient to is $f = \lceil \frac{n_k}{d+1} \rceil - 1$ and the safe region is nonempty in this case. For computing a point in the safe region when $f = \lceil \frac{n_k}{d+1} \rceil - 1$, we use the notion of *centerpoint*, which we explain next.

4.5.2 Centerpoint-based Resilient Vector Consensus

In the following, we define the notion of a *centerpoint*, and show that every centerpoint lies inside $\text{Safe}_f(S)$ for $f = \lceil \frac{n_k}{d+1} \rceil - 1$.

Definition 4.3. (Centerpoint) *Given a set S of n_k points in $\mathbb{R}^d$ in general positions,*[4] *where $n_k \geq d+1$, a centerpoint p is a point, not necessarily from S, such that any closed half-space*[5] *of $\mathbb{R}^d$ that contains p also contains at least $\lceil \frac{n_k}{d+1} \rceil$ points from S.*

Intuitively, a centerpoint lies in the "center region" of the set of points, in the sense that there are enough points of S on each side of a centerpoint. A centerpoint extends the notion of median to higher dimensions, and is an active topic of study in discrete geometry [3, 132, 139, 157]. Note that centerpoint is not unique, in fact, there can be infinitely many centerpoints. The set of all centerpoints constitutes the convex safe region. We have studied the connection between the centerpoint and the safe region in [3]. In particular, we have the following result.

Lemma 4.3. *Let S be a set of n_k points in $\mathbb{R}^d$, $\mathcal{C}(S)$ be the corresponding centerpoint region (set of all centerpoints) and $f = \lceil \frac{n_k}{d+1} \rceil - 1$, then*

$$\text{Safe}_f(S) \equiv \mathcal{C}(S).$$

An immediate consequence of Lemma 4.2 and Lemma 4.3 is that for a set S of n_k points in $\mathbb{R}^d$, of which any $f = \lceil \frac{n_k}{d+1} \rceil - 1$ points could be Byzantine, a centerpoint of S is always inside the convex hull of the $(n_k - f)$ normal points, regardless of the selection of the f Byzantine points.

Now that the existence of a point in the safe region for optimal number of Byzantine neighbors is guaranteed, one has to actually find such a point to aggregate to. It is easy to compute a centerpoint in lower dimensions. In two dimensions, the time complexity for computing a centerpoint is $O(n_k)$ using a prune and search algorithm in [140]. The algorithm iteratively removes a fraction of points from a given point set while ensuring that the centerpoint of the remaining points is also a centerpoint of the original point set. When

[4] A set of points in $\mathbb{R}^d$ is said to be in *general positions* if no hyperplane of dimension $d-1$ or less contains more than d points.
[5] Recall that closed half-space in $\mathbb{R}^d$ is a set of the form $\{x \in \mathbb{R}^d : a^T x \geq b\}$ for some $a \in \mathbb{R}^d \setminus \{0\}$.

the number of points becomes smaller than a constant, the algorithm uses a brute force method to compute a centerpoint [140]. In three dimensions, we can use Chan's algorithm in [142] to find a centerpoint in $O(n_k^2)$ time. [142] basically provides a randomized algorithm to some geometric linear program in $O(n_k^2)$ and uses this framework to find a geometric Tukey median of a given point set, which is guaranteed to be a centerpoint also. We have given an overview of these methods in our previous work [3]. We note that $d \leq 3$ is the case in many practical applications in robotics. For higher dimensions, i.e., $d > 3$, the time bound to compute a centerpoint is $O(n_k^{d-1})$ [142], which is impractical for very large d. However, in such cases, algorithms exist to compute an approximate centerpoint [144]. These approximations degrade the optimal bound on the number of Byzantine agents. For instance, given a set of n_k points, of which at most $\left(\frac{n_k}{d^{r/r-1}}\right)$ are Byzantine and the remaining are normal points, we can compute a point that is in the convex hull of normal points in time $O(n_k^{c \log d}(rd)^d)$, where r is any integer greater than 1, and c is some positive constant. By increasing r, the quality of approximation, and hence the bound on the number of Byzantine agents improves and approaches $\frac{n}{d}$; however, this also leads to an increase in the time complexity. Moreover, the method proposed in [158] generates approximate centerpoint in linear time complexity for any dimension. However, this will reduce the upper bound on Byzantine tolerance from $\lceil \frac{n_k}{d+1} \rceil - 1$ to $\lceil \frac{n_k}{4(d+1)^3} \rceil - 1$.

Proposition 4.2. *If $f \leq \lceil \frac{|\mathcal{N}_k|}{d+1} \rceil - 1$, Assumptions 1-3 are satisfied, and all normal agents implement the cooperative SGD with the same diminishing stepsize sequence defined in* (4.10) *using the centerpoint-based aggregation rule in the aggregation step, i.e.,*

$$Aggr\left(\left\{\hat{\theta}_{l,i} : l \in \mathcal{N}_k\right\}\right) \triangleq \mathcal{C}\left(\left\{\hat{\theta}_{l,i} : l \in \mathcal{N}_k\right\}\right),$$

then the network achieves resilient convergence as defined in Definition 4.1.

Proof. Since $f \leq \lceil \frac{|\mathcal{N}_k|}{d+1} \rceil - 1$, given Lemma 4.3 and Definition 4.2, it follows that

$$Aggr\left(\left\{\hat{\theta}_{l,i} : l \in \mathcal{N}_k\right\}\right) \in \text{Conv}(\left\{\hat{\theta}_{j,i} : j \in \mathcal{N}_k^+\right\}),$$

which satisfies the sufficient condition in (4.7). Then, the resilient convergence is guaranteed given the results provided in Proposition 4.1. □

4.6 Evaluation

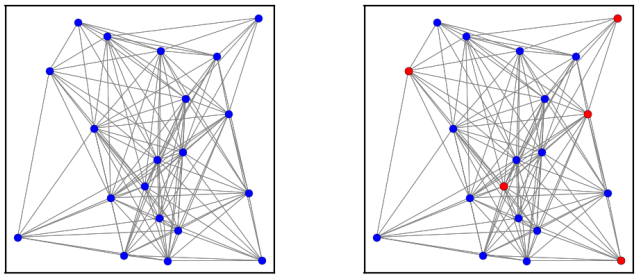
In this section, we evaluate the proposed centerpoint-based aggregation rule for the cooperative SGD algorithm using target pursuit and pattern recognition as case studies[6]. We compare it with other commonly used aggregation rules including the *average* ($a_{lk} = \frac{1}{|\mathcal{N}_k|}$ for $l \in \mathcal{N}_k$), *coordinate-wise median* (CM), and *geometric median* (GM), as well as the non-cooperative SGD (non-coop SGD). We define the coordinate-wise median and geometric median below.

- Coordinate-wise median: Let $\text{med}(\cdot)$ be the one-dimensional median, then the coordinate-wise median $\text{Median}(\cdot)$ of vectors $\{x_k \in \mathbb{R}^d, k \in [n]\}$ is defined to be $x^{\text{Med}} \triangleq \text{Median}\{x_k : k \in [n]\}$ with the j-th coordinate to be $(x^{\text{Med}})^j \triangleq \text{med}\{x_k^j : k \in [n]\}$ for each $j \in [d]$.
- Geometric median: The geometric median $\text{GM}(\cdot)$ of vectors $\{x_k \in \mathbb{R}^d, k \in [n]\}$ is defined to be $\text{GM}\{x_k : k \in [n]\} \triangleq \arg\min_{x \in \mathbb{R}^d} \sum_{k=1}^n \|x - x_k\|$.

We show in multiple cases that the cooperative SGD algorithm using centerpoint-based aggregation always outperforms the non-cooperative SGD in achieving a better average learning performance over the network at convergence, with or without the presence of Byzantine agents. However, the other rules either fail to converge to θ^* or exhibit a worse learning performance than the non-cooperative SGD, showing that such cooperation could be harmful to the overall network's performance.

4.6.1 Target Pursuit

In this example, we consider a mobile adaptive network [149] of n agents that move collectively in pursuit of a target located at $\theta^* \in \mathbb{R}^d$ that can be either static or time-varying.



(a) No attack (b) With 5 Byzantine agents

Figure 4.3: Network connectivity (blue nodes: normal agents, red nodes: Byzantine agents).

[6]Simulation code can be found at https://github.com/JianiLi/resilient_distributed_learning_centerpoint.

4.6.1.1 Background

Suppose the location of agent k at time i is denoted by $x_{k,i} \in \mathbb{R}^d$. The distance $d_k^o(i) \in \mathbb{R}$ between agent k and the target at time i can be expressed as

$$d_k^o(i) = {u_{k,i}^o}^\top(\theta^* - x_{k,i}),$$

where $u_{k,i}^o \in \mathbb{R}^d$ denotes the unit direction vector pointing from $x_{k,i}$ to θ^*. Suppose agents have only noisy observations $\{d_k(i), u_{k,i}\}$ of the distance and the unit direction vector, i.e.,

$$d_k(i) = d_k^o(i) + \eta_k^d(i), \quad u_{k,i} = u_{k,i}^o + \eta_{k,i}^u,$$

where $\eta_{k,i}^u \in \mathbb{R}^d$ and $\eta_k^d(i) \in \mathbb{R}$ denote noise terms. Let $\eta_k(i) = -{\eta_{k,i}^u}^\top(\theta^* - x_{k,i}) + \eta_k^d(i)$, $\hat{d}_k(i) = d_k(i) + u_{k,i}^\top x_{k,i}$, we have

$$\hat{d}_k(i) = u_{k,i}^\top \theta^* + \eta_k(i).$$

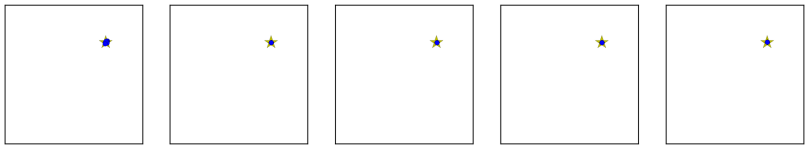
To optimize θ^*, consider

$$\min_\theta \left\{ F_{loc}(\theta) \triangleq \frac{1}{|\mathcal{N}|} \sum_{k \in \mathcal{N}} \mathbb{E}\|\hat{d}_k(i) - u_{k,i}^\top \theta\|^2 \right\}. \tag{4.14}$$

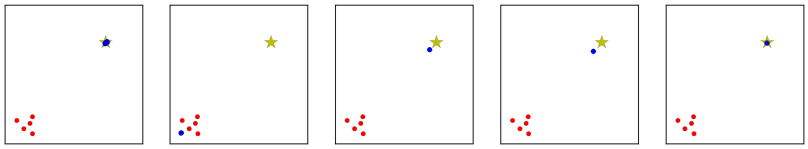
4.6.1.2 Static Target

In our simulation, we consider a network of $n = 20$ agents with $d = 2$. Figure 4.3 shows the initial deployment of agents with and without Byzantine attacks and their connectivity network. Agents are located in $[0, 1] \times [0, 1]$ region initially and agents connected with links are neighbors. The average neighborhood size is $\sum_{k=1}^n |\mathcal{N}_k|/n \approx 13.9$ and the underlying connectivity topology does not change throughout the simulation. The regression vector $u_{k,i}$ has uniform covariance matrix $R_{u,k} = \sigma_{u,k}^2 I_2$, $\sigma_{u,k}^2 \in [0, 1.0]$ where I_2 is the identity matrix of size 2. The noise variance of distance $\sigma_{d,k}^2 \in [1.0, 2.0], \forall k \in \mathcal{N}$. The time-varying stepsizes for updating location and velocity estimates are both $\alpha_{k,i} = \frac{2}{i+10}$ for $k \in \mathcal{N}$. Further, $\lambda = 0.5$, $\beta = 0.1$, $s = 1$ and $\Delta t = 0.2s$. The target location is denoted by $\theta^* = (5, 5)$. In the case of attack, we randomly select 5 agents as the Byzantine agents. For any normal agent $k \in \mathcal{N}$, it is guaranteed that the number of its Byzantine neighbors is upper bounded by $\lceil \frac{|\mathcal{N}_k|}{3} \rceil - 1$. Thus, the centerpoint-based aggregation should be resilient in such a network.

We run the non-cooperative SGD and cooperative SGD with average/CM/GM/centerpoint-based aggre-



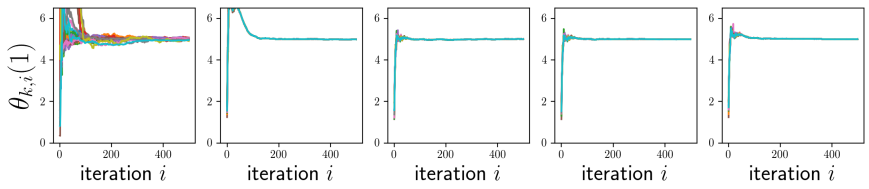
(a) No attack



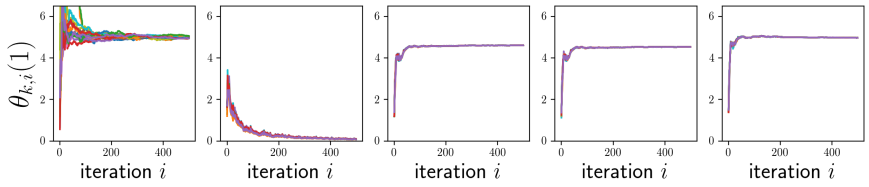
(b) With 5 Byzantine agents

Figure 4.4: Mobile network's final deployment for static target (from left to right: noncooperative SGD, cooperative SGD with average/CM/GM/centerpoint-based aggregation).

gation rules to estimate the target location $\theta_{k,i}$ and the velocity $v^g_{k,i}$. In the case of attack, Byzantine agents continuously send $(0, 0)$ to all normal agents as their current estimates of the target location and velocity. Figure 4.4 shows the final deployment of agents after 500 iterations, where the yellow star represents the target.



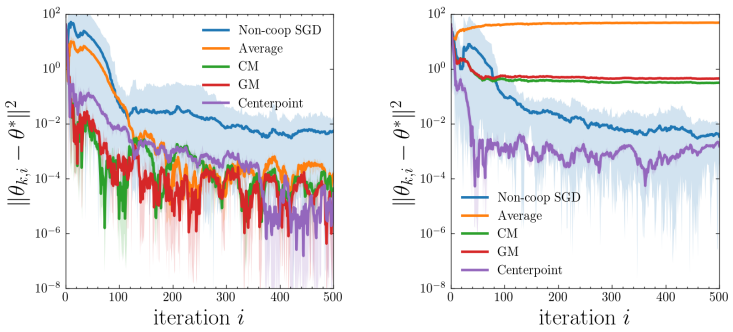
(a) No attack



(b) With 5 Byzantine agents

Figure 4.5: Static target estimates $\theta_{k,i}$ (1^{st} dimension). From left to right: noncooperative SGD, cooperative SGD with average/CM/GM /centerpoint-based aggregation .

In the case of no attack, we find all the four aggregation rules—average, CM, GM and centerpoint—converge to the target as shown in Figure 4.4(a). However, in the presence of Byzantine agents, only the centerpoint-based cooperative SGD converges to the target as shown in Figure 4.4(b). Figure 4.5 illustrates the state estimates as a function of time, where each line represents the estimates of a normal agent k. The learning accuracy (mean and range) measured by $\|\theta_{k,i} - \theta^*\|^2$, for $k \in \mathcal{N}$ is illustrated in Figure 4.6, where lines are the average values, and shaded area is the range between the minimum and the maximum values



(a) No attack (b) With 5 Byzantine agents

Figure 4.6: Estimation accuracy $\|\theta_{k,i} - \theta^*\|^2$ for $k \in \mathcal{N}$ with different aggregation rules for static target (lines are the average values, and shaded area is the range).

among the network. We observe that cooperative SGD with all the four aggregation rules achieve a better average learning accuracy at convergence (measured by $\sum_{k\in\mathcal{N}} \|\theta_{k,i} - \theta^*\|^2/|\mathcal{N}|$) than the non-cooperative SGD under no attack, whereas only the centerpoint-based aggregation achieves a better average learning accuracy than the non-cooperative SGD under attack.

4.6.1.3 Time-Varying Target

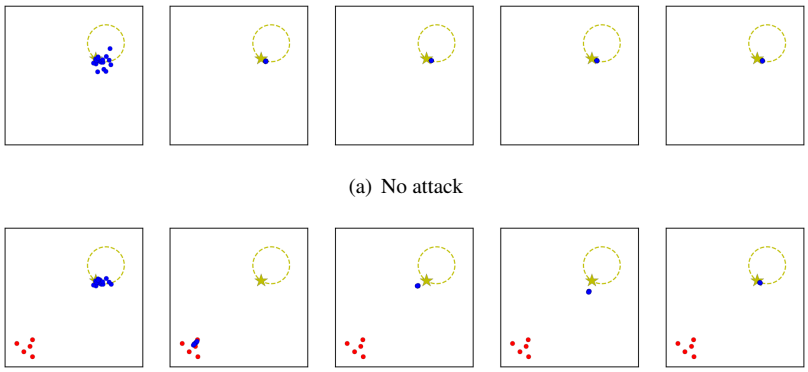
We next consider the case when the target is time-varying. Using time-varying target can make robots follow a desired trajectory, which can be used in swarm robotics. The location of the time-varying target is given by $(5 + \cos(0.01i), 5 + \sin(0.01i))$. The noise variance of distance $\sigma^2_{d,k} \in [2.0, 3.0], \forall k \in \mathcal{N}$. And we use fixed stepsize $\alpha_{k,i} = 0.05$, for $k \in \mathcal{N}$, $i \in \mathbb{N}$. The other setups and Byzantine attacks are the same as in Section 4.6.A.2.

Figure 4.7 shows the final deployment of agents after 1000 iterations, where the yellow dashed circle represents the time-varying target trajectory and the yellow star represents the current target. Figure 4.8 illustrates the state estimates as a function of time. And the learning accuracy measured by $\|\theta_{k,i} - \theta^*\|^2$ are illustrated in Figure 4.9. The simulation shows similar results to the case of static target.

4.6.1.4 Experiments on Robotarium

In addition to the numerical simulations, we carried out similar experiments using real robots on Robotarium [159], a multirobot testbed developed at the Georgia Institute of Technology. The robots are 11 cm wide, 10 cm long, and operate on a 3m x 2m area. We denote the bottom-left corner of the arena to be the original point with coordinates $(0, 0)$ and the upper-right corner to be $(3, 2)$.

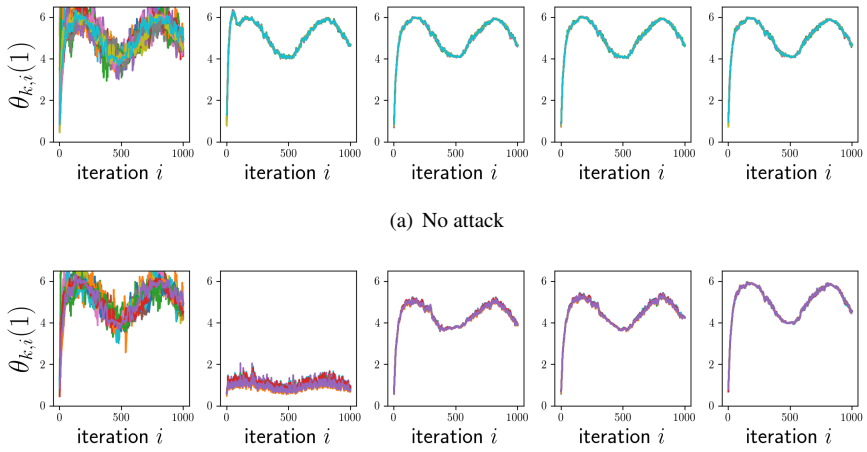
We consider a network of 11 normal robots. Parameters are selected to be $s = 1, \Delta t = 1s$. The target location is set to be $\theta^* = (2.4, 1.7)$. The regression vector $u_{k,i}$ has uniform covariance matrix $R_{u,k} = \sigma^2_{u,k} I_2$,



(a) No attack

(b) With 5 Byzantine agents

Figure 4.7: Mobile network's final deployment for time-varying target (from left to right: noncooperative SGD, cooperative SGD with average/CM/GM/centerpoint-based aggregation) .



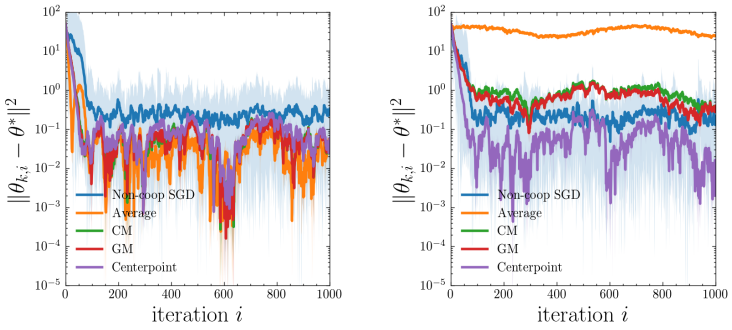
(a) No attack

(b) With 5 Byzantine agents

Figure 4.8: Time-varying target estimates $\theta_{k,i}$ (1^{st} dimension). From left to right: noncooperative SGD, cooperative SGD with average/CM/GM/centerpoint-based aggregation.

$\sigma^2_{u,k} \in [0.1, 0.5]$. The noise variance of distance $\sigma^2_{d,k} \in [0.5, 5.0]$. Both $\sigma^2_{d,k}$ and $\sigma^2_{u,k}$ decrease linearly as the distance to the target decreases. The fixed stepsize is 0.2. In the case of attack, five more Byzantine robots are introduced making the total number of robots to be 16. We consider the network to be modeled by a complete graph where every agent is the neighbor of every other agent. Since the centerpoint-based aggregation rule is resilient up to $\lceil \frac{16}{3} \rceil - 1 = 5$ Byzantine robots, we expect it to be resilient in the experiment.

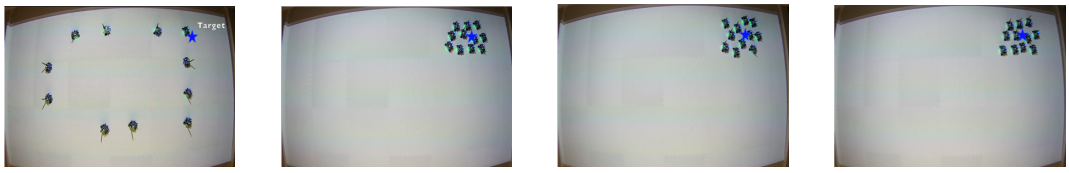
Figures 4.10 and 4.11 show the network deployments using CM/GM/centerpoint-based cooperative SGD under no attack and with attack, respectively. The Byzantine robots are indicated by the red circle, and the target location is highlighted by the blue star. Byzantine robots stay stationary throughout the experiment and continuously send wrong estimates of the target location $(0, 0)$ and velocity vector $(0, 0)$ to normal robots. We adopt the collision avoidance mechanism implemented by Robotarium in our experiment.



(a) No attack (b) With 5 Byzantine agents

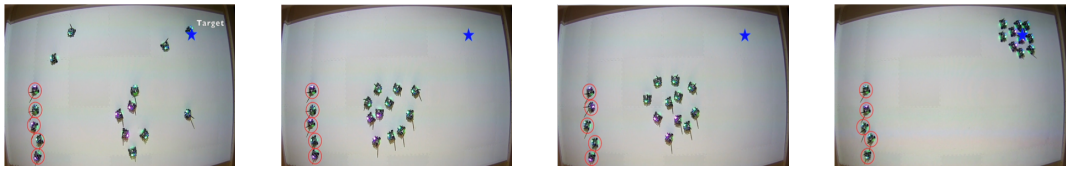
Figure 4.9: Estimation accuracy $\|\theta_{k,i} - \theta^*\|^2$ for $k \in \mathcal{N}$ with different aggregation rules, for time-varying target (lines are the average values, and shaded area is the range).

The results are similar to the simulation results. Without attacks, robots with CM/GM/centerpoint-based aggregation rules all converge to the target. However, in the presence of Byzantine agents, only robots using the centerpoint-based aggregation rule converge to the target.



(a) Initial deployment (b) Final deployment (CM) (c) Final deployment (GM) (d) Final deployment (centerpoint)

Figure 4.10: Network deployment under no attack for the multi-robot target pursuit.



(a) Initial deployment (b) Final deployment (CM) (c) Final deployment (GM) (d) Final deployment (centerpoint)

Figure 4.11: Network deployment with five Byzantine robots for the multi-robot target pursuit.

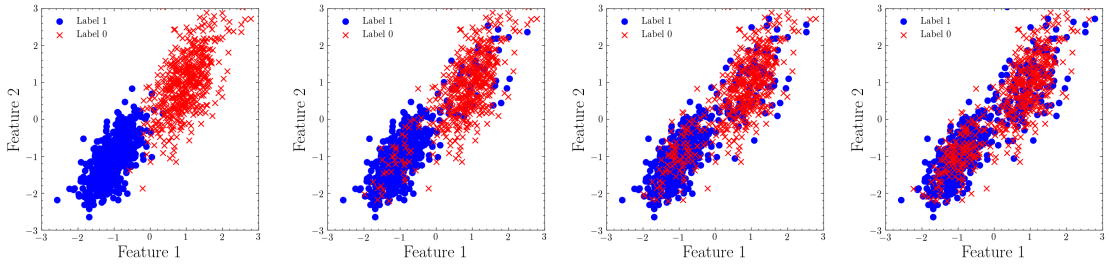
4.6.2 Pattern Recognition

In the second case study, we consider the case when robots perform a pattern recognition or detection task using sensor readings. We consider a network of 10 agents modeled by a complete graph where every agent is the neighbor of every other agent. Agents collect two-dimensional features (sensor readings) to perform binary classification. The real data distribution is given in Figure 4.12(a). The label-0 data has

mean $(1, 1)$ and covariance $((0.1468, 0.9233), (0.1863, 0.3456))$ and label-1 data has mean $(-1, -1)$ and covariance $((0.4170, 0.7203), (0.0001, 0.3023))$. We assume outliers in the training data such that the labels of the outliers are inverted – from 0 to 1 or from 1 to 0. The data distribution with $10\% - 30\%$ outliers is illustrated in Figures 4.12(b)–4.12(d). We use logistic regression to classify the data. The global cost function has the following form:

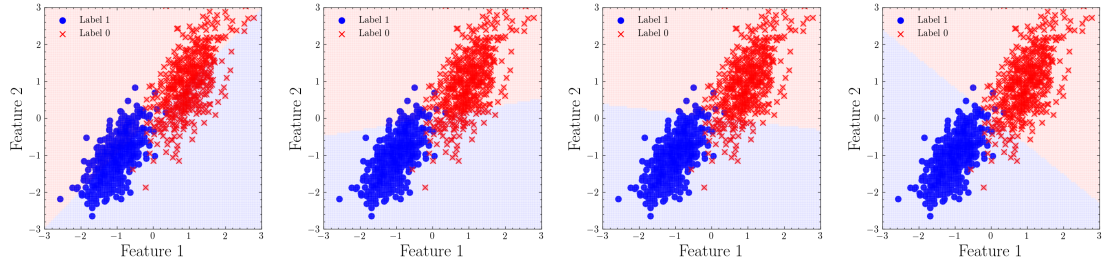
$$\min_{\theta} \left\{ F(\theta) \triangleq \frac{1}{|\mathcal{N}|} \sum_{k \in \mathcal{N}} -\mathbb{E}\left\{ y_k^i \log\left(g\left(\theta^\top x_k^i\right)\right) + \left(1 - y_k^i\right) \log\left(1 - g\left(\theta^\top x_k^i\right)\right)\right\} \right\}, \tag{4.15}$$

where $g(z) = \frac{1}{1+e^{-z}}$. The cooperative SGD algorithm in (4.2) and (4.3) can be used to optimize the above cost function. The time-varying stepsize is $\alpha_{k,i} = \frac{1}{i+10}$ for $k \in \mathcal{N}$.



(a) Real distribution (b) 10% Outliers (c) 20% Outliers (d) 30% Outliers

Figure 4.12: (a) Real data distribution, (b)-(d) data with outliers received by normal agents.



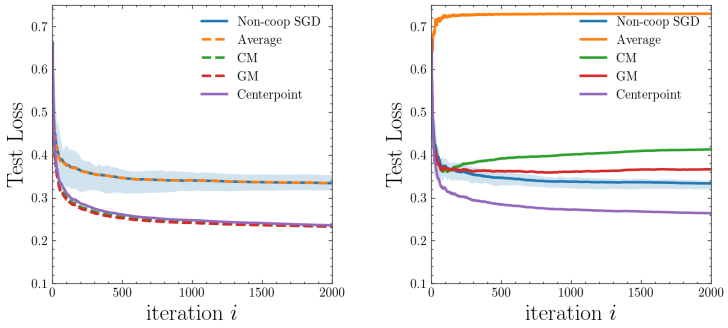
(a) Average (b) CM (c) GM (d) Centerpoint

Figure 4.13: Decision boundary achieved by different aggregation rules when 3 out of 10 agents are Byzantine.

We consider two scenarios. In the first scenario, every normal agent receives data with uniform outlier rate 20%. This simulates the case in which agents receive similar data resulting in similar learning performance. We compute the test loss of normal agents over 500 data samples from the real data distribution without outliers by (4.15). In the case of attack, we randomly pick 3 out of 10 normal agents as Byzantine agents that continuously send $(1, -1)$ as their estimates to the other normal agents. The test loss for the non-cooperative

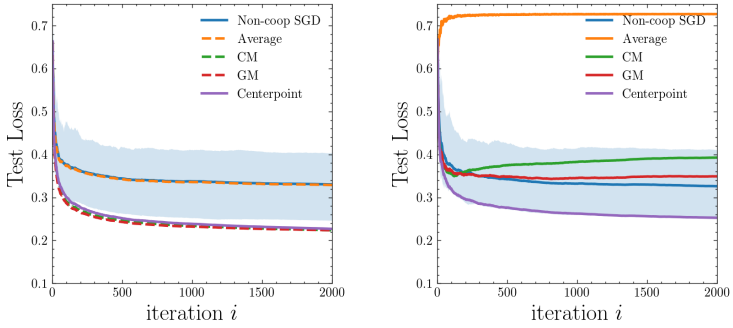
SGD, cooperation using average, CM, GM, and centerpoint is plotted in Figure 4.14. Since we consider a complete graph, normal agents receive the same messages in cooperation and therefore their aggregation results are the same. As a result, normal agents share the same test loss in the cooperative cases, which is also the mean of their test losses. We find the centerpoint-based aggregation outperforms the other cooperative aggregation rules as well as the non-cooperative SGD. When there is an attack, only the centerpoint-based aggregation converges with a better learning performance measured by the average test loss than the non-cooperative SGD. Figure 4.13 illustrates the decision boundary achieved by different aggregation rules under attack.

In the second scenario, every normal agent receives data from the real data distribution with different outlier rate of $10\% - 30\%$. This simulates the case in which agents receive different data resulting in different learning performance. Figures 4.15 illustrates the learning results. We observe that the results are similar to the previous example.



(a) No attack (b) With 3 Byzantine agent

Figure 4.14: Test loss on 500 test data samples from the real data distribution without outliers (Normal agents receive training data with uniform outlier rate 20%).



(a) No attack (b) With 3 Byzantine agents

Figure 4.15: Test loss on 500 test data samples from the real data distribution without outliers. (Normal agents receive training data with different outlier rates from 10% to 30%).

4.7 Discussion and Conclusion

The major computational step in the the proposed approach is the computation of a point in the safe region $\text{Safe}_f(S)$. We do so by computing a *centerpoint* of a set of points S in dimension d. If we have a set of N points (i.e., $|S| = N$), then a centerpoint can be computed in $O(N)$ time in $d = 2$, and $O(N^2)$ time in $d = 3$. Since in robotic applications, the position vector is in two or three dimensions, the case of centerpoint computation in $d = 2, 3$ is of particular interest. In general, the problem of checking whether a point is a centerpoint of a given set of points or not is a co-NP-complete problem. In higher dimensions ($d \geq 4$), the complexity of finding a centerpoint is unknown. However, there exist approximation algorithms and randomized algorithms that compute approximate centerpoints. We also note that a point in a safe region $\text{Safe}_f(S)$ can also be computed using other techniques, for instance, through linear programming [23]. The linear program uses a total of $\binom{n}{n-f}(d+1+n-f)$ constraints in $d+\binom{n}{n-f}(n-f)$ variables, which cannot be solved in polynomial time for $f = \lceil \frac{n}{d+1} \rceil - 1$ with the number of variables and constraints that are not polynomial in n. However, centerpoint-based computation of a point in $\text{Safe}_f(S)$ offers more advantages in terms of computational complexity and characterization.

In this work, we studied the resilient aggregation rules for distributed machine learning algorithms. We showed that the commonly used coordinate-wise median and geometric median-based aggregation methods do not guarantee resilient convergence for distributed learning. We proposed a centerpoint-based aggregation rule that generalizes the resilience property of the median into higher dimensions. The centerpoint-based aggregation rule guarantees that the distributed learning algorithms converge to the optimum state if the number of Byzantine agents in a normal agent's neighborhood is less than $\lceil \frac{n_k}{d+1} \rceil$, where n_k is the number of agents in the neighborhood, and d is the dimension of the state vector of the agents. Finally, we note that the framework and the corresponding methods and analysis can be easily generalized to federated learning. We aim to explore the trade-off between the computational cost for resilient aggregation and the degradation in learning performance for future work.

Chapter 5

Byzantine Resilient Distributed Diffusion in Least-Mean-Square (LMS) Algorithms for Multi-Task Networks [1]

In this chapter, we study resilient distributed diffusion for multi-task estimation in the presence of adversaries where networked agents must estimate distinct but correlated states of interest by processing streaming data. We show that in general diffusion strategies are not resilient to malicious agents that do not adhere to the diffusion-based information processing rules. In particular, by exploiting the adaptive weights used for diffusing information, we develop time-dependent attack models that drive normal agents to converge to states selected by the attacker. We show that an attacker that has complete knowledge of the system can always drive its targeted agents to its desired estimates. Moreover, an attacker that does not have complete knowledge of the system including streaming data of targeted agents or the parameters they use in diffusion algorithms, can still be successful in deploying an attack by approximating the needed information. The attack models can be used for both stationary and non-stationary state estimation. In addition, we present and analyze a resilient distributed diffusion algorithm that is resilient to any data falsification attack in which the number of compromised agents in the local neighborhood of a normal agent is bounded. The proposed algorithm guarantees that all normal agents converge to their true target states if appropriate parameters are selected. We also analyze trade-off between the resilience of distributed diffusion and its performance in terms of steady-state mean-square-deviation (MSD) from the correct estimates. Finally, we evaluate the proposed attack models and resilient distributed diffusion algorithm using stationary and non-stationary multi-target localization.

5.1 Introduction

Diffusion Least-Mean Squares (DLMS) is a powerful algorithm for distributed state estimation [16]. It enables networked agents to interact with neighbors to process streaming data and diffuse information across the network to perform the estimation tasks. Compared to a centralized approach, distributed diffusion offers multiple advantages including robustness to drifts in the statistical properties of the data, scalability, reliance on local data, and fast response among others. Applications of distributed diffusion include spectrum sensing in cognitive networks [160], target localization [161], distributed clustering [60], and biologically inspired

[1] ©2020 IEEE. Adapted with permission, from [Jiani Li, Waseem Abbas and Xenofon Koutsoukos, "Resilient Distributed Diffusion in Networks With Adversaries," in IEEE Transactions on Signal and Information Processing over Networks, vol. 6, pp. 1-17, 2020, doi: 10.1109/TSIPN.2019.2957731].

designs for mobile networks [162].

Diffusion strategies are known to be robust to node and link failures as well as to high noise levels [11, 163–165]. However, it is possible that a single adversarial agent that does not update its estimates according to the diffusion-based information processing rules, for instance by retaining a fixed value throughout, can fail other agents to converge to their true estimates. Resilience of diffusion-based distributed algorithms in the presence of such fixed-value Byzantine attacks has been studied in [16, 60]. A general approach to counteract such attacks is to allow agents to fuse information collected from other agents in local neighborhoods using adaptive weights instead of fixed ones. By doing so, only neighbors estimating a similar state will be assigned large weights so as to eliminate the influence of a fixed-value Byzantine adversary.

In this chapter, we consider distributed diffusion for multi-task estimation where networked agents must estimate distinct, but correlated states of interest by processing streaming data. Agents use adaptive weights when diffusing information with neighbors since adaptive weights have been successfully applied to multi-task distributed estimation problems. However, we are interested in understanding if adaptive weights introduce vulnerabilities that can be exploited by Byzantine adversaries. The first problem we consider is to analyze if it is possible for an attacker to compromise a node, and make other nodes in its neighborhood converge to a state selected by the attacker. Then, we consider a network attack and determine a minimum set of nodes to compromise to make all nodes within the network converge to attacker's desired state.

We assume a *strong attack* model, that is, the attacker has complete knowledge of the network topology, streaming data of targeted agents and their parameters used in the diffusion algorithm. A strong attacker can know the topology by monitoring the network, streaming data of agents by stealthily compromising their sensors/controllers and establishing backdoor channels, and diffusion parameters by doing reverse engineering. We note that having complete knowledge is a strong assumption, however, it is common to assume a strong attacker with complete knowledge of the system to examine the resilience of distributed networks [17, 18, 166–168]. In addition to this strong attack model, we also consider a *weak attack* model in which the attacker has no knowledge of streaming data of targeted agents or their parameters. We show that such an attacker can also be successful in preventing normal agents from converging to true estimates by approximating their states.

As a result, we show that DLMS, which was considered to be resilient against Byzantine agents by itself ([11, 16, 60]), is in fact, not resilient. A Byzantine agent sharing incorrect estimates whose values are not fixed and change over time (time-dependent Byzantine attack) can manipulate the normal agents to converge to incorrect estimates. On the one hand, adaptive weights improve the resilience of diffusion algorithms to fixed-value Byzantine attacks, but on the other hand, introduce vulnerabilities that can be exploited by time-dependent attacks. We analyze this issue in detail and propose a resilient diffusion algorithm that ensures

that normal agents converge to true final estimates in the presence of any data falsification attack. The main contributions of the chapter are summarized below:

- By exploiting the adaptive weights, we develop attack models that drive normal agents to converge to states selected by an attacker. The attack models can be used to deceive a specific node or the entire network and are applicable to both stationary and non-stationary state estimation. Although the attack models are based on a strong knowledge of the system, we also show that the attack can succeed without such knowledge.

- We propose a resilient distributed diffusion algorithm parameterized by a positive integer F. We show that if there are at most F compromised agents in the neighborhood of a normal agent, then the algorithm guarantees that normal agents converge to their actual goal states under any data falsification attack. If the parameter F selected by the normal agents is large, the resilient distributed diffusion algorithm degenerates to non-cooperative estimation. Thus, we also analyze trade-off between the resilience of distributed diffusion and its performance degradation in terms of the steady-state MSD.

- We evaluate the proposed attack models for both strong and weak attacks and the resilient distributed diffusion algorithm using both stationary and non-stationary multi-target localization. The simulation results are consistent with our theoretical analysis and show that the approach provides resilience to attacks while incurring performance degradation which depends on the assumption about the number of compromised agents.

The rest of the chapter is organized as follows: Section 5.3 briefly introduces distributed diffusion. Section 5.4 presents the attack and resilient distributed diffusion problems. Sections 5.5 and 5.6 discuss single node attack and network attack models respectively. Section 5.7 presents and analyzes the resilient distributed diffusion algorithm. Section 5.8 provides simulation results evaluating our approaches with multi-target localization. Section 5.9 discusses and evaluates the attack model that does not require complete knowledge of the system. Section 5.2 gives a brief overview of the related work and Section 5.10 concludes the chapter.

5.2 Related Work

Many distributed algorithms are vulnerable to cyber attacks. The existence of an adversarial agent may prevent the algorithm from performing the desired task. Distributed consensus and diffusion based strategies are often employed to resolve distributed estimation and optimization problems, for instance see [16, 169–173]. Resilience of consensus-based distributed algorithms in the presence of malicious nodes has received considerable attention in recent years. In particular, the approaches presented in [17, 174–176] consider the consensus problem for scalar parameters in the presence of attackers, and resilience is achieved by leveraging

high connectivity. Resilient consensus in the case of special network structures, such as triangular networks for distributed robotic applications [38], has also been studied. To achieve resilience in sparse networks, [36] presents the idea of employing few trusted nodes, which are hardened nodes that cannot be attacked. Resilience for concensus+innovation problems have also been studied by [14, 177, 178] in a fully-distributed way via agents' local observations and high network connectivity. Resilience can also be achieved via fault detection and isolation (FDI). For instance, [19] studied the FDI problem for linear consensus networks via high connectivity networks and global knowledge of the network structure by each agent. [20] considered a similar FDI problem for second-order systems. Authors in [21] presented distributed detection method for consensus+innovation algorithms via local observations of agents only. For attacks, typical approaches usually consider Byzantine adversaries with fixed target different than the true value [17] or with updates without time-dependent intention [14, 21] and assume that the goal of the attacker is to disrupt the convergence (stability) of the distributed algorithm. In contrast, this work focuses on attacks that do not disrupt convergence but drive normal agents to converge to states selected by the attacker. Moreover, in our attack model, the attacker continuously changes its values over time as compared to the fixed value attacks considered previously.

Resilience of diffusion-based distributed algorithms has been studied in [11, 16, 60]. Similar to the resilient consensus problems, fixed-value attacks are usually considered, and the main approach has been to use adaptive combination rules to counteract malicious values. This is an effective measure and has been applied to multi-task networks and distributed clustering problems [60]. Several variants focusing on adaptive weights applied to multi-task networks can be found in [97, 103–105]. Note that the essence of adaptive weights is similar to distributed detection. In contrast, it turns the detection method from a binary classification problem to a regression problem. Detection approach has also been applied in [104] for clustering over diffusion networks. Although adaptive weights provide some degree of resilience to byzantine adversaries with fixed values, we have shown in this work that adaptive weights may introduce vulnerabilities that allow time-dependent deception attacks.

Finally, there has been considerable work on applications of diffusion algorithms that include spectrum sensing in cognitive networks [160], target localization [161], distributed clustering [60], biologically inspired designs [162]. Although our approach can be used for resilience of various applications, we have focused on multi-target localization [105].

5.3 Preliminaries

We use normal and boldface fonts to denote deterministic and random variables respectively. The superscript $(\cdot)^*$ denotes complex conjugation for scalars and complex-conjugate transposition for matrices.

Consider a network of N (static) agents[2], in which an undirected edge (or a link) between two agents indicates that they share information and are neighbors of each other. The neighborhood of an agent k, denoted by $\mathcal{N}_k$ is the set of neighbors of k, including the agent k itself. At each iteration i, agent k has access to a scalar measurement $\boldsymbol{d}_k(i)$ and a regression vector $\boldsymbol{u}_{k,i}$ of size M with zero-mean and uniform covariance matrix $R_{u,k} \triangleq \mathbb{E}\{\boldsymbol{u}_{k,i}^* \boldsymbol{u}_{k,i}\} > 0$, which are related via a linear model of the following form:

$$\boldsymbol{d}_k(i) = \boldsymbol{u}_{k,i} w_k^0 + \boldsymbol{v}_k(i).$$

where $\boldsymbol{v}_k(i)$ represents a zero-mean i.i.d. additive noise with variance $\sigma_{v,k}^2$ and w_k^0 denotes the unknown $M \times 1$ state vector of agent k.

The objective of each agent is to estimate w_k^0 from (streaming) data $\{\boldsymbol{d}_k(i), \boldsymbol{u}_{k,i}\}$ $(k = 1, 2, ..., N, i \geq 0)$. The objective state can be static or dynamic and we represent it as w_k^0 or $\boldsymbol{w}_{k,i}^0$ respectively. For simplicity, we use w_k^0 to denote the objective state in both the static and dynamic cases.

The state w_k^0 can be computed as the the unique minimizer of the following cost function:

$$J_k(w) \triangleq \mathbb{E}\{\|\boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i} w\|^2\}. \tag{5.1}$$

An elegant adaptive solution for determining w_k^0 is the least-mean-squares (LMS) filter [16], where each agent k computes successive estimators of w_k^0 without cooperation (noncooperative LMS) as follows:

$$\boldsymbol{w}_{k,i} = \boldsymbol{w}_{k,i-1} + \mu_k \boldsymbol{u}_{k,i}^* [\boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i} \boldsymbol{w}_{k,i-1}],$$

where $\mu_k > 0$ is the step size (can be identical or distinct across agents).

Compared to noncooperative LMS, diffusion strategies introduce an aggregation step that incorporates information gathered from the neighboring agents into the adaptation mechanism. One powerful diffusion scheme is adapt-then-combine (ATC) [16] which optimizes the solution in a distributed and adaptive way using the following update:

$$\boldsymbol{\psi}_{k,i} = \boldsymbol{w}_{k,i-1} + \mu_k \boldsymbol{u}_{k,i}^* [\boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i} \boldsymbol{w}_{k,i-1}] \quad \text{(adaptation)} \tag{5.2}$$

$$\boldsymbol{w}_{k,i} = \sum_{l \in \mathcal{N}_k} a_{lk}(i) \boldsymbol{\psi}_{l,i} \ , \qquad \text{(combination)} \tag{5.3}$$

where $a_{lk}(i)$ represents the weight assigned to agent l from agent k that is used to scale the data it receives

[2]We use the terms agent and node interchangeably.

from l, and the weights satisfy the following constraints:

$$a_{lk}(i) \geq 0, \qquad \sum_{l \in \mathcal{N}_k} a_{lk}(i) = 1, \qquad a_{lk}(i) = 0 \text{ if } l \notin \mathcal{N}_k. \tag{5.4}$$

Here the intermediate state $\boldsymbol{\psi}_{k,i}$ (obtained by the adaptation step) is shared among neighboring agents and a combination of neighbors' intermediate states contribute to the current estimate $\boldsymbol{w}_{k,i}$ of agent k.

In the case where agents estimate a common state w^0 (i.e., w_k^0 is same for every k), several fixed combination rules can be adopted such as Laplacian, Metropolis, averaging, and maximum-degree [179]. In the case of multiple tasks, agents are pursuing distinct but correlated objectives w_k^0. In this case, the combination rules mentioned above are not applicable because they simply combine the estimation of all neighbors without distinguishing if the neighbors are pursuing the same objective. An agent estimating a different state will prevent its neighbors from estimating the state of interest.

Diffusion LMS (DLMS) has been extended for multi-task networks in [60] using the following adaptive weights:

$$a_{lk}(i) = \begin{cases} \frac{\gamma_{lk}^{-2}(i)}{\sum_{m \in \mathcal{N}_k} \gamma_{mk}^{-2}(i)}, & l \in \mathcal{N}_k \\ 0, & \text{otherwise.} \end{cases} \tag{5.5}$$

where $\gamma_{lk}^2(i) = (1-\nu_k)\gamma_{lk}^2(i-1) + \nu_k \|\boldsymbol{\psi}_{l,i} - \boldsymbol{w}_{k,i-1}\|^2$ and ν_k is a positive step size known as the forgetting factor. This update enables agents to continuously learn about the neighbors agents should cooperate with. During the estimation task, agents pursuing different objectives will continuously assign smaller weights to each other according to (5.5). Once the weights become negligible, communication links between agents do not contribute to the estimation task. Consequently, as the estimation proceeds, only the agents estimating the same state cooperate.

DLMS with adaptive weights (DLMSAW) outperforms the noncooperative LMS as measured by the steady-state mean-square-deviation performance (MSD) [16]. For sufficiently small step-sizes, the network performance of noncooperative LMS is defined as the average steady-state MSD level among agents:

$$\text{MSD}_{\text{ncop}} \triangleq \lim_{i \to \infty} \frac{1}{N} \sum_{k=1}^{N} \mathbb{E}\|\tilde{\boldsymbol{w}}_{k,i}\|^2 \approx \frac{\mu M}{2} \cdot (\frac{1}{N} \sum_{k=1}^{N} \sigma_{v,k}^2),$$

where $\tilde{\boldsymbol{w}}_{k,i} \triangleq w_k^0 - \boldsymbol{w}_{k,i}$ and M is the size of regression vector $\boldsymbol{u}_{k,i}$. The network MSD performance of the diffusion network (as well as the MSD performance of a normal agent in the diffusion network) can be approximated by

$$\text{MSD}_{\text{k}} \approx \text{MSD}_{\text{diff}} \approx \frac{\mu M}{2} \cdot \frac{1}{N} \cdot (\frac{1}{N} \sum_{k=1}^{N} \sigma_{v,k}^2). \tag{5.6}$$

In [16], it is shown that $\text{MSD}_{\text{diff}} = \frac{1}{N}\text{MSD}_{\text{ncop}}$, which demonstrates an N-fold improvement of MSD performance.

5.4 Problem Formulation

Diffusion strategies have been shown to be robust to node and link failures as well as to nodes or links with high noise levels [11, 164]. In this chapter, we are interested in understanding if the adaptive weights introduce vulnerabilities in the case a subset of nodes within the network is compromised by a cyber attack. In this direction, first we analyze if it is possible for an attacker who has compromised a node k to make nodes in $\mathcal{N}_k$ converge to a state selected by the attacker. Second, we consider a network attack model in which we determine a minimum set of nodes to compromise to make the entire network converge to states selected by the attacker. Finally, we formulate the resilient distributed diffusion problem that guarantees that normal agents are not driven to the attackers' desired states, and continue the normal operation with the cooperation among neighbors possibly with a degraded performance.

5.4.1 Single Node Attack Model

We consider false data injection attacks deployed by a *strong* attacker that has complete knowledge of the system. In particular, we assume the following for the strong attack.

Assumption 5.1. *A strong attacker knows the topology of the network, the streaming data of targeted agents and the diffusion algorithm parameters they use, such as* μ_k.

To examine the resilience of distributed networks, it is common to assume a strong attack with full knowledge of the system, for instance, Byzantine attackers having a complete knowledge of the system are considered in [17, 18, 166–168]. However, we also consider a weak attack model in Section 5.9 in which an attacker has no knowledge of agents' parameters and has no access to their streaming data. Compromised nodes are assumed to be Byzantine in the sense that they can send arbitrary messages to their neighbors, and can also send different messages to different neighbors.

The objective of the attacker is to drive the normal nodes to converge to a specific state. We assume a compromised node a wants agent k to converge to state

$$w_{k,i}^a = \begin{cases} w_k^a, & \text{for stationary estimation} \\ w_k^a + \theta_{k,i}^a, & \text{for non-stationary estimation.} \end{cases}$$

This is equivalent to minimizing the objective function of the following form:

$$\min_{\boldsymbol{w}_{k,i}} \lim_{i\to\infty} \mathcal{G}(\boldsymbol{w}_{k,i}), \qquad \boldsymbol{w}^a_{k,i} \in \mathcal{D}_{w,k}, \tag{5.7}$$

where

$$\mathcal{G}(\boldsymbol{w}_{k,i}) = \|\boldsymbol{w}_{k,i} - w^a_{k,i}\|^2,$$

and $\mathcal{D}_{w,k}$ is the domain of state $\boldsymbol{w}_{k,i}$.

Another objective of the attacker can be to delay the convergence time of the normal agents. We observe that if the compromised node can make its neighbors to converge to a selected state, it can keep changing this state before normal neighbors converge. By doing so, normal neighbors of the attacked node will never converge to a fixed state. Thus, the attacker can achieve its goal to prolong the convergence time of normal neighbors. For that reason, we focus on the attack model based on objective (5.7).

5.4.2 Network Attack Model

If the attacker has a specific target node that she wants to attack and make it converge to a specific state, the attacker can compromise any neighbor of this node to achieve the objective. In the case the attacker wants to compromise the entire network and drive the multi-task estimation to specific states, she needs to determine a minimum set of nodes to compromise such that every normal node in the network can be driven to an incorrect estimate. Computing such a minimum set directly depends on the underlying structure, and can be formulated as *minimum dominating set problem* in graphs as discussed in Section 5.6.

5.4.3 Resilient Distributed Diffusion

Distributed diffusion is said to be *resilient* if

$$\lim_{i\to\infty} \boldsymbol{w}_{k,i} = w^0_k. \tag{5.8}$$

for all normal agents k in the network which ensures that all the noncompromised nodes converge to the true state.

We note that if agents do not cooperate or interact with each other at all, such as in the non-cooperative diffusion, then adversary cannot impact agents' estimates. So, non-cooperative diffusion is resilient in this sense. At the same time, agents are also unable to utilize the information from other agents aiming to achieve the similar objective. Consequently, the steady-state MSD as result of non-cooperative diffusion can be quite large. Here, our objective is to design a resilient diffusion algorithm that guarantees convergence to

the true estimates in the presence of adversary and also results in smaller MSD (as compared to the non-cooperative diffusion) by leveraging cooperation and information exchange between agents. We assume that in the neighborhood of a normal node, there could be at most F compromised nodes [17]. Assuming bounds on the number of adversaries is typical for the resiliency analysis of distributed algorithms, and our resilient algorithm is also based on this assumption.

5.5 Single Node Attack Design

We design a strong attack in which the attacker drives the targeted node k to converge to a wrong estimate $w_{k,i}^a$ by making k follow a desired trajectory defined using stochastic gradient descent. The attacker's goal is to ensure that k, which implements adaptive-then-combine LMS, actually updates its estimates according to the stochastic gradient descent defined by the attacker. Thus, the main task is to determine conditions under which adaptive-then-combine LMS of k guarantees the convergence of k's estimate to $w_{k,i}^a$.

We summarize the conditions below and then analyze them in detail in this section. Firstly, an attacker needs to know the estimate of node k in the previous iteration. *Lemma 5.1* shows that an attacker can obtain the estimate given node k's streaming data and parameters. Secondly, Node k should not assign any weight to the messages from its non-attacked neighbors. *Lemma 5.2* ensures this objective. Finally, the magnitude of the stochastic gradient descent update should be sufficiently small. Details are given in *Proposition 5.1*.

5.5.1 Gradient-based Attack Design

Here, we present an attack based on gradient-descent updates, and in the next subsection, provide conditions under which the attack is successful. For stationary estimation, the following gradient-descent update with a sufficient small step size μ_k^a at the i^{th} iteration is sufficient to achieve the objective in (5.7):

$$\begin{aligned} \boldsymbol{w}_{k,i} &= \boldsymbol{w}_{k,i-1} - \mu_k^a \nabla_{\boldsymbol{w}} \mathcal{G}(\boldsymbol{w}_{k,i-1}) \\ &= \boldsymbol{w}_{k,i-1} - r_k^a (\boldsymbol{w}_{k,i-1} - w_{k,i}^a), \end{aligned} \tag{5.9}$$

where $r_k^a = 2\mu_k^a$ is a non-negative step size (that can also be time-varying). For non-stationary estimation, the form is slightly different and it is described by[3]

$$\boldsymbol{w}_{k,i} = \boldsymbol{w}_{k,i-1} - r_k^a (\boldsymbol{w}_{k,i-1} - x_i), \tag{5.10}$$

[3]See proof of Proposition 5.1 in the Appendix.

where

$$x_i = \begin{cases} w_k^a, & \text{for stationary estimation} \\ w_k^a + \theta_{k,i-1}^a + \frac{\Delta\theta_{k,i-1}^a}{r_{k,i}^a}, & \text{for non-stationary estimation} \end{cases}$$

with $\Delta\theta_{k,i}^a = \theta_{k,i+1}^a - \theta_{k,i}^a$. And the diffusion estimate of k is

$$\boldsymbol{w}_{k,i} = \sum_{l \in \mathcal{N}_k} a_{lk}(i)\boldsymbol{\psi}_{l,i} = \sum_{l \in \mathcal{N}_k \setminus a} a_{lk}(i)\boldsymbol{\psi}_{l,i} + a_{ak}(i)\boldsymbol{\psi}_{a,i}.$$

It is sufficient to achieve the attack objective (5.7) if the attacker could make the estimate of k follow the gradient-descent trajectory, i.e.,

$$\sum_{l \in \mathcal{N}_k \setminus a} a_{lk}(i)\boldsymbol{\psi}_{l,i} + a_{ak}(i)\boldsymbol{\psi}_{a,i} = \boldsymbol{w}_{k,i-1} - r_k^a(\boldsymbol{w}_{k,i-1} - w_{k,i}^a). \tag{5.11}$$

Since $\boldsymbol{\psi}_{l,i} = \boldsymbol{w}_{l,i-1} + \mu_l \boldsymbol{u}_{l,i}^*[\boldsymbol{d}_l(i) - \boldsymbol{u}_{l,i}\boldsymbol{w}_{l,i-1}]$ is a random variable that is not controlled by the attacker, the attacker should eliminate the influence of $\boldsymbol{\psi}_{l,i}$ for $l \in \mathcal{N}_k, l \neq a$. Sufficient conditions to hold (5.11), and thus to achieve the attack objective are as follows:

$$\boldsymbol{\psi}_{a,i} = \boldsymbol{w}_{k,i-1} - r_k^a(\boldsymbol{w}_{k,i-1} - x_i), \text{ and} \tag{5.12}$$

$$a_{lk}(i) \to 0, \forall l \in \mathcal{N}_k, l \neq a, \text{ and } a_{ak}(i) \to 1. \tag{5.13}$$

That is, the attacker uses the exchanging message $\boldsymbol{\psi}_{k,i}$ as indicated in (5.12) and the targeted node k updates its estimate based only on $\boldsymbol{\psi}_{k,i}$. $\boldsymbol{\psi}_{k,i}$ is computed given the knowledge of $\boldsymbol{w}_{k,i-1}$, that can be obtained by the attacker given *Lemma 5.1*.

Lemma 5.1. [4] *If a compromised node a has a knowledge of node k's streaming data $\{\boldsymbol{d}_k(i), \boldsymbol{u}_{k,i}\}$ and the parameter μ_k, then it can compute $\boldsymbol{w}_{k,i-1}$.*

Next, we see that by carefully designing $\boldsymbol{\psi}_{a,i}$ as explained in *Lemma 5.2*, conditions in (5.13) are satisfied.

Lemma 5.2. *If the attacker sends the message $\boldsymbol{\psi}_{a,i}$ satisfying $\|\boldsymbol{\psi}_{a,i} - \boldsymbol{w}_{k,i-1}\| \ll \|\boldsymbol{\psi}_{l,i} - \boldsymbol{w}_{k,i-1}\|$, $\forall l \in \mathcal{N}_k, l \neq a, \forall i$, then* (5.13) *will be true.*

5.5.2 Sufficient Conditions and Convergence Analysis

Here, using results from the previous subsection, we present conditions that guarantee a successful attack. A direct consequence of *Lemma 5.2* is that we could replace the condition in (5.13) by $\|\boldsymbol{\psi}_{a,i} - \boldsymbol{w}_{k,i-1}\| \ll$

[4] The proofs can be found in the Appendix.

$\|\boldsymbol{\psi}_{l,i} - \boldsymbol{w}_{k,i-1}\|, \forall l \in \mathcal{N}_k, l \neq a, \forall i$. At the same time, from (5.12), we get

$$\|\boldsymbol{\psi}_{a,i} - \boldsymbol{w}_{k,i-1}\| = \|r_k^a(\boldsymbol{w}_{k,i-1} - x_i)\|.$$

Therefore, a sufficient condition to achieve the attack objective can be rewritten as

$$\begin{aligned} \boldsymbol{\psi}_{a,i} &= \boldsymbol{w}_{k,i-1} - r_k^a(\boldsymbol{w}_{k,i-1} - x_i), \\ s.t.\ \|r_k^a(\boldsymbol{w}_{k,i-1} - x_i)\| &\ll \|\boldsymbol{\psi}_{l,i} - \boldsymbol{w}_{k,i-1}\|. \end{aligned} \tag{5.14}$$

Thus, the attacker has to select a sufficiently small value of r_k^a to make (5.14) true. Note that even though $r_k^a = 0$ is sufficient for (5.14), it renders the gradient of (5.9) zero and as a result no progress is made towards convergence to $w_{k,i}^a$. Also note that to use (5.14), it is assumed that the communication message $\boldsymbol{\psi}_{l,i}$ from every $l \in \mathcal{N}_k$ is known by the attacker, which can be achieved by intercepting the message. In practice, a sufficiently small value of r_k^a guarantees that the condition holds. The attacker can select a small r_k^a and observe if the attack succeeds; if not, decrease r_k^a to find an appropriate value. It is also worth noting that for a fixed value of r_k^a, (5.14) may not hold for some iteration i because of the randomness of variables. Yet we can always set $r_k^a = 0$ for such iterations i (no progress at the current point). However, in practice, the attack succeeds by using a small fixed value of $r_k^a > 0$ since estimation is robust to infrequent small values of $\|\boldsymbol{\psi}_{l,i} - \boldsymbol{w}_{k,i-1}\|$ caused by randomness given the smoothing property of the adaptive weight.

Next, we argue that (5.14) is sufficient to achieve the attack objective. We summarize the above discussion in *Proposition 5.1* and include a detailed proof in Appendix 5.A.

Proposition 5.1. *If $r_k^a > 0$ is selected such that $\forall l \in \mathcal{N}_k \cap l \neq a$, $\forall i \geq i_a$, $\|r_k^a(\boldsymbol{w}_{k,i-1} - x_i)\| \ll \|\boldsymbol{\psi}_{l,i} - \boldsymbol{w}_{k,i-1}\|$, then the compromised node a can realize the objective* (5.7) *by using $\boldsymbol{\psi}_{a,i}$ described in* (5.12) *as the communication message with k.*

Next, we discuss the convergence time of attack. Note that as $i \to \infty$,

$$\lim_{i \to \infty} (1 - r_k^a)^i = 0.^5$$

In practice, when the left side of the above equation is smaller than a certain small value ϵ, that is,

$$(1 - r_k^a)^{i_c^a(\epsilon)} \leq \epsilon,$$

we consider that the convergence to the desired state is achieved. Moreover, time required to reach the desired

[5] Refer to equation (26) in the Appendix.

state is denoted by $i_c^a(\epsilon)$, and is computed as

$$i_c^a(\epsilon) = \frac{\log \epsilon}{\log(1 - r_k^a)}. \tag{5.15}$$

It is also worth mentioning that it is not necessary to start the attack at the beginning of the diffusion task in order to guarantee the convergence of the attack. In other words, the attack can start at any time even after the diffusion algorithm has converged to its correct target as long as the condition in *Proposition 5.1* is satisfied.

5.6 Network Attack Design

In this section, we consider the case when multiple nodes are compromised using the attack model presented above. Our objective is to determine the minimum set of nodes to compromise in order to attack the entire network. For this, we show: (1) It is not necessary for the attacker to compromise multiple compromised nodes in order to attack a single node and (2) it is not possible for a compromised node to influence nodes, that is, make such nodes not converge to the desired states, that are not its immediate neighbors. Therefore, the minimum set to compromise is simply a *minimum dominating set* of the network, which we explain later in the section.

5.6.1 Impact of Compromised Nodes on Normal Nodes

In this subsection, first we discuss the impact of multiple compromised nodes attacking a single normal node, and then analyze the impact of a compromised node can make beyond its immediate neighbors.

Lemma 5.3. *If the compromised nodes send identical message as proposed in* (5.12)*, then multiple compromised nodes attacking one normal node is equivalent to one compromised node attacking the normal node.*

The next problem to consider is if a compromised node could indirectly impact its neighbors' neighbors that at the same time are not the neighbors of the attacker a. To illustrate this, we consider an attacker node a, a normal node l, and a large clique[6] of normal nodes $\mathcal{C}$ such that each node in a clique is connected to both a and l, and there is no edge between nodes a and l.

Using the proposed attack model, a is able to drive every node in the clique to converge to its selected state. We are interested in finding if the normal node l, that is connected to the clique, is also affected by the

[6]Every node is connected to every other node in a clique.

attack. The state of l is obtained by

$$\begin{aligned}\boldsymbol{w}_{l,i} &= \sum_{k\in\mathcal{C}} a_{kl}(i)\boldsymbol{\psi}_{k,i} + a_{ll}(i)\boldsymbol{\psi}_{l,i} \\ &= \sum_{k\in\mathcal{C}} a_{kl}(i)(\boldsymbol{w}_{k,i-1} + \mu_k \boldsymbol{u}_{k,i}^*[\boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i}\boldsymbol{w}_{k,i-1}]) \\ &\quad + a_{ll}(i)(\boldsymbol{w}_{l,i-1} + \mu_l \boldsymbol{u}_{l,i}^*[\boldsymbol{d}_l(i) - \boldsymbol{u}_{l,i}\boldsymbol{w}_{l,i-1}]).\end{aligned} \tag{5.16}$$

We use $\boldsymbol{R}_{k,i}$ to denote the random variable $\mu_k \boldsymbol{u}_{k,i}^*[\boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i}\boldsymbol{w}_{k,i-1}]$ for k in the clique and $\boldsymbol{R}_{l,i}$ to denote $\mu_l \boldsymbol{u}_{l,i}^*[\boldsymbol{d}_l(i) - \boldsymbol{u}_{l,i}\boldsymbol{w}_{l,i-1}]$ for normal node l. Suppose the compromised node a could affect nodes beyond its neighborhood, from some point i, $\boldsymbol{w}_{k,i}$ converges to w_k^a and $\boldsymbol{w}_{l,i}$ converges to w_l^a (assume both $w_k^a \neq w_k^0$ and $w_l^a \neq w_l^0$).

Thus, (5.16) turns into:

$$\begin{aligned}w_l^a &= \sum_{k\in\mathcal{C}} a_{kl}(i)(w_k^a + \boldsymbol{R}_{k,i}) + (1 - \sum_{k\in\mathcal{C}} a_{kl}(i))(w_l^a + \boldsymbol{R}_{l,i}) \\ &= \sum_{k\in\mathcal{C}} a_{kl}(i)(w_k^a - w_l^a + \boldsymbol{R}_{k,i} - \boldsymbol{R}_{l,i}) + w_l^a + \boldsymbol{R}_{l,i}.\end{aligned} \tag{5.17}$$

After inserting constants and random variables, (5.17) can be written as

$$\sum_{k\in\mathcal{C}} a_{kl}(i)(w_l^a - w_k^a) = \sum_{k\in\mathcal{C}} a_{kl}(i)\boldsymbol{R}_{k,i} + (1 - \sum_{k\in\mathcal{C}} a_{kl}(i))\boldsymbol{R}_{l,i}. \tag{5.18}$$

Here, $(w_k^a - w_l^a)$ is a constant and $a_{lk}(i)$ changes slowly and can be considered as a constant that does not change within a small period of time. Then, (5.18) implies a constant equals to a random variable, which does not hold except that both sides equal to zero. For the left side, that is when $\sum_{k\in\mathcal{C}} a_{kl}(i) \to 0$ or $(w_l^a - w_k^a) \to 0$. Consider, when $(w_l^a - w_k^a) \to 0$, that is, $w_l^a \to w_k^a$. In such cases,

$$\begin{aligned}\boldsymbol{R}_{l,i} &= \mu_l \boldsymbol{u}_{l,i}^*[\boldsymbol{d}_l(i) - \boldsymbol{u}_{l,i}\boldsymbol{w}_{l,i-1}] \\ &= \mu_l \boldsymbol{u}_{l,i}^*[\boldsymbol{u}_{l,i}w_l^0 + \boldsymbol{v}_l(i) - \boldsymbol{u}_{l,i}w_l^a] \\ &= \mu_l \boldsymbol{u}_{l,i}^*[\boldsymbol{u}_{l,i}(w_l^0 - w_l^a) + \boldsymbol{v}_l(i)] \neq \boldsymbol{0}.\end{aligned}$$

So is $\boldsymbol{R}_{k,i}$. Therefore, equation (5.18) does not hold under the condition $(w_l^a - w_k^a) \to 0$.

The other possible solution for equation (5.18) is when $\sum_{k\in\mathcal{C}} a_{kl}(i) \to 0$. This means l does not assign any weight to $k \in \mathcal{C}$ and operates by itself. In such cases, equation (5.18) holds when the right side of the equation is zero. Since $\sum_{k\in\mathcal{C}} a_{kl}(i) \to 0$, the right side turns into $\boldsymbol{R}_{l,i}$. We know when l converges to its

true objective state w_l^0, $\boldsymbol{R}_{l,i}$ is zero, i.e.,

$$\begin{aligned}\boldsymbol{R}_{l,i} &= \mu_l \boldsymbol{u}_{l,i}^*[\boldsymbol{d}_l(i) - \boldsymbol{u}_{l,i}\boldsymbol{w}_{l,i-1}] \\ &= \mu_l \boldsymbol{u}_{l,i}^*[\boldsymbol{u}_{l,i}w_l^0 + \boldsymbol{v}_l(i) - \boldsymbol{u}_{l,i}w_l^0] \\ &= \mu_l \boldsymbol{u}_{l,i}^*\boldsymbol{v}_l(i) \to \mathbf{0}.\end{aligned}$$

Thus, equation (5.18) holds under two conditions: First, $\sum_{k\in\mathcal{C}} a_{kl}(i) \to 0$, that is, l does not give any weight to $k \in \mathcal{C}$. Second, $\boldsymbol{R}_{l,i} \to 0$, that is, l converges to its true objective state w_l^0.

We note that the above two conditions indicate that l converges to its original goal state and will not assign any weight to its compromised neighbors under the above conditions. Based on this discussion, we have *Lemma 5.4*.

Lemma 5.4. *The attacker cannot change the convergence state of the nodes that are not its immediate neighbors.*

Next, we see how many compromised nodes are needed to attack the entire network.

5.6.2 Minimum Set of Compromised Nodes to Attack the Entire Network

Since it is not necessary to use more than one compromised nodes to attack one single normal agent, and a compromised node cannot affect nodes beyond its neighborhood, finding a minimum set of nodes to compromise in order to attack the entire network is equivalent to finding a minimum dominating set of the network as defined below [180].

Definition 5.1. (Dominating set) *A dominating set of a graph $G = (V, E)$ is a subset D of V such that every vertex not in D is adjacent to at least one member of D.*

Definition 5.2. (Minimum dominating set) *A minimum dominating set of a graph is a dominating set of the smallest size.*

It should be noted that finding a minimum dominating set of a network is an NP-complete problem but approximate solutions using greedy approaches work well in practice (for instance, see [180]). With the above discussion, we state the following:

Proposition 5.2. *The compromised nodes need to form a dominating set if the attacker wants every node in the network to converge to its desired state.*

Based on the above discussion, we observe that the above condition is both necessary and sufficient.

5.7 Resilient Distributed Diffusion

In this section, we propose a resilient diffusion algorithm that guarantees convergence of normal nodes to their actual states if the number of compromised nodes in the neighborhood of a normal node is bounded. The proposed algorithm takes a non-negative integer F as an input parameter. If the number of compromised nodes in the neighborhood of a normal node is at most F, then the algorithm is resilient to any such attack. It is obvious that selecting a large F value achieves a higher level of resilience, while selecting $F = 0$ means that the algorithm is not resilient to any attack. However, there exists a trade-off between the resilience and the steady-state MSD performance of the algorithm, which we will analyze in detail. Since the proposed algorithm is adapted from the known DLMSAW, we call it a *Resilient Diffusion Least Mean Square with Adaptive Weights (R-DLMSAW)*. We also note that in contrast to the connectivity requirements needed by resilient concensus problems [17], since in resilient diffusion, connectivity does not affect convergence, but only the estimation performance measured by the steady-state MSD.

Since our algorithm can achieve resilience to up to F compromised nodes, we assume that there can be at most F compromised nodes in the neighborhood of any node, which is also referred to as the F-local model in [17]. Specifically, we define:

Definition 5.3. (F-local model) *A node satisfies the F-local model if there is at most F compromised nodes in its neighborhood.*

Definition 5.4. (F-local network) *A network is considered to satisfy the F-local model if every node in the network has at most F compromised nodes in its neighborhood.*

While the chapter focuses on the F-local model, scenarios involving bounds on the total number of compromised nodes within the network (F-total model [17]) can also be analyzed using a similar approach. Next, we describe our resilient diffusion algorithm.

5.7.1 Resilient Diffusion Algorithm (R-DLMSAW)

In the context of distributed consensus, it is shown in [17] that for Mean-Subsequence-Reduced (MSR) algorithms, that during the state update phase, a node discards the values of neighbors that are too far off from the node's own value, resilience against attacks can be achieved, that is, distributed consensus in the presence of compromised nodes (F-local and F-total models) is guaranteed. In distributed diffusion, we recall that a node updates its estimate by taking a weighted average of the estimates of all of its neighbors (5.3). For resilient diffusion, we utilize a similar idea as in [17], that is instead of considering the estimates of all neighbors during the state update phase, only consider values from a subset of neighbors sharing close

estimates. We show that this strategy guarantees convergence of normal nodes to true estimates. Before outlining the resilient distributed diffusion algorithm, we first explain the notion of the cost of a node.

Following (5.3), normal agent k follows diffusion dynamics given by

$$\boldsymbol{w}_{k,i} = \sum_{l \in \mathcal{N}_k} a_{lk}(i) \boldsymbol{\psi}_{l,i}.$$

Thus, the cost function in (5.1) in the i^{th} iteration can be written as:

$$\begin{aligned} J_k(\boldsymbol{w}_{k,i}) &= J_k(\sum_{l \in \mathcal{N}_k} a_{lk}(i) \boldsymbol{\psi}_{l,i}) \\ &= \mathbb{E}\{\|\boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i}(\sum_{l \in \mathcal{N}_k} a_{lk}(i) \boldsymbol{\psi}_{l,i})\|^2\}. \end{aligned}$$

Since $\sum_{l \in \mathcal{N}_k} a_{lk}(i) = 1$, we have

$$\boldsymbol{d}_k(i) = \sum_{l \in \mathcal{N}_k} a_{lk}(i) \boldsymbol{d}_k(i).$$

Thus,

$$\begin{aligned} J_k(\boldsymbol{w}_{k,i}) &= \mathbb{E}\left\{ \left\| \sum_{l \in \mathcal{N}_k} a_{lk}(i) \boldsymbol{d}_k(i) - \sum_{l \in \mathcal{N}_k} a_{lk}(i) \boldsymbol{u}_{k,i} \boldsymbol{\psi}_{l,i} \right\|^2 \right\} \\ &= \mathbb{E}\left\{ \left\| \sum_{l \in \mathcal{N}_k} a_{lk}(i) (\boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i} \boldsymbol{\psi}_{l,i}) \right\|^2 \right\} \\ &\approx \sum_{l \in \mathcal{N}_k} a_{lk}^2(i) \mathbb{E}\{\|\boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i} \boldsymbol{\psi}_{l,i}\|^2\} \\ &= \sum_{l \in \mathcal{N}_k} a_{lk}^2(i) J_k(\boldsymbol{\psi}_{l,i}) \\ &= \frac{\sum_{l \in \mathcal{N}_k} \gamma_{lk}^{-4}(i) J_k(\boldsymbol{\psi}_{l,i})}{[\sum_{m \in \mathcal{N}_k} \gamma_{mk}^{-2}(i)]^2}. \end{aligned} \tag{5.19}$$

The goal of k is to minimize its cost at every iteration, i.e., to minimize $J_k(\boldsymbol{w}_{k,i})$ by discarding F neighbors' message. Therefore, the removal set $\mathcal{R}_k(i)$ of size F should be selected by

$$\begin{aligned} \mathcal{R}_k(i) &= \arg\min J_k(\boldsymbol{w}_{k,i}) \\ &= \arg\min \frac{\sum_{l \in \mathcal{N}_k \setminus \mathcal{R}_k(i)} \gamma_{lk}^{-4}(i) J_k(\boldsymbol{\psi}_{l,i})}{[\sum_{m \in \mathcal{N}_k \setminus \mathcal{R}_k(i)} \gamma_{mk}^{-2}(i)]^2}. \end{aligned}$$

We note that the algorithm presented here is a generalization of the algorithm in [1] which is resilient to a specific type of Byzantine attack and has a lower computational cost. In contrast, the algorithm proposed

in this work is resilient to any Byzantine attack, but has a higher computational cost. Thus, there is a trade off between the computation complexity of the algorithm and the scope of attacks to which the algorithm is resilient.

To compute the cost $J_k(\boldsymbol{\psi}_{l,i}) = \mathbb{E}\|\boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i}\boldsymbol{\psi}_{l,i}\|^2$, agent k has to store all the streaming data. Alternatively, we can approximate $J_k(\boldsymbol{\psi}_{l,i})$ using a moving average based on the previous iterations.

Next, we outline the basic idea of the proposed resilient distributed diffusion algorithm below, and present the details of *R-DLMSAW* in Algorithm 2.

1. If $F \geq |\mathcal{N}_k|$, agent k updates its current state $\boldsymbol{w}_{k,i}$ using only its own $\boldsymbol{\psi}_{k,i}$, which degenerates distributed diffusion to non-cooperative LMS.

2. If $F < |\mathcal{N}_k|$, at each iteration i, agent k computes $\binom{|\mathcal{N}_k|}{F}$ possible removal sets, and selects the one by removing which $J_k(\boldsymbol{\psi}_{l,i})$ is minimized. Then, the agent updates its current weight $a_{lk}(i)$ and state $\boldsymbol{w}_{k,i}$ without using information from nodes in $\mathcal{R}_k(i)$.

We note that for $F = 0$, DLMSAW and R-DLMSAW are essentially identical.

Algorithm 2: Resilient distributed diffusion under F-local bounds (R-DLMSAW)

Input: $\gamma_{lk}^2(-1) = 0$, maintain $n \times 1$ matrix $D_{k,i} = \mathbf{0}_{n\times 1}$ and $n \times M$ matrix $U_{k,i} = \mathbf{0}_{n\times M}$ for all $k = 1, 2, ..., N$, and $l \in \mathcal{N}_k$

for $k = 1, 2, ..., N, i \geq 0$ **do**
- $e_k(i) = \boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i}\boldsymbol{w}_{k,i-1}$
- $\boldsymbol{\psi}_{k,i} = \boldsymbol{w}_{k,i-1} + \mu_k \boldsymbol{u}_{k,i}^* e_k(i)$
- **if** $F \geq |\mathcal{N}_k|$ **then**
  - $\boldsymbol{w}_{k,i} = \boldsymbol{\psi}_{k,i}$
- **else**
  - $\gamma_{lk}^2(i) = (1 - \nu_k)\gamma_{lk}^2(i-1) + \nu_k\|\boldsymbol{\psi}_{l,i} - \boldsymbol{w}_{k,i-1}\|^2$
  - Update $D_{k,i}$ and $U_{k,i}$ by adding $\boldsymbol{d}_k(i)$ and $\boldsymbol{u}_{k,i}$ and removing $\boldsymbol{d}_k(i-n)$ and $\boldsymbol{u}_{k,i-n}$
  - $J_k(\boldsymbol{\psi}_{l,i}) = \mathbb{E}\|D_{k,i} - U_{k,i}\boldsymbol{\psi}_{l,i}\|^2$
  - Compute all possible discarded set $\mathcal{R}_k(i)^1$, $\mathcal{R}_k(i)^2$, ..., $\mathcal{R}_k(i)^{\binom{|\mathcal{N}_k|}{F}}$
  - $J_{\min} = \infty$
  - **for** $j = 1, 2, \ldots, \binom{|\mathcal{N}_k|}{F}$ **do**
    - $J = \frac{\sum_{l\in\mathcal{N}_k\setminus\mathcal{R}_k(i)^j} \gamma_{lk}^{-4}(i) J_k(\boldsymbol{\psi}_{l,i})}{[\sum_{m\in\mathcal{N}_k\setminus\mathcal{R}_k(i)^j} \gamma_{mk}^{-2}(i)]^2}$
    - **if** $J < J_{\min}$ **then**
      - $\mathcal{R}_k(i) = \mathcal{R}_k(i)^j$
      - $J_{\min} = J$
  - $a_{lk}(i) = \frac{\gamma_{lk}^{-2}(i)}{\sum_{m\in\mathcal{N}_k\setminus\mathcal{R}_k(i)} \gamma_{mk}^{-2}(i)}, l \in \mathcal{N}_k\setminus\mathcal{R}_k(i)$
  - $\boldsymbol{w}_{k,i} = \sum_{l\in N_k\setminus\mathcal{R}_k(i)} a_{lk}(i)\boldsymbol{\psi}_{l,i}$
- **return** $\boldsymbol{w}_{k,i}$

Proposition 5.3. *If the network is a F-local network, then R-DLMSAW is resilient to any message falsification attack.*

Proof. Given the F-local model, we assume that there are $n \leq F$ compromised nodes in the neighborhood of a normal node k. In the case of $F \geq |\mathcal{N}_k|$, k updates its state without using information from neighbors. Next, consider the case when $F < |\mathcal{N}_k|$. To deploy the attack, the attacker must try to make the message it sends to the normal nodes not being discarded by the normal nodes. This can only be achieved if the cost of keeping the attacker's message is smaller than keeping some normal agents' message (discarding the attacker's message). Therefore, any attack message not being discarded actually results in a cost smaller than the normal case. Therefore, R-DLMSAW is resilient to any message falsification attack. From the attacker's perspective, since its goal is to maximize cost $J_k(\boldsymbol{w}_{k,i})$, the optimal strategy for the attacker is not to make this cost even smaller. As a result, the information from the attacker will be discarded. Thus,

$$\boldsymbol{w}_{k,i} = \sum_{l \in \mathcal{N}_k \setminus \mathcal{R}_k(i)} a_{lk}(i) \boldsymbol{\psi}_{l,i},$$

meaning the algorithm performs the diffusion adaptation step as if there were no compromised node in its neighborhood. Note that messages from normal neighbors may also be discarded since F may be greater than the number of compromised neighbors. However, the distributed diffusion algorithm is robust to node and link failures [11], and it converges to the true state despite the links to some or all of its neighbors fail. Finally, the algorithm will converge and equation (5.8) holds, showing the resilience of R-DLMSAW. □

5.7.2 Trade-off Between Resilience and MSD Performance

An important aspect of R-DLMSAW is the selection of parameter F by each normal node. On the one hand, selection of a large F degrades the performance of the diffusion algorithm as measured by the steady-state MSD, but on the other hand, a smaller F might result in an algorithm that is not resilient against attacks. In the following, we summarize the trade-off between the steady-state MSD performance and resilience.

It is rather obvious that if a normal node selects F smaller than the number of compromised nodes in its neighborhood, then the messages from the compromised nodes might not be discarded entirely during the state update phase of R-DLMSAW. As a result, the algorithm might not be resilient against the attack, and the normal node might eventually converge to the attacker's desired state. However, if F is selected too large, then in the worst case, normal agents discard all the information from their neighbors. The algorithm becomes a non-cooperative diffusion algorithm and incurs an N-fold MSD performance deterioration. Thus, the performance of R-DLMSAW lies somewhere in-between the cooperative diffusion and non-cooperative diffusion depending on the choice of F selected.

Consider a connected network with N normal agents running R-DLMSAW. Let $\sigma_{v,k}^2 = \{\sigma_{v,1}^2, \ldots, \sigma_{v,N}^2\}$

be the noise variance. Suppose by selecting some F the network is resilient, but is no longer a connected graph and is decomposed into n connected sub-networks, each of which is denoted by S_j where $j \in \{1, \cdots, n\}$. Using (5.6), the steady-state MSD for each sub-network is

$$\text{MSD}_{S_j} \approx \frac{\mu M}{2} \cdot \frac{1}{(|S_j|)^2} \sum_{k \in S_j} \sigma_{v,k}^2,$$

where $|S_j|$ is the number of nodes in j^{th} sub-network. The steady-state MSD for the overall network (consisting of sub-networks) after running R-DLMSAW is the weighted average of the steady-state MSD of the sub-networks, that is

$$\text{MSD}_{\text{after}} = \frac{1}{N} \sum_{j=1}^{n} \text{MSD}_{S_j} \cdot |S_j| \approx \frac{\mu M}{2N} \cdot \sum_{j=1}^{n} \frac{1}{|S_j|} \sum_{k \in S_j} \sigma_{v,k}^2.$$

At the same time, the steady-state MSD for the (original) connected network before running R-DLMSAW is

$$\text{MSD}_{\text{before}} \approx \frac{\mu M}{2} \cdot \frac{1}{N^2} \sum_{k=1}^{N} \sigma_{v,k}^2 \approx \frac{\mu M}{2N} \cdot \sum_{j=1}^{n} \frac{1}{N} \sum_{k \in S_j} \sigma_{v,k}^2.$$

The difference between the two is

$$\text{MSD}_{\text{after}} - \text{MSD}_{\text{before}} = \frac{\mu M}{2N} \cdot \sum_{j=1}^{n} (\frac{1}{|S_j|} - \frac{1}{N}) \sum_{k \in S_j} \sigma_{v,k}^2.$$

We know that $|s_j| \leq N$. Therefore, $\frac{1}{|S_j|} - \frac{1}{N} \geq 0$, meaning the steady-state MSD of the network after running R-DLMSAW is worse than the steady-state MSD of the original network, and as the network is decomposed into more sub-networks, $\sum_{j=1}^{n} (\frac{1}{|S_j|} - \frac{1}{N})$ and $\text{MSD}_{\text{after}}$ becomes larger.

Therefore, it is crucial to select an appropriate F, that is a value with which the algorithm is resilient against compromised nodes and at the same time useful links between nodes are preserved. To this end, a simple way to select F is to first estimate $\boldsymbol{w}_{\text{ncop},k,i}$ by a non-cooperative diffusion and compute $J_k(\boldsymbol{w}_{\text{ncop},k,i})$. Then, starting with a small F, for instance $F = 0$, perform cooperative diffusion and compute $J_k(\boldsymbol{w}_{\text{coop},k,i})$. If $J_k(\boldsymbol{w}_{\text{coop},k,i}) > J_k(\boldsymbol{w}_{\text{ncop},k,i})$, it means that a compromised node is able to effect the estimation, and therefore increase F by 1. We keep repeating this as long as $J_k(\boldsymbol{w}_{\text{coop},k,i}) > J_k(\boldsymbol{w}_{\text{ncop},k,i})$ is true.

5.8 Evaluation

In this section, we evaluate three algorithms, *non-cooperative diffusion*, *DLMSAW*, and *R-DLMSAW*; and compare their performance for *no-attack* and *attack* scenarios. We evaluate the proposed attack model

and resilient algorithms using the application of multi-target localization [105, 179] for both stationary and non-stationary targets.

We consider a network of $N = 100$ agents, in which each agent's objective is to estimate the unknown location of its target of interest by the noisy observations of both the distance and the direction vector towards the target. These agents and targets are distributed in a plane. The location of agent k is denoted by the two-dimensional vector $p_k = [x_k, y_k]^\top$, and similarly the location of target is represented by the vector $w_k^0 = [x_k^0, y_k^0]^\top$. Figure 5.1 illustrates how an agent estimates the location of the target.

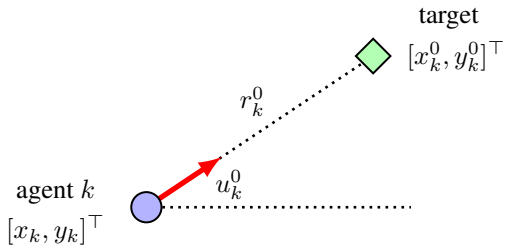


Figure 5.1: Illustration of target localization.

In Figure 5.1, the distance between agent k and the target is denoted by $r_k^0 = \|w_k^0 - p_k\|$, and the unit direction vector from agent k to the target is $u_k^0 = \frac{(w_k^0 - p_k)^\top}{\|w_k^0 - p_k\|}$. Therefore, the relationship holds such that $r_k^0 = u_k^0(w_k^0 - p_k)$. Since agents have only noisy observations $\{\boldsymbol{r}_k(i), \boldsymbol{u}_{k,i}\}$ of the distance and direction vector at every iteration i, we get the following:

$$\boldsymbol{r}_k(i) = \boldsymbol{u}_{k,i}(w_k^0 - p_k) + \boldsymbol{v}_k(i).$$

If we use the adjusted signal $\boldsymbol{d}_k(i)$, such that

$$\boldsymbol{d}_k(i) = \boldsymbol{r}_k(i) + \boldsymbol{u}_{k,i}p_k,$$

then we derive the following linear model for variables $\{\boldsymbol{d}_k(i), \boldsymbol{u}_{k,i}\}$ in order to estimate the target w_k^0:

$$\boldsymbol{d}_k(i) = \boldsymbol{u}_{k,i}w_k^0 + \boldsymbol{v}_k(i).$$

As a result, agents can rely on DLMSAW algorithm for the multi-target localization problem. Figure 5.2(a) shows the network topology before the application of diffusion algorithms. For better readability, we only illustrate the network topology of agents without showing targets.

For stationary target localization, the location of the two stationary targets are given by

$$w_k^0 = \begin{cases} [0.1, 0.1]^\top, & \text{for } k \text{ depicted in blue} \\ [0.9, 0.9]^\top, & \text{for } k \text{ depicted in green} \end{cases}$$

Non-stationary targets are given by

$$\boldsymbol{w}_{k,i}^0 = \begin{cases} \begin{bmatrix} 0.1 + 0.1\cos(2\pi\omega i) \\ 0.1 + 0.1\sin(2\pi\omega i) \end{bmatrix}, \text{for } k \text{ depicted in blue} \\ \begin{bmatrix} 0.9 + 0.1\cos(2\pi\omega i) \\ 0.9 + 0.1\sin(2\pi\omega i) \end{bmatrix}, \text{for } k \text{ depicted in green} \end{cases}$$

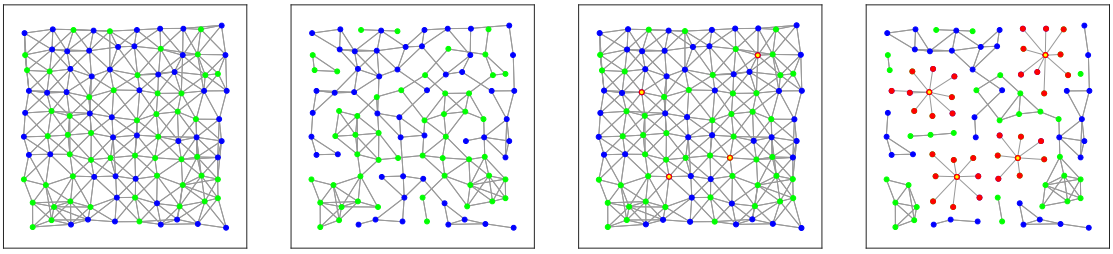
where $\omega = \frac{1}{2000}$.

Regression data is white Gaussian with diagonal covariance matrices $R_{u,k} = \sigma_{u,k}^2 I_M$ with $M = 2$, $\sigma_{u,k}^2 \in [0.8, 1.2]$ and noise variance $\sigma_k^2 \in [0.15, 0.2]$. The step size of $\mu_k = 0.01$ and the forgetting factor $\nu_k = 0.01$ are set uniformly across the network. Note that we adopt a signal-to-noise ratio (SNR) of $5 - 10$ dB in our setup. However, the same results are generated if we choose low SNR values.

5.8.1 Strong Attacks

We consider the strong attack model discussed in Sections 5.5 and 5.6. The attacker aims at making the normal agents estimate a specific location selected by the attacker. In this evaluation, we select the attacker's targeted location to be $w_k^a = [0.5, 0.5]^\top$, and the attack parameters are selected uniformly across the compromised agents as $r_k^a = 0.002$. For non-stationary estimation, we select $\theta_{k,i}^a = [0.1\cos(2\pi\omega_a i)$, $0.1\sin(2\pi\omega_a i)]^\top$, $\Delta\theta_{k,i}^a = [-0.2\pi\omega_a \sin(2\pi\omega_a i)$, $0.2\pi\omega_a \cos(2\pi\omega_a i)]^\top$, where $\omega_a = \frac{1}{2000}$. Figure 5.2(b) shows the network topology at the end of the simulation using DLMSAW with no attack for both stationary and non-stationary tasks. If the weights between agents k and l are such that $a_{lk}(i) < 0.01$ and $a_{kl}(i) < 0.01$, then we remove the link between such nodes from the network. We observe that only the links between agents estimating the same target are kept, that is green nodes are connected with green nodes only, and blue nodes are connected with only blue ones, thus, illustrating the robustness of DLMSAW in multi-task networks. Figure 5.3(a) and Figure 5.3(b) shows the estimation dynamics by DLMSAW for the targets' locations $\boldsymbol{w}_{k,i}(1)$ and $\boldsymbol{w}_{k,i}(2)$ for every agent k and iteration i from 0 to 5000 under no attack. Here $\boldsymbol{w}_{k,i}(1)$ and $\boldsymbol{w}_{k,i}(2)$ are the first and second element of the estimate respectively, that is $\boldsymbol{w}_{k,i} = [\boldsymbol{w}_{k,i}(1), \boldsymbol{w}_{k,i}(2)]^\top$. It is shown that the two groups of nodes converge to their goal state.

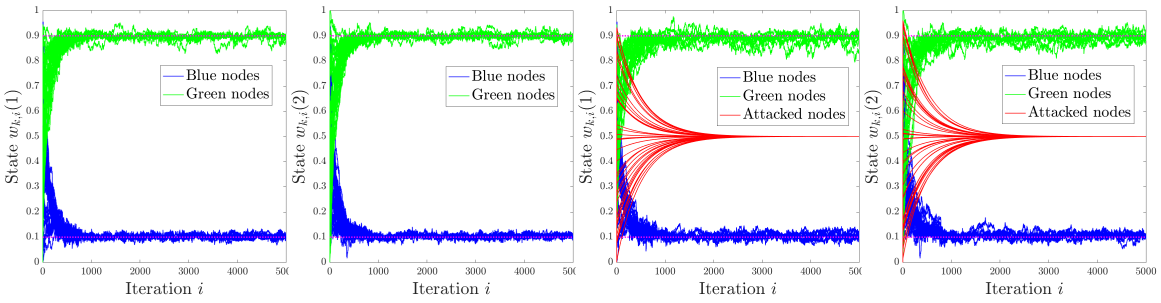
Figure 5.2(c) shows the initial network topology with compromised nodes. There are four compromised nodes (red nodes with yellow centres) in the network. Figure 5.2(d) shows the network topology at the end of DLMSAW in the case of a strong attack. All red nodes are the normal agents converging to w_k^a. We observe



(a) Initial network topology (no compromised nodes) (b) At the end of DLMSAW with no attack (c) Initial network topology (with compromised nodes) (d) At the end of DLMSAW under strong attack

Figure 5.2: Network topologies in the case of DLMSAW algorithm.

that neighbors of a compromised node communicate only with the compromised node, and not with any other node in the network. As a result, compromised nodes successfully drive all of their neighbors to desired states w_k^a as discussed in Section 5.6. Figure 5.3(c) and Figure 5.3(d) shows the estimation dynamics by DLMSAW for the targets' location $\boldsymbol{w}_{k,i}(1)$ and $\boldsymbol{w}_{k,i}(2)$ for every agent k and iteration from 0 to 5000 under attack. The attacked nodes in the figure refer to the immediate neighbors of the compromised nodes. It is shown that all the immediate neighbors of compromised nodes are driven to converge to w_k^a whereas all the other normal nodes converge to their original goal states.



(a) $\boldsymbol{w}_{k,i}(1)$ (under no attack) (b) $\boldsymbol{w}_{k,i}(2)$ (under no attack) (c) $\boldsymbol{w}_{k,i}(1)$ (under strong attack) (d) $\boldsymbol{w}_{k,i}(2)$ (under strong attack)

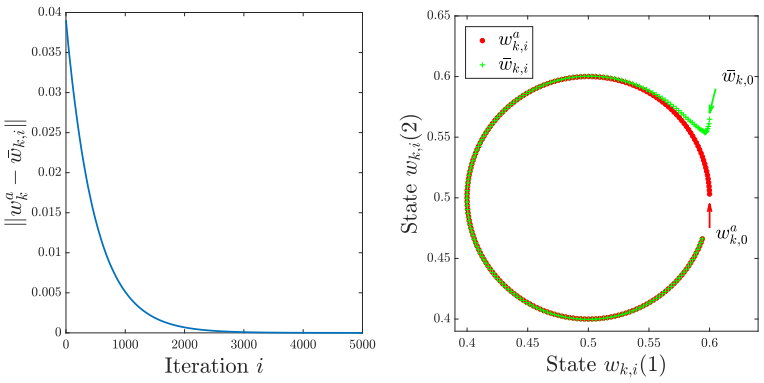
Figure 5.3: Estimation dynamics for stationary target localization by DLMSAW.

Figure 5.4(a) shows the convergence of nodes under attack (stationary targets). We note at around 3000 iterations, the difference between the average state of nodes under attack and the attacker's desired state w_k^a becomes almost zero. This observation is also consistent with the result in (5.15), as for $i = 3000$ and $r_k^a = 0.002$, the value of ϵ turns out to be 0.0025, which is indeed quite small and indicates the convergence of node's estimate to w_k^a.

Figure 5.4(b) shows the average state dynamics of nodes under attack for non-stationary targets. Since states are changing over time, we illustrate the dynamics of average states' changing with respect to the dynamics of attacker's selected state, instead of a convergence plot like 5.4(a). Here, the X-coordinate

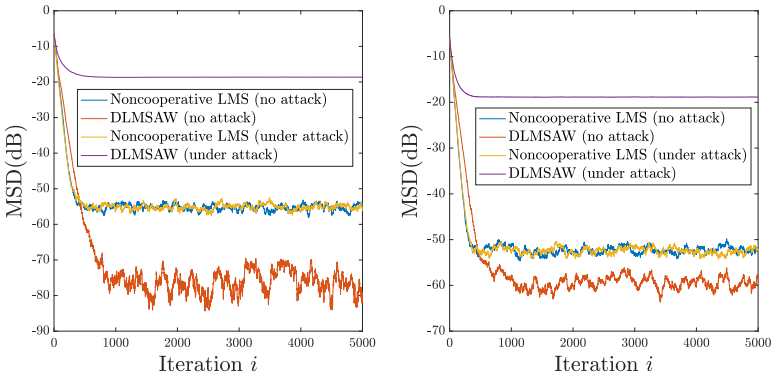
denotes the first element of the estimation vector, i.e., $\boldsymbol{w}_{k,i}(1)$, and Y-coordinate denotes the second, i.e., $\boldsymbol{w}_{k,i}(2)$. At iteration 0, the average state $\overline{w}_{k,i}$ of the nodes under attack is different than the attacker's desired state $w^a_{k,i}$. As the attack proceeds, $\overline{w}_{k,i}$ gradually converges towards $w^a_{k,i}$, which shows the effectiveness of attack for non-stationary state estimation.

Figure 5.5 shows the steady-state MSD performance of DLMSAW and non-cooperative LMS. We observe that under no attack, cooperation indeed improves the steady-state MSD performance of DLMSAW. However, in the case of an attack, the steady-state MSD level of DLMSAW is quite high, whereas, the steady-state MSD level of non-cooperative LMS is barely affected by the attack.



(a) Stationary targets (b) Non-stationary targets

Figure 5.4: Average state dynamics of compromised nodes' neighbors (under strong attack).



(a) Stationary targets (b) Non-stationary targets

Figure 5.5: Steady-state MSD levels in non-cooperative LMS and DLMSAW (under strong attack).

5.8.2 Resilient Diffusion for Strong Attacks

To evaluate R-DLMSAW, we compute the cost $J_k(\boldsymbol{\psi}_{l,i})$ using the streaming data from the latest 100 iterations. We adopt uniform F for every normal agent but it can be distinct for each agent. R-DLMSAW

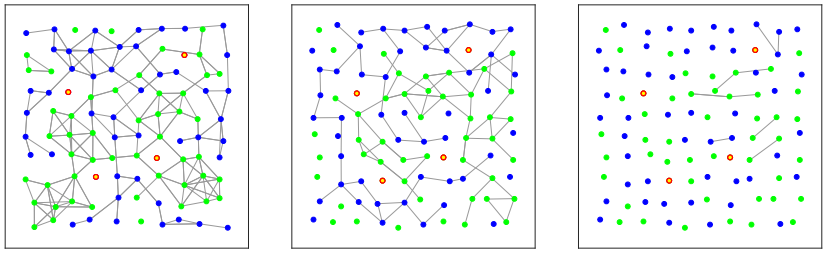
behaves identically to DLMSAW at one extreme, that is when $F = 0$, and on the other extreme it behaves like a non-cooperative LMS algorithm, that is for large F. We consider the same initial network as in Figure 5.2(a) and consider an attack consisting of four compromised nodes as previously. Note that there is at most one compromised node in the neighborhood of a normal agent. Figure 5.6 shows network topologies after executing R-DLMSAW for various values of F. Since there is at most one compromised node in the neighborhood of a normal agent, the selection of $F = 1$ should be sufficient to guarantee that none of the normal nodes converge to attacker's desired states, which is indeed the case as indicated by the removal of all links between normal and compromised nodes in Figure 5.6(a). As we increase F, resilience against attack is certainly achieved, but at the same time the network becomes sparser as illustrated in Figures 5.6(b) and 5.6(c). In the case of non-stationary state estimation, the resulting network topologies are similar, and hence, are not presented.

Figure 5.7 shows the estimation dynamics by R-DLMSAW for the targets' location $\boldsymbol{w}_{k,i}(1)$ and $\boldsymbol{w}_{k,i}(2)$ for every agent k and iteration i from 0 to 5000 under attack. The attacked nodes in the figure refer to the immediate neighbors of the compromised nodes. Since there is at most one compromised node in a normal node's neighborhood, setting $F \geq 1$ will make R-DLMSAW algorithm resilient to attacks, which is demonstrated by the results from the figure. We also observe that by setting a smaller F value, which is sufficient to to make the algorithm resilient, we achieve better estimation performance ($F = 1$ has less noise than that of $F = 5$).

Figure 5.8 shows the steady-state MSD level of the network for the three algorithms, that is, non-cooperative LMS, DLMSAW, and R-DLMSAW. The simulation results validate claims in Section 5.7. We observe that in the presence of compromised nodes, DLMSAW performs the worst and has the highest steady-state MSD. Since there is at most one compromised node in the neighborhood of any normal node, the most appropriate value of F for R-DLMSAW is 1. We note that the steady-state MSD is indeed minimum for $F = 1$. As we increase F, the steady-state MSD also increases. In fact, for $F = 5$, the performance of R-DLMSAW and non-cooperative LMS is almost the same as we expect.

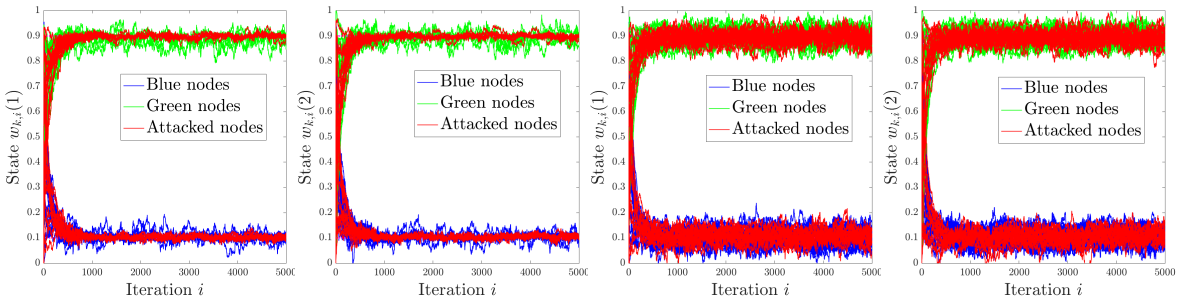
5.9 Weak Attacks

Though it is common to assume a strong attacker with complete knowledge when examining the resilience of a distributed system, it is interesting to examine what an attacker can do in practise if all the information is not available. In this section, we analyze how the attack can still be deployed on a normal agent k without the assumption of a strong knowledge by the attacker (streaming data and parameters used by k). We assume that an attacker has access only to the intermediate estimates shared by agents with others in their neighborhood.



(a) $F = 1$ (b) $F = 3$ (c) $F = 4$

Figure 5.6: Network topologies at the end of R-DLMSAW under strong attack (stationary targets) for various values of F.



(a) $\boldsymbol{w}_{k,i}(1)$ $(F = 1)$ (b) $\boldsymbol{w}_{k,i}(2)$ $(F = 1)$ (c) $\boldsymbol{w}_{k,i}(1)$ $(F = 5)$ (d) $\boldsymbol{w}_{k,i}(2)$ $(F = 5)$

Figure 5.7: Estimation dynamics for stationary target localization by R-DLMSAW under strong attack.

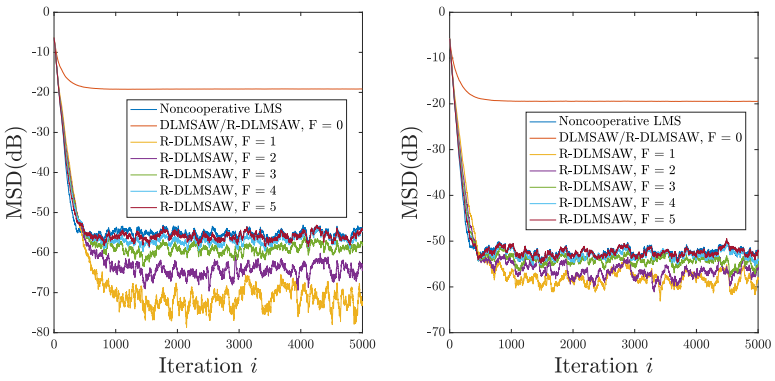
For instance, if $l \in \mathcal{N}_k$ then agent k receives $\boldsymbol{\psi}_{l,i}$ from l and attacker also has an access to it. We show that the other knowledge needed by the attacker can actually be approximated in an alternative way, and the success of the attack relies on how accurate this information can be approximated. We refer to such an attack in which attacker can only gather intermediate estimates and not the other data (including streaming data and agent parameters) as the *weak attack*.

The strong attack in (5.10) relies essentially on the knowledge of $\boldsymbol{w}_{k,i-1}$, that is the estimated state of agent k in the last iteration. If the attacker has complete knowledge, it can compute $\boldsymbol{w}_{k,i-1}$ exactly as Lemma 5.1 indicates. However, without such knowledge, $\boldsymbol{w}_{k,i-1}$ can only be approximately computed. We note that approximating $\boldsymbol{w}_{k,i-1}$ is equivalent to approximating the weight matrix $A_k(i) = [a_{lk}(i)], \forall l \in \mathcal{N}_k$. This is true because $\boldsymbol{w}_{k,i} = \sum_{l \in N_k} a_{lk}(i)\boldsymbol{\psi}_{l,i}$, and $\boldsymbol{\psi}_{l,i}$ is received by the attacker a from l.

Next, we discuss how to compute the approximated weight matrix $\hat{A}_k(i-1)$ using only the information $\boldsymbol{\psi}_{l,i}, \forall l \in \mathcal{N}_k$. Note that the adaptation step (5.2) of diffusion can be written as,

$$\boldsymbol{\psi}_{k,i} = \boldsymbol{w}_{k,i-1} + \nabla_{k,i} = A_k(i-1)\Psi_{k,i-1} + \nabla_{k,i}.$$

where $\nabla_{k,i} = \mu_k \boldsymbol{u}_{k,i}^*(\boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i}\boldsymbol{w}_{k,i-1})$, $\Psi_{k,i-1}$ is an $|\mathcal{N}_k| \times M$ matrix $\Psi_{k,i-1} = [\boldsymbol{\psi}_{l,i-1}], \forall l \in \mathcal{N}_k$.



(a) Stationary targets (b) Non-stationary targets

Figure 5.8: A comparison of MSD performance of non-cooperative LMS, DLMSAW, and R-DLMSAW under strong attack.

Thus, $\nabla_{k,i} = \boldsymbol{\psi}_{k,i} - A_k(i-1)\Psi_{k,i-1}$, and therefore,

$$\lim_{i\to\infty} \mathbb{E}\{\|\nabla_{k,i}\|^2\} = \lim_{i\to\infty} \mathbb{E}\{\|\boldsymbol{\psi}_{k,i} - A_k(i-1)\Psi_{k,i-1}\|^2\}.$$

Since $\lim_{i\to\infty} \mathbb{E}\{\|\nabla_{k,i}\|^2\} = 0$, the value of $A_k(i)$ can be approximated by assigning a cost function

$$\ell(A_k(i)) \triangleq \mathbb{E}\{\|\boldsymbol{\psi}_{k,i+1} - A_k(i)\Psi_{k,i}\|^2\},$$

where $A_k(i)$ is the global minimizer of $\ell(A_k(i))$ as $i \to \infty$. Next, we compute the successive estimators of the weight matrix based on stochastic gradient descent method as follows:

$$\begin{aligned} \hat{A}_k(i) &= \hat{A}_k(i-1) - \mu'_A \nabla_A \ell(\hat{A}_k(i-1)) \\ &= \hat{A}_k(i-1) + \mu_A \Psi_{k,i-1}(\boldsymbol{\psi}_{k,i} - \hat{A}_k(i-1)\Psi_{k,i-1}), \end{aligned} \tag{5.20}$$

where $\mu_A = \frac{1}{2}\mu'_A$.

Also recall weight matrix $A_k(i)$ has to satisfy the condition (5.4). Thus, to make the adaptive approximation of weight matrix hold condition (5.4), we introduce two more steps following (5.20), that is the clip step and the normalization step. In the clip step, the negative weights are clipped and are set to zero; and the in the normalization step, weights are divided by their sum. The operation for approximating weight matrix of a normal agent k is summarized in *Algorithm 3*.

We then approximate normal agent k's estimated state by

$$\hat{\boldsymbol{w}}_{k,i} = \hat{A}_k(i)\Psi_{k,i},$$

Algorithm 3: Approximate weight matrix for agent k

Input: $l \in N_k$, randomized $a_{lk}(0)$ satisfying (5.4), μ_A, $\boldsymbol{\psi}_{k,i}$, $\Psi_{k,i-1}$

for $i > 0$ **do**
  $A_k(i) = A_k(i-1) + \mu_A \Psi_{k,i-1}(\boldsymbol{\psi}_{k,i} - A_k(i-1)\Psi_{k,i-1})$
  for $l \in \mathcal{N}_k$ **do**
    $a_{lk}(i) = \max(a_{lk}(i), 0)$
  $A_k(i) = \frac{A_k(i)}{\sum a_{lk}(i)}$
  return $A_k(i)$

and use $\hat{\boldsymbol{w}}_{k,i}$ instead of $\boldsymbol{w}_{k,i}$. The attack model in (5.10) then becomes

$$\boldsymbol{\psi}_{a,i} = \hat{\boldsymbol{w}}_{k,i-1} + r_k^a(x_i - \hat{\boldsymbol{w}}_{k,i-1}). \tag{5.21}$$

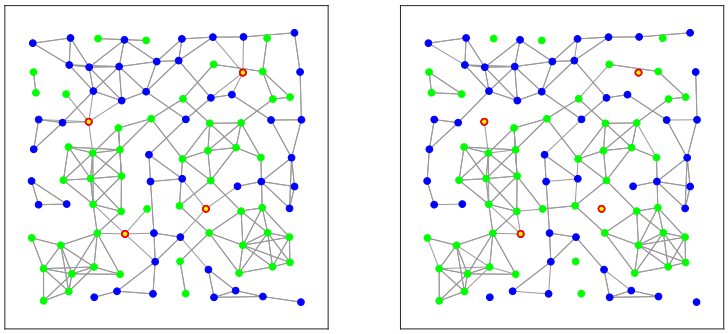
Note that the sufficient condition listed in *Proposition 5.1* guarantees the convergence of the attack objective. However, without an exact knowledge of $\boldsymbol{w}_{k,i-1}$ it is not guaranteed the sufficient condition can be satisfied. In other words, the success of the attack relies highly on how accurate the state $\hat{\boldsymbol{w}}_{k,i}$ can be approximated. In the following, we provide evaluation results for such an attack.

5.9.1 Evaluation

We adopt the same evaluation set-up as we used in Section 5.8. Initial network topology is the same as in 5.2(a). Parameters we select are: $\sigma_{u,k}^2 \in [0.75, 0.85]$, $\sigma_k^2 \in [0.75, 0.85]$ for each agent k and $\mu_A = 0.002$, while all the other settings are the same as in Section 5.8.

At the end of DLMSAW under weak attack, we reach the network topology as shown in Figure 5.9(a). From the plots, we find some of the agents maintain connection with the compromised nodes, while others do not, which is not the case with a strong attack, where all the neighboring agents of a compromised node end up cooperating only with the compromised node. The main reason for this is that the weak attack may not have an accurate approximation of normal agents' state. Without an accurate approximation, compromised nodes may not be able to collect large weights from their neighbors and may not keep influencing the states of their neighbors.

Figure 5.10 illustrates the estimation precision ($\|\hat{\boldsymbol{w}}_{k,i} - \boldsymbol{w}_{k,i}\|$) by the attacker. It shows that the attacker has different levels of accuracy to estimate the states of its neighboring agents. For some agents, the attacker has accurate approximation along the simulation iterations. As a result, the attacker is more likely to make its attack successful on those agents. However, for other agents, the attacker does not have very good approximation accuracy and therefore, it is hard for the attacker to successfully attack such agents. Figure 5.12 shows the state estimation dynamics of normal agents (wherein attacked nodes refer to the neighboring nodes of the



(a) DLMSAW (b) R-DLMSAW

Figure 5.9: Network topologies at the end of simulation under weak attack.

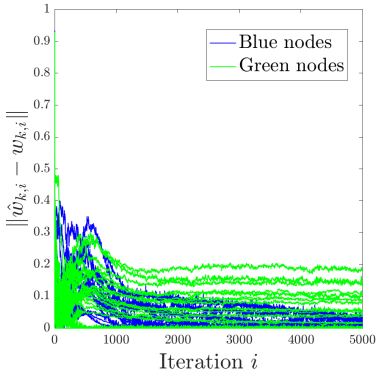


Figure 5.10: Sate estimation precision.

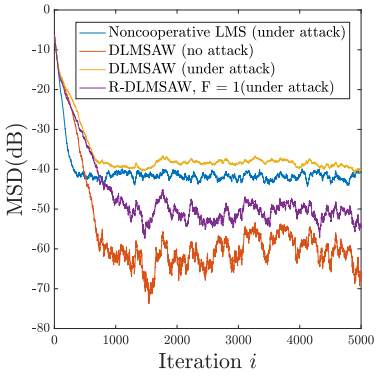
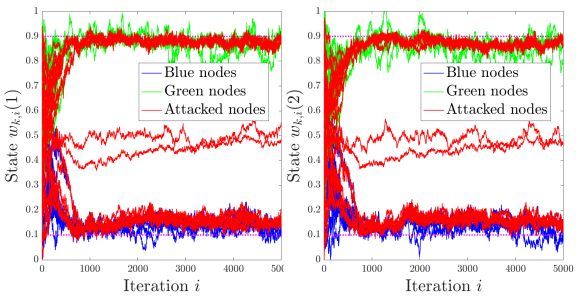


Figure 5.11: Steady-state MSD comparison under weak attack.

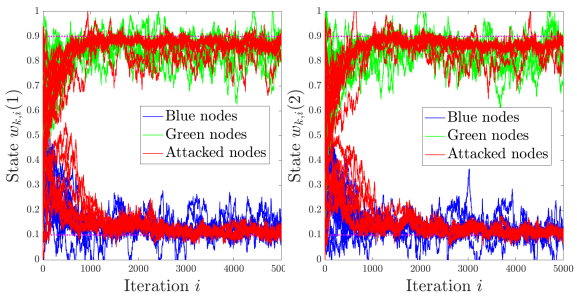
compromised nodes). We find the attacker can only drive a few of its neighbors to its desired state, whereas most of the normal neighbors converge to their true goal state, which is consistent with the results of Figure 5.10. The steady-state MSD performance for the weak attack is shown in the yellow line in Figure 5.11. We find that such an attack still worsens the network steady-state MSD as compared to the non-cooperative LMS (the blue line) and DLMSAW without attack (the red line).

Next, we evaluate the proposed resilient diffusion algorithm R-DLMSAW against the weak attack. The network topology at the end of simulation is shown in Figure 5.9(b). Most normal agents have cut the link with the compromised nodes. Yet some links are maintained because these compromised nodes behave in a benign way as to send message with a smaller cost than a normal neighbor of the targeted node. In other words, these compromised nodes exchange a state message similar to normal nodes in order to maintain communication with them. Therefore, such links need not to be cut down to achieve the network resilience. Figure 5.13 shows the estimation dynamics of normal nodes by R-DLMSAW. We find none of the attacked nodes are driven to the attacker's selected state. All the nodes successfully converge to their true goal states. The purple line in Figure 5.11 shows the steady-state MSD performance of R-DLMSAW with $F = 1$.



(a) $\boldsymbol{w}_{k,i}(1)$ (b) $\boldsymbol{w}_{k,i}(1)$

Figure 5.12: Estimation dynamics for stationary target localization by DLMSAW under weak attack.



(a) $\boldsymbol{w}_{k,i}(1)(F=1)$ (b) $\boldsymbol{w}_{k,i}(2)(F=1)$

Figure 5.13: Estimation dynamics for stationary target localization by R-DLMSAW under weak attack.

We observe that this line lies between the noncooperative LMS and DLMSAW (without attack), and has a much smaller steady-state MSD than DLMSAW under such attack. This illustrates the effectiveness of the proposed resilient diffusion algorithm by showing that the algorithm is resilient to not only strong but also to weak attacks, as well as other data falsification attacks.

5.10 Conclusion

In this chapter, we studied distributed diffusion for multi-task networks and investigated vulnerabilities introduced by adaptive weights. Cooperative diffusion is a powerful strategy to perform optimization and estimation tasks, however, its performance and accuracy can deteriorate significantly in the presence of adversarial nodes. In fact, cooperative diffusion performs significantly better (in terms of steady-state MSD) as compared to non-cooperative diffusion if there are no adversarial nodes. However, with adversaries, cooperative diffusion could be even worse than the non-cooperative diffusion. To illustrate this, we proposed attack models that can drive normal agents—implementing distributed diffusion (DLMSAW)—to any state selected by the attacker, for both stationary and non-stationary estimation. We then proposed a resilient distributed diffusion algorithm (R-DLMSAW) to counteract adversaries' effect. The proposed algorithm always performs at least as good as the non-cooperative diffusion, but if an input parameter F in the algorithm is selected appropriately, it performs significantly better than the non-cooperative diffusion in the presence of adversaries. We also analyzed how the performance of R-DLMSAW changes with the selection of parameter F by the nodes. We evaluated our approach by applying it to stationary and non-stationary multi-target localization. In future, we are interested in generalizing our model to other types of distributed diffusion algorithms and with the missing data. It is also worth investigating the relationship between the underlying network connectivity and the steady-state performance of such algorithms.

5.A Proofs

5.A.1 Proof for Lemma 5.1

The message received by a from $k \in \mathcal{N}_a$ is $\boldsymbol{\psi}_{k,i}$. Agent a can compute $\boldsymbol{w}_{k,i-1}$ from $\boldsymbol{\psi}_{k,i}$ using

$$\boldsymbol{w}_{k,i-1} = \boldsymbol{\psi}_{k,i} - \mu_k \boldsymbol{u}_{k,i}^* (\boldsymbol{d}_k(i) - \boldsymbol{u}_{k,i} \boldsymbol{w}_{k,i-1}),$$

from which it can compute $\boldsymbol{w}_{k,i-1}$ as:

$$\boldsymbol{w}_{k,i-1} = \frac{\boldsymbol{\psi}_{k,i} - \mu_k \boldsymbol{u}_{k,i}^* \boldsymbol{d}_k(i)}{1 - \mu_k \boldsymbol{u}_{k,i}^* \boldsymbol{u}_{k,i}}.$$

Given the knowledge of μ_k, $\boldsymbol{d}_k(i)$, and $\boldsymbol{u}_{k,i}$, the value $\boldsymbol{w}_{k,i-1}$ can be computed exactly.

5.A.2 Proof for Lemma 5.2

We use $\delta_{a,k,i}$ to denote $\|\boldsymbol{\psi}_{a,i} - \boldsymbol{w}_{k,i-1}\|$, and $\delta_{l,k,i}$ to denote $\|\boldsymbol{\psi}_{l,i} - \boldsymbol{w}_{k,i-1}\|$, for $l \in \mathcal{N}_k, l \neq a$. Since

$$\gamma_{lk}^2(i) = (1-\nu_k)\gamma_{lk}^2(i-1) + \nu_k \|\boldsymbol{\psi}_{l,i} - \boldsymbol{w}_{k,i-1}\|^2, l \in \mathcal{N}_k,$$

Suppose the attack starts at i_a, then at iteration $(i_a + n)$,

$$\begin{aligned}
&\gamma_{ak}^2(i_a+n) \\
=&(1-\nu_k)\gamma_{ak}^2(i_a+n-1) + \nu_k \delta_{a,k,i_a+n}^2 \\
=&(1-\nu_k)((1-\nu_k)\gamma_{ak}^2(i_a+n-2) + \nu_k \delta_{a,k,i_a+n-1}^2) \\
&+ \nu_k \delta_{a,k,i_a+n}^2 \\
=&(1-\nu_k)^{n+1}\gamma_{ak}^2(i_a-1) \\
&+ \nu_k[(1-\nu_k)^n \delta_{a,k,i_a}^2 + (1-\nu_k)^{n-1}\delta_{a,k,i_a+1}^2 \\
&+ \ldots + (1-\nu_k)\delta_{a,k,i_a+n-1}^2 + \delta_{a,k,i_a+n}^2],
\end{aligned}$$

$$\begin{aligned}
\gamma_{lk}^2(i_a+n) =&(1-\nu_k)^{n+1}\gamma_{lk}^2(i_a-1) \\
&+ \nu_k[(1-\nu_k)^n \delta_{l,k,i_a}^2 + (1-\nu_k)^{n-1}\delta_{l,k,i_a+1}^2 \\
&+ \ldots + (1-\nu_k)\delta_{l,k,i_a+n-1}^2 + \delta_{l,k,i_a+n}^2].
\end{aligned}$$

For large enough n, $(1-\nu_k)^{n+1} \to 0$. Since we assume $\|\boldsymbol{\psi}_{a,i} - \boldsymbol{w}_{k,i-1}\| \ll \|\boldsymbol{\psi}_{l,i} - \boldsymbol{w}_{k,i-1}\|$, i.e., $\delta_{a,k,i} \ll \delta_{l,k,i}$, for $i \geq i_a + n$, $\gamma_{ak}^2(i) \ll \gamma_{lk}^2(i)$ holds. Thus,

$$\frac{a_{lk}(i)}{a_{ak}(i)} \propto \frac{\gamma_{lk}^{-2}(i)}{\gamma_{ak}^{-2}(i)} \to 0. \tag{22}$$

Given the property of weights, (5.13) is true.

5.A.3 Proof for Lemma 5.3

We use $\mathcal{A}$ to denote the set of compromised nodes targeting at the same normal node k. The proposed attack strategy results in the following condition holding as proved in *Lemma 5.2*:

$$\begin{aligned} &\frac{a_{lk}(i)}{a_{ak}(i)} \to 0, \qquad l \in \mathcal{N}_k \backslash \mathcal{A}, a \in \mathcal{A}, \\ &(i \geq i_a + n, \text{ subject to } (1-\nu_k)^{n+1} = 0). \end{aligned}$$

Given that $\sum_{l \in \mathcal{N}_k} a_{lk} = 1$, we have

$$a_{lk}(i) = 0, a_{ak}(i) = \frac{1}{|\mathcal{A}|}, \qquad l \in \mathcal{N}_k \backslash \mathcal{A}, a \in \mathcal{A},$$

where $|\mathcal{A}|$ denotes the number of nodes in $\mathcal{A}$. Since every compromised node $a \in \mathcal{A}$ sends the same message and is assigned the same weight that sums up to 1, it is equivalent to only one compromised node attacking the target node and being assigned a weight of 1. Therefore, there is no need for multiple compromised nodes attacking a single normal node.

5.A.4 Proof for Proposition 5.1

The constraint of r_k^a is consistent with the condition of Lemma 5.2. Thus, from some point i, the state of node k will be attacked as to be:

$$\begin{aligned} \boldsymbol{w}_{k,i} &= \boldsymbol{w}_{k,i-1} - r_k^a(\boldsymbol{w}_{k,i-1} - x_i) \\ &= r_k^a x_i + (1 - r_k^a)\boldsymbol{w}_{k,i-1}, \\ &(i \geq i_a + n, \text{ subject to } (1-\nu_k)^{n+1} = 0). \end{aligned} \tag{23}$$

Let X_i be $\boldsymbol{w}_{k,i}$, X_{i-1} be $\boldsymbol{w}_{k,i-1}$, A_i be $r_k^a x_i$, and B be $(1 - r_k^a)$. Equation (23) turns to:

$$X_i = A_i + BX_{i-1}. \tag{24}$$

Assume $\lim_{i\to\infty} X_{i-1} = X_{i-1}^0$ and $\lim_{i\to\infty} X_i = X_i^0$, then for $i \to \infty$ we get:

$$X_i^0 = A_i + BX_{i-1}^0. \tag{25}$$

Subtract (25) from (24), we get $X_i - X_i^0 = B(X_{i-1} - X_{i-1}^0)$. Let $\varepsilon_i = X_i - X_i^0$, for $i = 0, 1, 2, \ldots$, then $\varepsilon_i = B\varepsilon_{i-1} = B^2\varepsilon_{i-2} = \ldots = B^i\varepsilon_0$. The necessary and sufficient requirement for convergence is $\lim_{i\to\infty} \varepsilon_i = 0$ or, $\lim_{i\to\infty} B^i\varepsilon_0 = 0$, that is,

$$\lim_{i\to\infty} B^i = 0. \tag{26}$$

Therefore, we get a necessary and sufficient requirement for convergence as $|B| < 1$. Since $B = 1 - r_k^a$, and $r_k^a \in (0, 1)$, we get $B \in (0, 1)$. Therefore, $\lim_{i\to\infty}(X_i - X_i^0) = 0$. The assumption $\lim_{i\to\infty} X_i = X_i^0$ holds, and therefore, X_i is convergent to X_i^0.

To get the value of X_i^0, we need to analyze the following two scenarios: stationary state estimation and non-stationary state estimation, separately.

5.A.4.1 Stationary state estimation

In stationary scenarios, the convergence state is independent of time, that is, $X_i^0 = X_{i-1}^0 = X^0$. Therefore, equation (25) turns to:

$$X^0 = A_i + BX^0.$$

Thus, $(1 - B)X^0 = A_i$, $X^0 = \frac{A_i}{1-B}$. The convergent point is:

$$\boldsymbol{w}_{k,i} = \frac{r_k^a x_{i+1}}{1 - (1 - r_k^a)} = \frac{r_k^a w_k^a}{1 - (1 - r_k^a)} = w_k^a = \boldsymbol{w}_{k,i}^a, \quad i \to \infty$$

which realizes the attacker's objective (5.7).

5.A.4.2 Non-stationary state estimation

In non-stationary scenarios, we first assume $x_i = w_k^a + \theta_{k,i-1}^a$ and later we will show how $\theta_{k,i-1}^a$ turns to $\theta_{k,i-1}^a + \frac{\Delta\theta_{k,i-1}^a}{r_k^a}$.

Assume the convergence point X_i^0 is a combination of a time-independent value and a time-dependent value, such that $X_i^0 = X^0 + \rho_i$. After taking original values into (25), we get

$$X^0 + \rho_i = r_k^a(w_k^a + \theta_{k,i-1}^a) + (1 - r_k^a)(X_0 + \rho_{i-1}). \tag{27}$$

Next, we divide (27) into the time-independent and time-dependent components to get

$$X^0 = w_k^a, \quad \rho_i - \rho_{i-1} = r_k^a(\theta_{k,i-1}^a - \rho_{i-1}).$$

Let $\Delta\rho_{i-1} = \rho_i - \rho_{i-1}$, we get:

$$\rho_{i-1} = \theta_{k,i-1}^a - \frac{\Delta\rho_{i-1}}{r_k^a} \quad \text{and} \quad \rho_i = \theta_{k,i}^a - \frac{\Delta\rho_i}{r_k^a}. \tag{28}$$

Thus, $\Delta\rho_{i-1} = \rho_i - \rho_{i-1} = \theta_{k,i}^a - \theta_{k,i-1}^a - \frac{1}{r_k^a}(\Delta\rho_i - \Delta\rho_{i-1})$. Let $\Delta\theta_{k,i-1}^a = \theta_{k,i}^a - \theta_{k,i-1}^a$ and $\Delta^2\rho_{i-1} = \Delta\rho_i - \Delta\rho_{i-1}$, then $\Delta\rho_{i-1} = \Delta\theta_{k,i-1}^a - \frac{\Delta^2\rho_{i-1}}{r_k^a}$ or $\Delta\rho_i = \Delta\theta_{k,i}^a - \frac{\Delta^2\rho_i}{r_k^a}$. If we assume $\frac{\Delta^2\rho_i}{r_k^a} \ll \Delta\theta_{k,i}^a$, then we have $\Delta\rho_i = \Delta\theta_{k,i}^a$. Therefore, (28) can be written as $\rho_i = \theta_{k,i}^a - \frac{\Delta\theta_{k,i}^a}{r_k^a}$. Thus, the dynamic convergence point for k is

$$\boldsymbol{w}_{k,i} = w_k^a + \theta_{k,i}^a - \frac{\Delta\theta_{k,i}^a}{r_k^a}, \qquad i \to \infty.$$

This means when sending $\boldsymbol{\psi}_{a,i} = \boldsymbol{w}_{k,i-1} + r_k^a(w_k^a + \theta_{k,i-1}^a - \boldsymbol{w}_{k,i-1})$ as the communication message, the compromised node a can make k converge to $w_k^a + \theta_{k,i}^a - \frac{\Delta\theta_{k,i}^a}{r_k^a}$. To make agent k converge to a desired state $w_k^a + \Omega_{k,i}^a$, we assume the message sent is

$$\boldsymbol{\psi}_{a,i} = \boldsymbol{w}_{k,i-1} + r_k^a(w_k^a + m_{i-1} - \boldsymbol{w}_{k,i-1}).$$

The corresponding convergence point will be $w_k^a + m_i - \frac{\Delta m_i}{r_k^a}$. We want the following equation to hold,

$$w_k^a + m_i - \frac{\Delta m_i}{r_k^a} = w_k^a + \Omega_{k,i}^a. \tag{29}$$

Assuming $\Delta^2 m_i \to 0$, the solution of (29) is: $m_i = \Omega_{k,i}^a + \frac{\Delta\Omega_{k,i}^a}{r_k^a}$, meaning to make k converge to a desired state $w_k^a + \Omega_{k,i}^a$, the compromised node a should send communication message:

$$\boldsymbol{\psi}_{a,i} = \boldsymbol{w}_{k,i-1} + r_k^a(w_k^a + \Omega_{k,i-1}^a + \frac{\Delta\Omega_{k,i-1}^a}{r_k^a} - \boldsymbol{w}_{k,i-1}).$$

Thus, to make k converge to $w_k^a + \theta_{k,i}^a$, the compromised node a should send communication message:

$$\boldsymbol{\psi}_{a,i} = \boldsymbol{w}_{k,i-1} + r_k^a(w_k^a + \theta_{k,i-1}^a + \frac{\Delta\theta_{k,i-1}^a}{r_k^a} - \boldsymbol{w}_{k,i-1}).$$

The convergence point is:

$$\boldsymbol{w}_{k,i} = w_k^a + \theta_{k,i}^a = \boldsymbol{w}_{k,i}^a, \qquad i \to \infty,$$

which realizes the attacker's objective (5.7).

We can verify the convergence point by putting $x_i = w_k^a + \theta_{k,i-1}^a + \frac{\Delta\theta_{k,i-1}^a}{r_k^a}, \boldsymbol{w}_{k,i} = w_k^a + \theta_{k,i}^a, \boldsymbol{w}_{k,i-1} = w_k^a + \theta_{k,i-1}^a$ back into equation (23), we get:

$$\begin{aligned} w_k^a + \theta_{k,i}^a &= r_k^a(w_k^a + \theta_{k,i-1}^a + \frac{\Delta\theta_{k,i-1}^a}{r_k^a}) + (1 - r_k^a)(w_k^a + \theta_{k,i-1}^a) \\ \theta_{k,i}^a &= r_k^a(\theta_{k,i-1}^a + \frac{\Delta\theta_{k,i-1}^a}{r_k^a}) + (1 - r_k^a)\theta_{k,i-1}^a \\ \theta_{k,i}^a &= \theta_{k,i-1}^a + \Delta\theta_{k,i-1}^a. \end{aligned}$$

The resulting equation holds, illustrating the validity of the convergence state.

Chapter 6

Byzantine Resilient Distributed Multi-Task Learning [1]

Distributed multi-task learning provides significant advantages in multi-agent networks with heterogeneous data sources where agents aim to learn distinct but correlated models simultaneously. However, distributed algorithms for learning relatedness among tasks are not resilient in the presence of Byzantine agents. In this chapter, we present an approach for Byzantine resilient distributed multi-task learning. We propose an efficient online weight assignment rule by measuring the accumulated loss using an agent's data and its neighbors' models. A small accumulated loss indicates a large similarity between the two tasks. In order to ensure the Byzantine resilience of the aggregation at a normal agent, we introduce a step for filtering out larger losses. We analyze the approach for convex models and show that normal agents converge resiliently towards the global minimum. Further, aggregation with the proposed weight assignment rule always results in an improved expected regret than the non-cooperative case. Finally, we demonstrate the approach using three case studies, including regression and classification problems, and show that our method exhibits good empirical performance for non-convex models, such as convolutional neural networks.

6.1 Introduction

Distributed machine learning models are gaining much attention recently as they improve the learning capabilities of agents distributed within a network with no central entity. In a distributed multi-agent system, agents interact with each other to improve their learning capabilities by leveraging the shared information via exchanging either data or models. In particular, agents that do not have enough data to build refined models or agents that have limited computational capabilities, benefit most from such cooperation. Distributed learning also addresses the single point of failure problem as well as scalability issues and is naturally suited to mobile phones, autonomous vehicles, drones, healthcare, smart cities, and many other applications [48–51].

In networks with heterogeneous data sources, it is natural to consider the multi-task learning (MTL) framework, where agents aim to learn distinct but correlated models simultaneously [15]. Typically, prior knowledge of the relationships among models is assumed in MTL. The relationships among agents can be promoted via several methods, such as mean regularization, clustered regularization, low-rank and sparse

[1] ©2020 NeurIPS. Adapted with permission, from [Jiani Li, Waseem Abbas, and Xenofon Koutsoukos. "Byzantine Resilient Distributed Multi-Task Learning". In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems (NeurIPS) 2020, December 6-12. 2020. URL: https : / / proceedings .neurips.cc/paper/2020/hash/d37eb50d868361ea729bb4147eb3c1d8-Abstract.html].

structures regularization [91–93]. However, in real-world applications, such relationships are unknown beforehand and need to be estimated online from data. Learning similarities among tasks to promote effective cooperation is a primary consideration in MTL. There has been extensive work for learning the relationship matrix *centrally* by optimizing a global convex regularized function [94–96]. In contrast, this chapter focuses on computationally efficient *distributed* learning of the relationship among agents that does not require optimizing a relationship matrix centrally [60, 97–99].

Although the distributed approach to learning and promoting similarities among neighbors from online data has many advantages, it is not resilient to Byzantine agents. Fault-tolerance for MTL is discussed in [15], focusing on dropped nodes that occasionally stop sending information to their neighbors. In [2], the relationship promoted by measuring the quadratic distance between two model parameters for distributed MTL is shown to be vulnerable to gradient-based attacks, and a Byzantine resilient distributed MTL algorithm is proposed for regression problems to cope with such attacks. The proposed algorithm relies on a user-defined parameter F to filter out information from F neighbors in the aggregation step and is resilient to F Byzantine neighbors, but requires exponential time with respect to the number of neighbors.

In this chapter, we propose an *online weight adjustment rule* for MTL that is guaranteed to achieve resilient convergence for every normal agent using the rule. Compared to [2], the proposed method is suited for both regression and classification problems, is resilient to an arbitrary number of Byzantine neighbors (without the need to select a pre-defined parameter F bounding the number of Byzantine neighbors), and has linear time complexity. To the best of our knowledge, this is the first solution that aims to address the Byzantine resilient cooperation in distributed MTL networks via a resilient similarity promoting method. We note that the proposed rule is not limited to the multi-task setting but can also be used for general distributed machine learning and federated learning systems to achieve resilient consensus. We list our contributions below.

- We propose an efficient Byzantine resilient online weight adjustment rule for distributed MTL. We measure similarities among agents based on the accumulated loss of an agent's data and the models of its neighbors. In each iteration, a normal agent computes the weights assigned to its neighbors in time that is linear in the size of its neighborhood and the dimension of the data.

- We show that aggregation with the proposed weight assignment rule always results in an improved expected regret than the non-cooperative case, and normal agents converge resiliently towards the global minimum. Even when all the neighbors are Byzantine, a normal agent can still resiliently converge to the global minimum bounded by the same expected regret as without any cooperation with other agents, achieving resilience to an arbitrary number of Byzantine agents.

- We conduct three experiments for both regression and classification problems and demonstrate that our approach yields good empirical performance for non-convex models, such as convolutional neural networks.

6.2 Related Work

Multi-Task Learning. MTL deals with the problem of learning multiple related tasks simultaneously to improve the generalization performance of the models learned by each task with the help of the other auxiliary tasks [85, 86]. The extensive literature in MTL can be broadly categorized into two categories based on how the data is collected. The *centralized* approach assumes the data is collected beforehand at a centralized entity. Many successful MTL applications with deep networks, such as in natural language processing and computer vision, fall into this category [87–90]. This approach usually learns multiple objectives from a shared representation by sharing layers and splitting architecture in the deep networks. On the other hand, the *distributed* approach assumes data is collected separately by each task in a distributed manner. This approach is naturally suited to model distributed learning in multi-agent systems such as mobile phones, autonomous vehicles, and smart cities [49–51]. We focus on distributed MTL in this chapter.

Relationship Learning in MTL. Although it is often assumed that a clustered, sparse, or low-rank structure among tasks is known *a priori* [91–93], such information may not be available in many real-world applications. Learning the relatedness among tasks online from data to promote effective cooperation is a principle approach in MTL when the relationships among tasks are not known *a priori*. There has been extensive work in online relationship learning that can be broadly categorized into centralized and distributed methods. The first group assumes that a centralized server collects the task models and utilizes a convex formulation of the regularized MTL optimization problem over the relationship matrix, which is learned by solving the convex optimization problem [94–96]. The second group relies on a distributed architecture in which agents learn relationships with their neighbors based on the similarities of their models and accordingly adjust weights assigned to neighbors [60, 97–99]. Typical similarity metrics, such as $\mathcal{H}$ divergence [55, 56, 100] and Wasserstein distance [56, 101], can be used in MTL in the same way they are used in domain adaptation, transfer learning, and adversarial learning. However, such metrics are mainly designed for measuring the divergence in data distributions and are not suitable for online relationship learning due to efficiency and privacy concerns in data sharing.

Resilient Aggregation in Distributed ML. Inspired by the resilient consensus algorithms in multi-agent networks [130, 181], various resilient aggregation rules have been adapted in distributed ML, including the coordinate-wise trimmed mean [52], the coordinate-wise median [52, 63, 64], the geometric median [58, 62], and the Krum algorithm [18]. However, studies have shown that these rules are not resilient against certain

attacks [78–80]. The centerpoint based aggregation rule [4] has been proposed recently that guarantees resilient distributed learning to Byzantine attacks. However, since each agent fits a distinct model in MTL, consensus-based resilient aggregation rules are not directly applicable to MTL.

6.3 Distributed Multi-Task Learning

Background. Consider a network of n agents[2] modeled by an *undirected graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents agents and $\mathcal{E}$ represents interactions between agents. A bi-directional edge $(l, k) \in \mathcal{E}$ means that agents k and l can exchange information with each other. Since each agent also has its own information, we have $(k, k) \in \mathcal{E}, \forall k \in \mathcal{V}$. The *neighborhood* of k is the set $\mathcal{N}_k = \{l \in \mathcal{V} | (l, k) \in \mathcal{E}\}$. Each agent k has data $\left\{(x_k^i, y_k^i)\right\}$ sampled randomly from the distribution generated by the random variable ξ_k, where $x_k^i \in \mathbb{R}^{d_x}$, $y_k^i \in \mathbb{R}^{d_y}$. We use $\ell(\theta_k; \xi_k)$ to denote a convex *loss function* associated with the prediction function parameterized by θ_k for agent k. MTL is concerned with fitting separate models θ_k to the data for agent k via the *expected risk function* $r_k(\theta_k) = \mathbb{E}\left[\ell(\theta_k; \xi_k)\right]$. We use θ_k^* to denote the global minimum of the convex function $r_k(\theta_k)$. The model parameters can be optimized via the following objective function:

$$\min_{\Theta} \left\{ \sum_{k=1}^{n} r_k(\theta_k) + \eta \mathcal{R}(\Theta, \Omega) \right\}, \tag{6.1}$$

where $\Theta = [\theta_1, \ldots, \theta_n] \in \mathbb{R}^{d_x \times n}$, $\mathcal{R}(\cdot)$ is a convex regularization function promoting the relationships among the agents, and $\Omega \in \mathbb{R}^{n \times n}$ models the relationships among the agents that can be assigned *a priori* or can be estimated from data. An example of the regularizer takes the form of $\mathcal{R}(\Theta, \Omega) = \lambda_1 \mathrm{Tr}(\Theta \Omega \Theta^\top) + \lambda_2 \mathrm{Tr}(\Theta \Theta^\top)$, where λ_1, λ_2 are non-negative parameters. In a *centralized* setting, where a centralized server optimizes the relationship matrix by collecting the models of agents, an optimal solution $\Omega = \frac{(\Theta^\top \Theta)^{\frac{1}{2}}}{\mathrm{Tr}((\Theta^\top \Theta))^{\frac{1}{2}}}$ is proposed in [95] for learning the structure of clustered MTL using the above regularizer. In the *distributed* case, the task relationships Ω are not learned centrally and we can use the *adapt-then-combine (ATC) diffusion* algorithm [182] as a projection-based distributed solution of (6.1):

$$\hat{\theta}_{k,i} = \theta_{k,i-1} - \mu_k \nabla \ell(\theta_{k,i-1}; \xi_k^{i-1}), \qquad \text{(adaptation)} \tag{6.2}$$

$$\theta_{k,i} = \sum_{l \in \mathcal{N}_k} a_{lk} \hat{\theta}_{l,i}, \text{subject to} \sum_{l \in \mathcal{N}_k} a_{lk} = 1, a_{lk} \geq 0, a_{lk} = 0 \text{ if } l \notin \mathcal{N}_k, \qquad \text{(combination)} \tag{6.3}$$

where $\mathcal{N}_k$ is the neighborhood of agent k, μ_k is the step size, and a_{lk} denotes the weight assigned by agent k to l, which should accurately reflect the similarity relationships among agents[3]. $\nabla \ell(\theta_{k,i-1}; \xi_k^{i-1})$ is the gradient using the instantaneous realization ξ_k^{i-1} of the random variable ξ_k. At each iteration i, agent k minimizes

[2] Each agent is modeled as a separate task, thus, the terms *agent* and *task* are used interchangeably.

[3] μ_k and a_{lk} can be time-dependent, but when context allows, we write $\mu_{k,i}$ as μ_k and $a_{lk}(i)$ as a_{lk} for simplicity.

the individual risk using stochastic gradient descent (SGD) given local data followed by a combination step that aggregates neighboring models according to the weights assigned to them. The weights $\{a_{lk}\}$ are free parameters selected by the designer and they serve the same purpose as Ω in a centralized formulation. Thus, there is no need to design Ω in the case of distributed MTL that utilizes ATC diffusion algorithm for aggregation [105].

Online Weight Adjustment Rules. Without knowing the relationships *a priori*, one can assume the existence of similarities among agents and can learn these similarities online from data. The approach is based on the distance between the model parameters of agents, where a small distance indicates a large similarity [60, 97, 183, 184]. A common approach to learning similarities between two agents online is given by

$$a_{lk}(i) = \frac{\|\tilde{\theta}_k^* - \hat{\theta}_{l,i}\|^{-2}}{\sum_{p\in\mathcal{N}_k} \|\tilde{\theta}_k^* - \hat{\theta}_{p,i}\|^{-2}}, \tag{6.4}$$

where $\tilde{\theta}_k^*$ is an approximation of θ_k^* since θ_k^* is unknown. Examples include using the current model $\tilde{\theta}_k^* = \theta_{k,i-1}$, and one-step ahead approximation $\tilde{\theta}_k^* = \hat{\theta}_{k,i} + \mu_k \nabla \ell(\hat{\theta}_{k,i}; \xi_k^{i-1})$. Although the ℓ_2 norm is widely used, this formulation of weights can be generalized to ℓ_p norm as well.

6.4 Problem Formulation

Byzantine agents can send arbitrary different information to different neighbors usually with a malicious goal of disrupting the network's convergence. It has been shown in [2] that normal agents assigning weights according to (6.4) are vulnerable to Byzantine agents. Particularly, by sending $\|\hat{\theta}_{b,i} - \tilde{\theta}_k^*\| \ll \|\hat{\theta}_{k,i} - \tilde{\theta}_k^*\|$, a Byzantine agent b can gain a large weight from k and continuously drive its normal neighbor k towards a desired malicious point.

To address the vulnerabilities of the online weight adjustment rules derived from (6.4), this chapter aims to design an efficient resilient online weight assignment rule in the presence of Byzantine agents for MTL. Let the *expected regret* $\mathbb{E}[r_k(\theta_{k,i}) - r_k(\theta_k^*)]$ be the value of the expected difference between the risk of $\theta_{k,i}$ and the optimal decision θ_k^*. We aim to design weights $A_k = [a_{1k}, \ldots, a_{nk}] \in \mathbb{R}^{1\times n}$ for a normal agent k that satisfy the following conditions:

Resilient Convergence. It must be guaranteed that using the computed weights A_k, every normal agent k resiliently converges to θ_k^*, even in the presence of Byzantine neighbors.

Improved Learning Performance. Cooperation among agents is meaningful only when it improves the learning performance. Hence, it is important to guarantee that for every normal agent, the combination step using the computed weights A_k always results in an improved expected regret, even in the presence of

Byzantine agents, i.e.,

$$\mathbb{E}[r_k(\theta_{k,i}) - r_k(\theta_k^*)] \leq \mathbb{E}[r_k(\hat{\theta}_{k,i}) - r_k(\theta_k^*)], \forall k \in \mathcal{N}^+, i \in \mathbb{N} \tag{6.5}$$

Computational Efficiency. At each iteration, a normal agent k needs to compute the weights A_k in time that is linear in the size of the neighborhood of k and the dimension of the data, i.e., in $O(|\mathcal{N}_k|(d_x + d_y))$ time.

6.5 Loss-based Online Weight Adjustment

6.5.1 Weight Optimization

We follow a typical approach of learning the optimal weight adjustment rule [60, 97, 183, 184] in which the goal is to minimize the quadratic distance between the aggregated model $\theta_{k,i}$ and the true model θ_k^* over the weights, i.e., $\min_{A_k} \|\theta_{k,i} - \theta_k^*\|^2$. Using (6.3), we get an equivalent problem:

$$\min_{A_k} \left\| \sum_{l \in \mathcal{N}_k} a_{lk} \hat{\theta}_{l,i} - \theta_k^* \right\|^2, \text{ subject to } \sum_{l \in \mathcal{N}_k} a_{lk} = 1, a_{lk} \geq 0, a_{lk} = 0 \text{ if } l \notin \mathcal{N}_k,$$

where $\left\| \sum_{l \in \mathcal{N}_k} a_{lk} \hat{\theta}_{l,i} - \theta_k^* \right\|^2 = \sum_{l \in \mathcal{N}_k} \sum_{p \in \mathcal{N}_k} a_{lk} a_{pk} (\hat{\theta}_{l,i} - \theta_k^*)^\top (\hat{\theta}_{p,i} - \theta_k^*)$. As in a typical approximation approach, we consider

$$\left\| \sum_{l \in \mathcal{N}_k} a_{lk} \hat{\theta}_{l,i} - \theta_k^* \right\|^2 \approx \sum_{l \in \mathcal{N}_k} a_{lk}^2 \left\| \hat{\theta}_{l,i} - \theta_k^* \right\|^2. \tag{6.6}$$

The weight assignment rule (6.4) is an optimal solution of (6.6) using the approximation of θ_k^*, which as we discuss above, can be easily attacked. To avoid the use of the distance between model parameters as a similarity measure, we introduce a *resilient* counterpart, which is the *accumulated loss* (or risk). Assume risk functions r_k to be m-strongly convex[4], then it holds that

$$r_k(\hat{\theta}_{l,i}) - r_k(\theta_k^*) \geq \langle \nabla r_k(\theta_k^*), \hat{\theta}_{l,i} - \theta_k^* \rangle + \frac{m}{2} \|\hat{\theta}_{l,i} - \theta_k^*\|^2,$$

where $r_k(\hat{\theta}_{l,i}) = \mathbb{E}\left[\ell(\hat{\theta}_{l,i}; \xi_k)\right]$. Since $\nabla r_k(\theta_k^*) = 0$, we obtain

$$\|\hat{\theta}_{l,i} - \theta_k^*\|^2 \leq \frac{2}{m} \left(r_k(\hat{\theta}_{l,i}) - r_k(\theta_k^*) \right). \tag{6.7}$$

Instead of directly minimizing the right side of (6.6), we consider minimizing its upper bound given in (6.7). Later in Section 6.6, we show that this alternate approach facilitates the resilient distributed MTL, which

[4] See Definition 6.2.

cannot be achieved by minimizing the distance between models directly. Hence, by combining (6.6) and (6.7), we consider the following minimization problem:

$$\min_{A_k} \sum_{l\in\mathcal{N}_k} a_{lk}^2 \left(r_k(\hat{\theta}_{l,i}) - r_k(\theta_k^*)\right) \text{ subject to } \sum_{l\in\mathcal{N}_k} a_{lk} = 1, a_{lk} \geq 0, a_{lk} = 0 \text{ if } l \notin \mathcal{N}_k.$$

This optimization problem indicates that if a neighbor l's model has a small regret on agent k's data distribution, then it should be assigned a large weight. Since θ_k^* is unknown, one can use $r_k(\theta_{k,i})$ to approximate $r_k(\theta_k^*)$. Alternatively, since $r_k(\theta_k^*)$ is small compared to $r_k(\theta_{l,i})$, we could simply assume $r_k(\theta_k^*) = 0$ and consider the following minimization problem:

$$\min_{A_k} \sum_{l\in\mathcal{N}_k} a_{lk}^2 r_k(\hat{\theta}_{l,i}) \text{ subject to } \sum_{l\in\mathcal{N}_k} a_{lk} = 1, a_{lk} \geq 0, a_{lk} = 0 \text{ if } l \notin \mathcal{N}_k. \tag{6.8}$$

Let λ be the Lagrange multiplier. We define the Lagrangian of (6.8) given the constraints on the weights as

$$\mathcal{L}(a_{lk}, \lambda) = \sum_{l\in\mathcal{N}_k} a_{lk}^2 r_k(\hat{\theta}_{l,i}) + \lambda(1 - \sum_{l\in\mathcal{N}_k} a_{lk}).$$

Set $\nabla_{a_{lk},\lambda}\mathcal{L}(a_{lk}, \lambda) = \left(\frac{\partial\mathcal{L}}{\partial a_{lk}}, \frac{\partial\mathcal{L}}{\partial\lambda}\right) = 0$, i.e.,

$$\begin{cases} 2a_{lk}r_k(\hat{\theta}_{l,i}) - \lambda = 0, \forall l \in \mathcal{N}_k, \\ 1 - \sum_{l\in\mathcal{N}_k} a_{lk} = 0. \end{cases}$$

Thus, $a_{lk} = \frac{\lambda}{r_k(\hat{\theta}_{l,i})}, \forall l \in \mathcal{N}_k$ and $\sum_{l\in\mathcal{N}_k} a_{lk} = 1$. We have $\lambda \sum_{l\in\mathcal{N}_k} \frac{1}{r_k(\hat{\theta}_{l,i})} = 1$ and hence $\lambda = \frac{1}{\sum_{l\in\mathcal{N}_k} r_k(\hat{\theta}_{l,i})^{-1}}$. Using the Lagrangian relaxation, we obtain the optimal solution of (6.8) as

$$a_{lk}(i) = \frac{r_k(\hat{\theta}_{l,i})^{-1}}{\sum_{p\in\mathcal{N}_k} r_k(\hat{\theta}_{p,i})^{-1}}. \tag{6.9}$$

We can approximate $r_k(\hat{\theta}_{l,i})$ using the exponential moving average $\varphi_{lk}^i = (1-\nu_k)\varphi_{lk}^{i-1} + \nu_k \ell(\hat{\theta}_{l,i}; \xi_k)$, where ν_k is the forgetting factor. Given $\mathbb{E}[\varphi_{lk}^i] = (1-\nu_k)\mathbb{E}[\varphi_{lk}^{i-1}] + \nu_k \mathbb{E}[\ell(\hat{\theta}_{l,i}; \xi_k)]$, we obtain $\lim_{i\to\infty} \mathbb{E}[\varphi_{lk}^i] = \lim_{i\to\infty} \mathbb{E}[\ell(\hat{\theta}_{l,i}; \xi_k)] = \lim_{i\to\infty} r_k(\hat{\theta}_{l,i})$, which means φ_{lk}^i converges (in expectation) to $\lim_{i\to\infty} r_k(\hat{\theta}_{l,i})$. Hence, we can use φ_{lk}^i to approximate $r_k(\hat{\theta}_{l,i})$. Note that in addition to the smoothing methods, one can use the average batch loss to approximate $r_k(\hat{\theta}_{l,i})$ when using the (mini-) batch gradient descent in the place of SGD for adaptation.

6.5.2 Filtering for Resilience

Let $\mathcal{N}_k^+$ denote the set of k's normal neighbors with $|\mathcal{N}_k^+| \geq 1$. We assume there are q Byzantine neighbors in the set $\mathcal{B} = \mathcal{N}_k \backslash \mathcal{N}_k^+$. In the following, we examine the resilience of the cooperation using (6.9) in the presence of Byzantine agents.

Lemma 6.1. *The following condition holds for the combination step* (6.3) *using weights* (6.9)*:*

$$\mathbb{E}\left[r_k(\theta_{k,i}) - r_k(\theta_k^*)\right] \leq \frac{1}{|\mathcal{N}_k|} \sum_{l \in \mathcal{N}_k} \mathbb{E}\left[r_k\left(\hat{\theta}_{l,i}\right) - r_k(\theta_k^*)\right].$$

Proof. Given (6.3), $r_k(\theta_{k,i}) = r_k\left(\sum_{l \in \mathcal{N}_k} a_{lk}(i)\hat{\theta}_{l,i}\right)$. Using Jensen's inequality, we have

$$r_k(\theta_{k,i}) \leq \sum_{l \in \mathcal{N}_k} a_{lk}(i) r_k\left(\hat{\theta}_{l,i}\right). \tag{6.10}$$

Subtracting $r_k(\theta_k^*)$ from both sides of (6.10) and taking expectations over the joint distribution ξ_k, we obtain

$$\begin{aligned}\mathbb{E}\left[r_k(\theta_{k,i}) - r_k(\theta_k^*)\right] &\leq \sum_{l \in \mathcal{N}_k} \mathbb{E}[a_{lk}(i)] \mathbb{E}\left[r_k\left(\hat{\theta}_{l,i}\right) - r_k(\theta_k^*)\right] \\ &\leq \frac{\sum_{l \in \mathcal{N}_k} \mathbb{E}\left[r_k(\hat{\theta}_{l,i})\right]^{-1} \mathbb{E}\left[r_k\left(\hat{\theta}_{l,i}\right) - r_k(\theta_k^*)\right]}{\sum_{p \in \mathcal{N}_k} \mathbb{E}\left[r_k(\hat{\theta}_{p,i})\right]^{-1}}.\end{aligned} \tag{6.11}$$

We next prove the right-hand side of (6.11) is less than $\frac{1}{|\mathcal{N}_k|} \sum_{l \in \mathcal{N}_k} \mathbb{E}\left[r_k(\hat{\theta}_{l,i}) - r_k(\theta_k^*)\right]$. For succinctness, we use $\chi_{l,i}$ to denote $\mathbb{E}\left[r_k\left(\hat{\theta}_{l,i}\right)\right]^{-1}$, and $\Delta_{l,i}$ to denote $\mathbb{E}\left[r_k\left(\hat{\theta}_{l,i}\right) - r_k(\theta_k^*)\right]$. And we aim to prove $\frac{\sum_{l \in \mathcal{N}_k} \chi_{l,i}\Delta_{l,i}}{\sum_{p \in \mathcal{N}_k} \chi_{p,i}} \leq \frac{1}{|\mathcal{N}_k|} \sum_{l \in \mathcal{N}_k} \Delta_{l,i}$, or equivalently, $|\mathcal{N}_k| \sum_{l \in \mathcal{N}_k} \chi_{l,i}\Delta_{l,i} \leq \sum_{p \in \mathcal{N}_k} \chi_{p,i} \sum_{l \in \mathcal{N}_k} \Delta_{l,i}$.

When $|\mathcal{N}_k| = 1$, one can easily validate that this condition holds. When $|\mathcal{N}_k| \geq 2$, let l_1^i be the one with the smallest risk $r_k\left(\hat{\theta}_{l_1^i,i}\right) = \min_{l \in \mathcal{N}_k} r_k\left(\hat{\theta}_{l,i}\right)$ and l_2^i be the one with the second smallest risk $r_k\left(\hat{\theta}_{l_2^i,i}\right) = \min_{l \in \mathcal{N}_k \backslash l_1^i} r_k\left(\hat{\theta}_{l,i}\right)$. Hence, $\chi_{l_1^i,i} \geq \chi_{l_2^i,i} \geq \chi_{l,i}$, and $\Delta_{l_1^i,i} \leq \Delta_{l_2^i,i} \leq \Delta_{l,i}$ for $l \in \mathcal{N}_k \backslash \{l_1^i, l_2^i\}$. Thus,

$$\begin{aligned}&|\mathcal{N}_k| \sum_{l \in \mathcal{N}_k} \chi_{l,i}\Delta_{l,i} - \sum_{p \in \mathcal{N}_k} \chi_{p,i} \sum_{l \in \mathcal{N}_k} \Delta_{l,i} = \sum_{l \in \mathcal{N}_k} \chi_{l,i} \left(|\mathcal{N}_k|\Delta_{l,i} - \sum_{p \in \mathcal{N}_k} \Delta_{p,i}\right) \\ =&\chi_{l_1^i,i} \left((|\mathcal{N}_k| - 1)\Delta_{l_1^i,i} - \sum_{l \in \mathcal{N}_k \backslash l_1^i} \Delta_{l,i}\right) + \sum_{l \in \mathcal{N}_k \backslash l_1^i,i} \chi_{l,i} \left(|\mathcal{N}_k|\Delta_{l,i} - \sum_{p \in \mathcal{N}_k} \Delta_{p,i}\right) \\ \leq&\chi_{l_1^i,i} \left((|\mathcal{N}_k| - 1)\Delta_{l_1^i,i} - \sum_{l \in \mathcal{N}_k \backslash l_1^i} \Delta_{l,i}\right) + \chi_{l_2^i,i} \left(\sum_{l \in \mathcal{N}_k \backslash l_1^i} |\mathcal{N}_k|\Delta_{l,i} - (|\mathcal{N}_k| - 1) \sum_{p \in \mathcal{N}_k} \Delta_{p,i}\right) \\ =&\chi_{l_1^i,i} \left((|\mathcal{N}_k| - 1)\Delta_{l_1^i,i} - \sum_{l \in \mathcal{N}_k \backslash l_1^i} \Delta_{l,i}\right) + \chi_{l_2^i,i} \left(\sum_{l \in \mathcal{N}_k \backslash l_1^i} \Delta_{l,i} - (|\mathcal{N}_k| - 1)\Delta_{l_1^i,i}\right)\end{aligned}$$

$$|\mathcal{N}_k| \sum_{l\in\mathcal{N}_k} \chi_{l,i}\Delta_{l,i} - \sum_{p\in\mathcal{N}_k} \chi_{p,i} \sum_{l\in\mathcal{N}_k} \Delta_{l,i} = \left(\chi_{l_1^i,i} - \chi_{l_2^i,i}\right) \left(\left(|\mathcal{N}_k| - 1\right) \Delta_{l_1^i,i} - \sum_{l\in\mathcal{N}_k\setminus l_1^i} \Delta_{l,i} \right)$$
$$= \left(\chi_{l_1^i,i} - \chi_{l_2^i,i}\right) \left(\sum_{l\in\mathcal{N}_k\setminus l_1^i} \left(\Delta_{l_1^i,i} - \Delta_{l,i}\right) \right) \leq 0.$$

Therefore, $\frac{\sum_{l\in\mathcal{N}_k} \chi_{l,i}\Delta_{l,i}}{\sum_{p\in\mathcal{N}_k} \chi_{p,i}} \leq \frac{1}{|\mathcal{N}_k|} \sum_{l\in\mathcal{N}_k} \Delta_{l,i}$. Put it back to (6.11), we obtain

$$\mathbb{E}\left[r_k(\theta_{k,i}) - r_k(\theta_k^*)\right] \leq \frac{1}{|\mathcal{N}_k|} \sum_{l\in\mathcal{N}_k} \mathbb{E}\left[r_k\left(\hat{\theta}_{l,i}\right) - r_k(\theta_k^*)\right],$$

which completes the proof. □

Since l can be a Byzantine agent, it is possible that $\mathbb{E}\left[r_k\left(\hat{\theta}_{l,i}\right) - r_k(\theta_k^*)\right]$ is a large value. Consequently, we cannot compute a useful upper bound on the value of $\mathbb{E}\left[r_k(\theta_{k,i}) - r_k(\theta_k^*)\right]$ given Lemma 6.1 and cannot provide further convergence guarantees. To facilitate the resilient cooperation, we consider a modification of (6.9) as follows.

$$a_{lk}(i) = \begin{cases} \frac{r_k(\hat{\theta}_{l,i})^{-1}}{\sum_{p\in\mathcal{N}_k^{\leq}} r_k(\hat{\theta}_{p,i})^{-1}}, & \text{if } r_k(\hat{\theta}_{l,i}) \leq r_k(\hat{\theta}_{k,i}), \\ 0, & \text{otherwise,} \end{cases} \tag{6.12}$$

where $\mathcal{N}_k^{\leq}$ denotes the set of neighbors with $r_k(\hat{\theta}_{l,i}) \leq r_k(\hat{\theta}_{k,i})$. This implies that the cooperation filters out the information coming from the neighbors incurring a larger risk and cooperate only with the remaining neighbors. In the next section, we show how this modification benefits learning and guarantees the resilient convergence of MTL.

6.5.3 Computational Complexity

It takes $O(d_x + d_y)$ time to compute $\ell(\hat{\theta}_{l,i}; \xi_k^i)$. Using the exponential moving average method for approximating $r_k(\hat{\theta}_{l,i})$, for a normal agent k, at each iteration i, the total time for computing $A_k(i)$ with the proposed rule (6.12) is $O(|\mathcal{N}_k|(d_x + d_y))$.

6.6 Byzantine Resilient Convergence Analysis

We make the following general assumptions for the convergence of SGD [156] to derive our results.

Definition 6.1. *(L-Lipschitz continuous gradient). A differentiable convex function f is said to have an L-Lipschitz continuous gradient, if there exists a constant $L > 0$, such that*

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \forall x, y.$$

If f has an L-Lipschitz continuous gradient, then it holds that

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|^2, \forall x, y.$$

Definition 6.2. *(m-strongly convex). A differentiable convex function f is said to be m-strongly convex if there exists a constant $m > 0$, such that*

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{m}{2}\|y - x\|^2, \forall x, y.$$

If f is m-strongly convex and has an L-Lipschitz continuous gradient, then it is obvious that $m \leq L$.

Assumption 6.1. *For every normal agent k, the risk function $r_k(\cdot)$ is m-*strongly convex *and has L-*Lipschitz continuous gradient*.*

Assumption 6.2. *For every normal agent k, the stochastic gradient $\nabla \ell(\theta_{k,i}; \xi_k^i)$ is an unbiased estimate of $\nabla r_k(\theta_{k,i})$, i.e., $\mathbb{E}[\nabla \ell(\theta_{k,i}; \xi_k^i)] = \nabla r_k(\theta_{k,i})$, for all $i \in \mathbb{N}$.*

Assumption 6.3. *For every normal agent k, there exists $c_k \geq 1$, such that for all $i \in \mathbb{N}$, $\mathbb{E}[\|\nabla \ell(\theta_{k,i}; \xi_k^i)\|_2^2] \leq \sigma_k^2 + c_k \|\nabla r_k(\theta_{k,i})\|_2^2$.*

Given these assumptions, the convergence of a normal agent running SGD is guaranteed with appropriate step size [156]. Using the proposed rule (6.12), under these assumptions, we further guarantee the convergence of the normal agents running the ATC diffusion algorithm in Theorem 6.1.

Theorem 6.1. *A normal agent k which runs the ATC diffusion algorithm using the loss-based weights* (6.12) *converges towards θ_k^* with $\lim_{i\to\infty} \mathbb{E}\left[r_k\left(\theta_{k,i}\right) - r_k(\theta_k^*)\right] \leq \frac{\mu_k L \sigma_k^2}{2m}$, for fixed stepsize $\mu_k \in (0, \frac{1}{Lc_k}]$, in the presence of an arbitrary number of Byzantine neighbors. Further, it holds that*

$$\mathbb{E}[r_k(\theta_{k,i}) - r_k(\theta_k^*)] \leq \mathbb{E}[r_k(\hat{\theta}_{k,i}) - r_k(\theta_k^*)], \forall k \in \mathcal{N}^+, i \in \mathbb{N}.$$

Proof. Let $\mathbb{E}[\cdot]$ denote the expected value taken with respect to the joint distribution of all random variables ξ_k and ξ_l for $l \in \mathcal{N}_k^{\leq}$, i.e.

$$\mathbb{E}\left[\cdot\right] = \mathbb{E}_{\xi_k} \mathbb{E}_{\{\xi_l | l \in \mathcal{N}_k^{\leq}\}}\left[\cdot\right].$$

Similar to the proof for Lemma 6.1, using $\mathcal{N}_k^{\leq}$ in the place of $\mathcal{N}_k$, with rule (6.12), we obtain

$$\mathbb{E}\left[r_k(\theta_{k,i}) - r_k(\theta_k^*)\right] \leq \frac{1}{|\mathcal{N}_k^{\leq}|} \sum_{l \in \mathcal{N}_k} \mathbb{E}\left[r_k\left(\hat{\theta}_{l,i}\right) - r_k(\theta_k^*)\right]. \tag{6.13}$$

For every $l \in \mathcal{N}_k^{\leq}$, we have $r_k(\hat{\theta}_{l,i}) \leq r_k(\hat{\theta}_{k,i})$ and hence

$$\frac{1}{|\mathcal{N}_k^{\leq}|} \sum_{l \in \mathcal{N}_k} \mathbb{E}\left[\left(r_k(\hat{\theta}_{l,i}) - r_k(\theta_k^*)\right)\right] \leq \mathbb{E}\left[\left(r_k(\hat{\theta}_{k,i}) - r_k(\theta_k^*)\right)\right].$$

Put it back to (6.13), we obtain

$$\mathbb{E}\left[r_k\left(\theta_{k,i}\right) - r_k(\theta_k^*)\right] \leq \frac{1}{|\mathcal{N}_k^{\leq}|} \sum_{l \in \mathcal{N}_k^{\leq}} \mathbb{E}\left[r_k\left(\hat{\theta}_{l,i}\right) - r_k(\theta_k^*)\right] \leq \mathbb{E}\left[r_k\left(\hat{\theta}_{k,i}\right) - r_k(\theta_k^*)\right], \forall k \in \mathcal{N}^+, i \in \mathbb{N}, \tag{6.14}$$

which yields (6.5).

We next prove the convergence of the algorithm with the proposed weight assignment rule. Given Assumptions 1-3, we obtain from [156] that using constant step size $\mu_k \in (0, \frac{1}{Lc_k}]$, it holds that

$$\mathbb{E}\left[r_k\left(\hat{\theta}_{k,i}\right) - r_k(\theta_k^*)\right] - \frac{\mu_k L \sigma_k^2}{2m} \leq (1 - \mu_k m)\left(\mathbb{E}\left[r_k\left(\theta_{k,i-1}\right) - r_k(\theta_k^*)\right] - \frac{\mu_k L \sigma_k^2}{2m}\right).$$

Combined with (6.14), we obtain

$$\mathbb{E}\left[r_k\left(\theta_{k,i}\right) - r_k(\theta_k^*)\right] - \frac{\mu_k L \sigma_k^2}{2m} \leq (1 - \mu_k m)\left(\mathbb{E}\left[r_k\left(\theta_{k,i-1}\right) - r_k(\theta_k^*)\right] - \frac{\mu_k L \sigma_k^2}{2m}\right). \tag{6.15}$$

Given $\mu_k \in (0, \frac{1}{Lc_k}]$, with $c_k \geq 1$, $m \leq L$, it holds that $(1 - \mu_k m) \in [0, 1)$. Applying (6.15) repeatedly through iteration $i \in \mathbb{N}$, we obtain

$$\begin{aligned}\mathbb{E}\left[r_k\left(\theta_{k,i}\right) - r_k(\theta_k^*)\right] &\leq \frac{\mu_k L \sigma_k^2}{2m} + (1 - \mu_k m)^i \left(r_k\left(\theta_{k,0}\right) - r_k(\theta_k^*) - \frac{\mu_k L \sigma_k^2}{2m}\right) \\ &\xrightarrow{i \to \infty} \frac{\mu_k L \sigma_k^2}{2m}.\end{aligned}$$

This means $\theta_{k,i}$ converges towards θ_k^* with the expected regret bounded by $\frac{\mu_k L \sigma_k^2}{2m}$. □

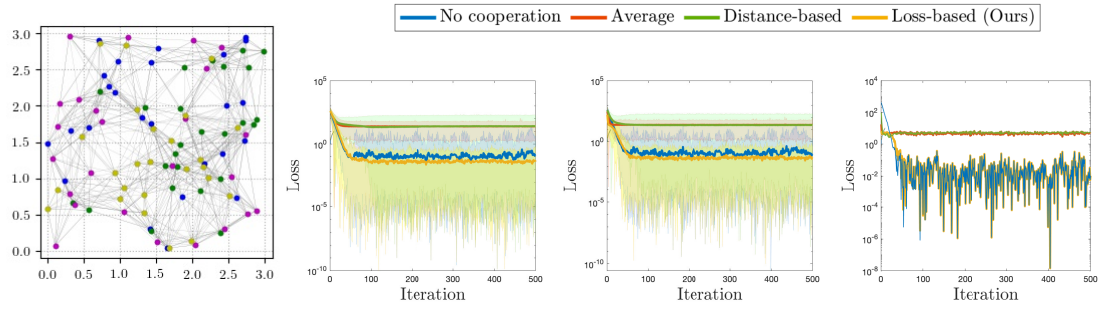
Theorem 6.1 indicates that cooperation using weights in (6.12) is always at least as good as the non-cooperative case, as measured by the expected regret, which satisfies the conditions lsited in Section 6.4. Note that even when all the neighbors of a normal agent are Byzantine, one can still guarantee that the agent's learning performance as a result of cooperation with neighbors using (6.12) will be same as the non-cooperative case.

Discussion. We assume convex models to carry out the analysis, which is typical in the literature. However, the intuition behind the approach is — *to measure the relatedness of a neighbor to itself, a normal agent evaluates the loss of the neighbor using the neighbor's model parameters and its own data, and cuts down*

the cooperation if this loss is larger than the agent's own loss — and the same idea should also apply to non-convex models. In the next section, we also evaluate our methods on non-convex models, such as CNNs, which generates experimental results similar to those produced by convex models.

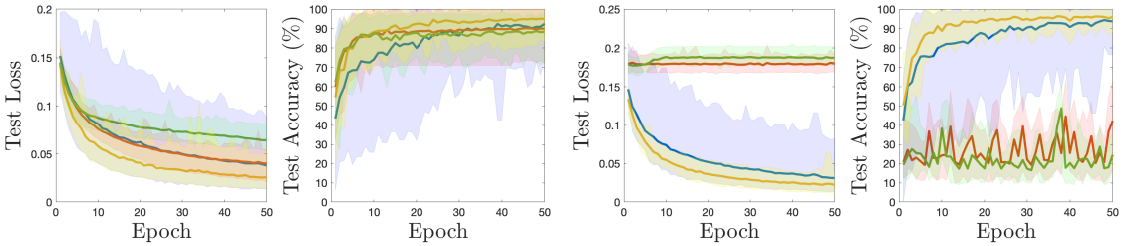
6.7 Evaluation

In this section, we evaluate the resilience of the proposed online weight adjustment rule (6.12) with the smoothing method discussed in Section 6.5, and compare it with the non-cooperative case, the average weights ($a_{lk} = \frac{1}{|\mathcal{N}_k|}$), and the quadratic distance-based weights (6.4) (with $\tilde{\theta}_k^* = \theta_{k,i-1}$ and use the same smoothing method $\phi_{lk}^i = (1-\nu_k)\phi_{lk}^{i-1} + \nu_k \|\tilde{\theta}_k^* - \hat{\theta}_{l,i}\|^2$ in the place of $\|\tilde{\theta}_k^* - \hat{\theta}_{l,i}\|^2$, with the same forgetting factor ν_k used for (6.12)). We use three distributed MTL case studies, including the regression and classification problems, with and without the presence of Byzantine agents. Although the convergence analysis in Section 6.6 is based on convex models and SGD, we show empirically that the weight assignment rule (6.12) performs well for non-convex models, such as CNNs and mini-batch gradient descent. Our code is available at https://github.com/JianiLi/resilientDistributedMTL.



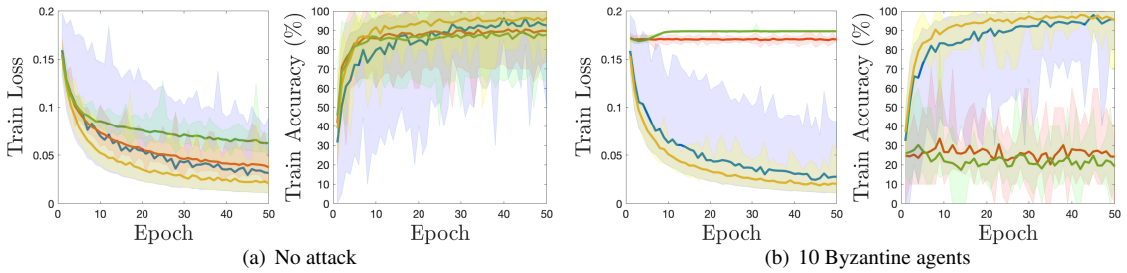
(a) Network topology (b) No attack (c) 20 Byzantine agents (d) 99 Byzantine agents

Figure 6.1: Target localization: network topology and loss of streaming data for normal agents.



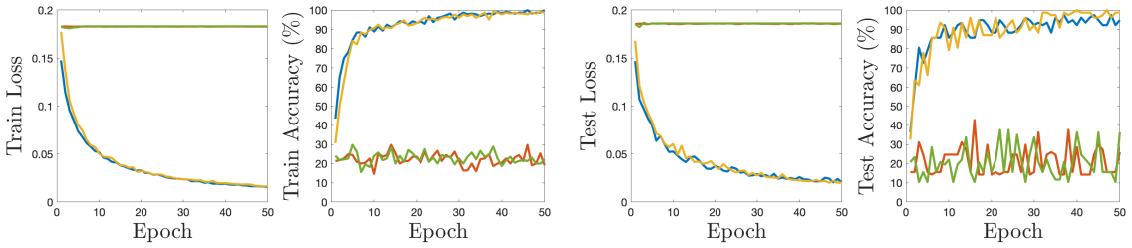
(a) No attack (b) 10 Byzantine agents

Figure 6.2: Human action recognition: average testing loss and accuracy for normal agents.



(a) No attack (b) 10 Byzantine agents

Figure 6.3: Human action recognition: average training loss and accuracy for normal agents.



(a) Training (b) Testing

Figure 6.4: Human action recognition: average training/testing loss and accuracy for normal agents with 29 Byzantine agents.

6.7.1 Datasets and Simulation Setups

- **Target localization**: Target localization is a widely-studied linear regression problem [185]. The task is to estimate the location of the target by minimizing the squared error loss of noisy streaming sensor data. We consider a network of 100 agents with four targets as shown in Figure 6.1(a). Agents in the same color share the same target, however, they do not know this group information beforehand. The four target locations in $\mathbb{R}^2$ are: $(10.84, 10.76)$, $(20.42, 20.26)$, $(20.51, 10.40)$, $(10.78, 20.30)$. Agents' locations are indicated in Figure 6.1(a). An edge between two agents means they are neighbors. At each iteration, every agent k has a noisy observation (streaming data) of the distance $\boldsymbol{d}_k(i)$ and the unit direction vector $\boldsymbol{u}_{k,i}$ pointing from x_k to its target based on built-in sensors. Let $\theta_k \in \mathbb{R}^2$ denote the estimation of the target location for agent k, then the loss is computed as $\ell_k(\theta_{k,i}; \xi_k^i) = \|\boldsymbol{d}_k(i) - (\theta_k - x_k)^\top \boldsymbol{u}_{k,i}\|^2$, and the agent estimates θ_k using the SGD algorithm as well as the ATC diffusion algorithm with different weight assignment rules. The distance measurement data has noise variance $\sigma_{d,k}^2 \in [0.1, 0.2]$, and the unit direction vector has additive white Guassian noise with diagnonal covariance matrices $R_{u,k} = \sigma_{u,k}^2 I_2$, with $\sigma_{u,k}^2 \in [0.01, 0.1]$ for different k. We tune the step-sizes and forgetting factors from the interval $(0, 1)$ and find the best empirical performance by setting them to be $\mu_k = 0.1$ and $\nu_k = 0.1$ for every normal agent k. φ_{lk}^{-1} and ϕ_{lk}^{-1} are initialized to be zero for all $l \in \mathcal{N}_k$. Byzantine agents are designed to continuously send random values for each dimension from the interval $[15, 16]$ at each iteration.

- **Human activity recognition**[5]: Mobile phone sensor data (accelerometer and gyroscope) is collected from 30 individuals performing one of six activities: {walking, walking-upstairs, walking-downstairs, sitting, standing, lying-down}. The goal is to predict the activities performed using 561-length feature vectors for each instance generated by the processed sensor signals [49]. We model each individual as a separate task and use a complete graph to model the network topology. We use linear model as the prediction function with cross-entropy-loss. We randomly split the data into 75% training and 25% testing for each agent. During training, ten of the thirty agents are randomly selected to have access to much less data (about $\frac{1}{10}th$) than the other agents at each epoch. This is to model the realistic scenario in which some of the agents may have less data samples and they may learn slowly than others. We use mini-batch gradient descent with batch size of 10. We tune the step-sizes and forgetting factors from the interval $(0, 1)$ and find the best empirical performance by setting them to be $\mu_k = 0.01$ and $\nu_k = 0.05$ for every normal agent k. φ_{lk}^{-1} and ϕ_{lk}^{-1} are initialized to be zero for all $l \in \mathcal{N}_k$. Byzantine agents are designed to send a model with very small noisy elements for each dimension from the interval $[0, 0.1]$ at each iteration.

- **Digit classification**: We consider a network of ten agents performing digit classification. Five of the ten agents have access to the MNIST dataset[6] [186] (group 1) and the other five have access to the synthetic dataset[7] (group 2) that is composed by generated images of digits embedded on random backgrounds [187]. All the images are preprocessed to be 28×28 grayscale images. We model each agent as a separate task and use a complete graph to model the network topology. An agent does not know which of its neighbors are performing the same task as the agent itself. We use a CNN model of the same architecture for each agent and cross-entropy-loss. The preprocessed examples of the two datasets are given in Figure 6.5. The details of the CNN architecture is given in Table 6.1. For each group, we consider that agents have access to uneven sizes of training data. Specifically, for each agent, we randomly feed $200 - 2000$ training data and 400 testing data from the corresponding dataset for each epoch. We use mini-batch gradient descent with batch size of 64. We tune the step-sizes and forgetting factors from the interval $(0, 1)$ and find the best empirical performance by setting them to be $\mu_k = 0.001$ and $\nu_k = 0.05$ for every normal agent. φ_{lk}^{-1} and ϕ_{lk}^{-1} are initialized to be zero for all $l \in \mathcal{N}_k$. Byzantine agents are designed to send a model with very small noisy elements for each dimension from the interval $[0, 0.1]$ at each iteration.

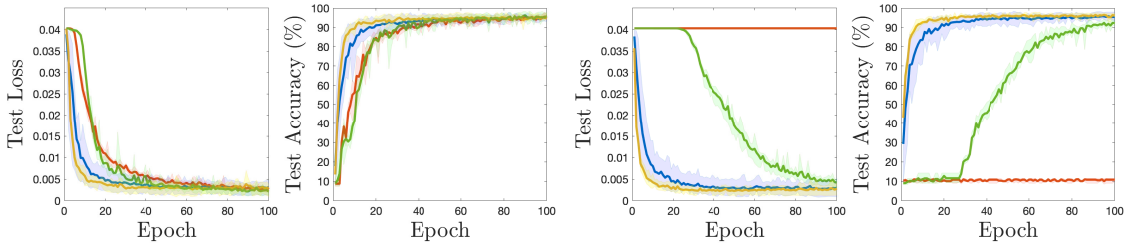
[5] https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones
[6] http://yann.lecun.com/exdb/mnist
[7] https://www.kaggle.com/prasunroy/synthetic-digits

Table 6.1: CNN architecture of digit classification

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 28, 28]	320
ReLU-2	[-1, 32, 28, 28]	0
MaxPool2d-3	[-1, 32, 14, 14]	0
Conv2d-4	[-1, 64, 14, 14]	18,496
ReLU-5	[-1, 64, 14, 14]	0
MaxPool2d-6	[-1, 64, 7, 7]	0
Conv2d-7	[-1, 64, 7, 7]	36,928
ReLU-8	[-1, 64, 7, 7]	0
MaxPool2d-9	[-1, 64, 3, 3]	0
Linear-10	[-1, 128]	73,856
ReLU-11	[-1, 128]	0
Linear-12	[-1, 10]	1,290

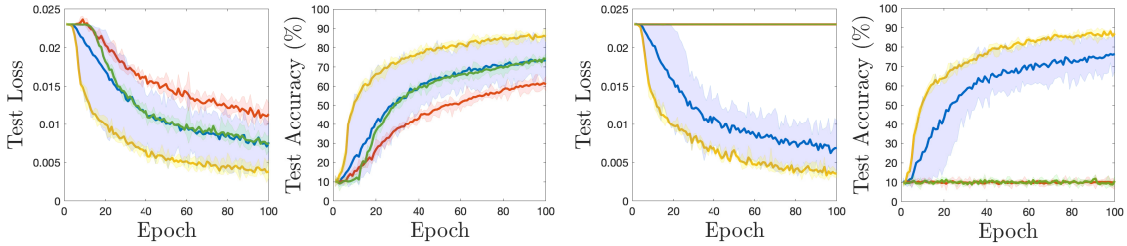
(a) MNIST (b) Synthetic digits

Figure 6.5: Examples of the digit classification dataset



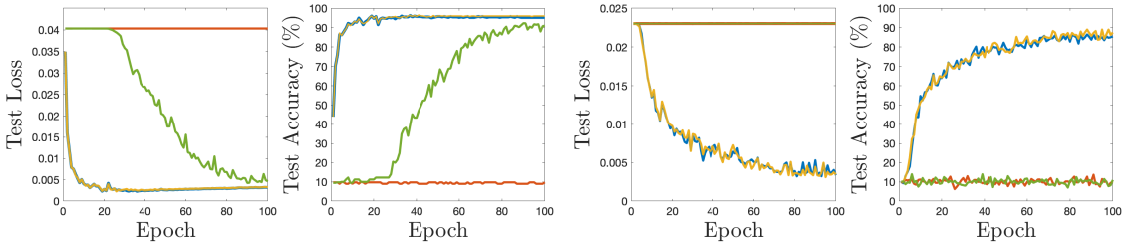
(a) No attack (b) 2 Byzantine agents

Figure 6.6: Digit classification: average testing loss and accuracy for normal agents in group 1.



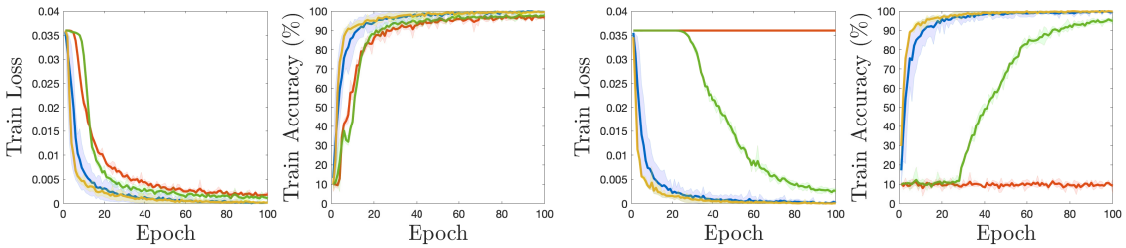
(a) No attack (b) 2 Byzantine agents

Figure 6.7: Digit classification: average testing loss and accuracy for normal agents in group 2.



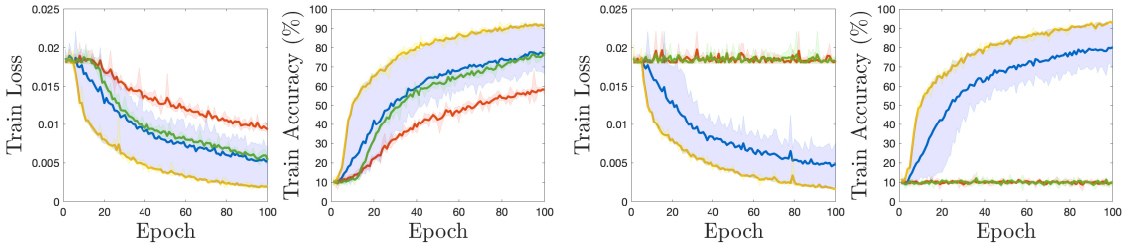
(a) Group 1 (b) Group 2

Figure 6.8: Digit classification: average testing loss and accuracy for normal agents, with 8 Byzantine agents (four for each group).



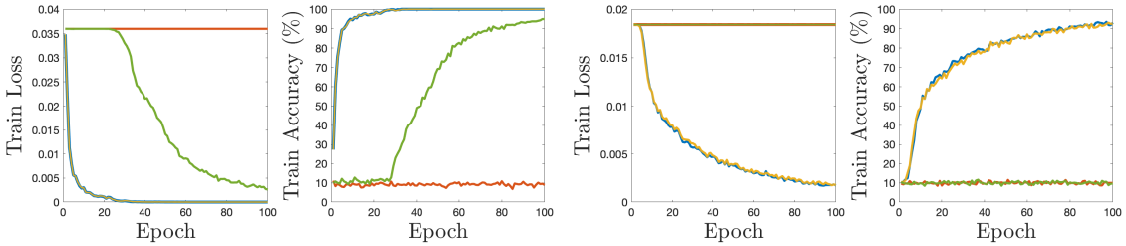
(a) No attack (b) 2 Byzantine agents

Figure 6.9: Digit classification: average training loss and accuracy for normal agents in group 1.



(a) No attack (b) 2 Byzantine agents

Figure 6.10: Digit classification: average training loss and accuracy for normal agents in group 2.



(a) Group 1 (b) Group 2

Figure 6.11: Digit classification: average training loss and accuracy for normal agents, with 8 Byzantine agents (four for each group).

6.7.2 Results

We plot the mean and range of the average loss of every normal agent for the target localization problem in Figure 6.1(b)–d. Similarly, we plot the mean and range of the average training/testing loss and classification accuracy of every normal agent for human action recognition in Figure 6.2 - Figure 6.4, and for digit classification in Figure 6.6 - Figure 6.11. At each iteration, Byzantine agents send random values (for each dimension) from the interval $[15, 16]$ for target localization, and $[0, 0.1]$ for the other two case studies.

In all of the examples, we find that the loss-based weight assignment rule (6.12) outperforms all the other rules and the non-cooperative case, with respect to the mean and range of the average loss and accuracy with and without the presence of Byzantine agents. Hence, our simulations validate the results indicated by (6.5) and imply that the loss-based weights (6.12) have accurately learned the relationship among agents. Moreover, normal agents having a large regret in their estimation benefit from cooperating with other agents having a small regret. We also consider the extreme case in which there is only one normal agent in the network, and all the other agents are Byzantine. In such a case, the loss-based weight assignment rule (6.12) has the same performance as the non-cooperative case, thus, showing that it is resilient to an arbitrary number of Byzantine agents.

Comparing the results between groups 1 and 2 for digit classification reveals that cooperation is most beneficial when there is a substantial divergence in agents' learning performances. Given limited training data, agents in group 1 are able to build refined models. It is harder for agents receiving less training data in group 2 to achieve a high learning performance as the synthetic digit classification is a more challenging task than the MNIST digit classification. Using the weight assignment rule (6.12), those agents receiving less data (and therefore, struggling to learn a good model), are able to benefit from the cooperation with the neighbors having learned a refined model. At the same time, agents exhibiting high learning performance will not be negatively affected by such cooperation.

6.8 Conclusion

In this chapter, we propose an efficient online weight adjustment rule for learning the similarities among agents in distributed multi-task networks with an arbitrary number of Byzantine agents. We argue that a widely used approach of measuring the similarities based on the distance between two agents' model parameters is vulnerable to Byzantine attacks. To cope with such vulnerabilities, we propose to measure similarities based on the (accumulated) loss using an agent's data and its neighbors' models. A small loss indicates a large similarity between the agents. To eliminate the influence of Byzantine agents, a normal agent filters out the information from neighbors whose losses are larger than the agent's own loss. With filtering, aggregation

using the loss-based weight adjustment rule results in an improved expected regret than the non-cooperative case and guarantees that each normal agent converges resiliently towards the global minimum. The experiment results validate the effectiveness of our approach.

Broader Impact

The problem of Byzantine resilient aggregation of distributed machine learning models has been actively studied in recent years; however, the issue of Byzantine resilient distributed learning in multi-task networks has received much less attention. It is a general intuition that MTL is robust and resilient to cyber-attacks since it can identify attackers by measuring similarities between neighbors. In this chapter, we have shown that some commonly used similarity measures are not resilient against certain attacks. With an increase in data heterogeneity, we hope this chapter could highlight the security and privacy concerns in designing distributed MTL frameworks.

Chapter 7

Distributed Clustering for Cooperative Multi-Task Learning Networks

Distributed learning enables collaborative training of machine learning models across multiple agents by exchanging model parameters without sharing local data. Each agent generates data from distinct but related distributions and multi-task learning can be effectively used to model related tasks. This chapter focuses on clustered multi-task learning where agents are partitioned into clusters with distinct objectives, and agents in the same cluster share the same objective. The structure of such clusters is unknown apriori. Cooperation with the agents in the same cluster is beneficial and improves the overall learning performance. However, indiscriminate cooperation of agents with different objectives leads to undesired outcome. Accurately capturing the clustering structure benefits the cooperation and offers many practical benefits, for instance, it helps advertising companies better target their ads. In this chapter, we propose an adaptive clustering method that allows distributed agents to learn the most appropriate neighbors to collaborate with and form clusters. We prove convergence of every agent towards its objective and analyze the network learning performance using the proposed clustering method. Further, to determine how one should aggregate the neighbors' model parameters after the clustering step, we present the optimal combination weights that optimize the network's learning performance. The theoretical analysis is well-validated by the evaluation results using target localization and digit classification, showing that the proposed clustering method outperforms existing methods as well as the case where agents do not cooperate with each other.

7.1 Introduction

Distributed learning has attracted increasing attention due to the growth of machine learning (ML) applications in distributed devices within multi-agent networks, such as mobile phones, wearable devices, and smart homes [48–51]. In such networks, multiple agents can operate in a distributed and cooperative manner to achieve a learning task. For example, consider learning the behavior of users in a cellular network based on data generated using various mobile applications. Each user may generate data that follows a distinct distribution and it is common to learn separate models for each user. However, people may exhibit similar behaviors and similarities among models commonly exist [15]. In this case, cooperation among agents could be leveraged to promote the learning performance over the network.

Given data privacy concerns, cooperation among agents in a network relies typically on exchanging model

parameters instead of data. In a distributed learning network, an agent communicates model parameters with its local neighbors and also updates these parameters by incorporating the neighbors' information [16]. It has been demonstrated that such cooperation enables improved learning performance over the network [73]. Compared to federated learning [188], distributed learning with fully-decentralized networked agents does not require a central server, thus addressing single-point-of-failure and scalability issues.

It is natural to model a distributed learning network using clustered multi-task learning in which agents with similar interests are grouped in the same cluster, and different clusters represent distinct interests [185]. We note that if there are no similarities among the agents, then the clustered multi-task network reduces to a non-cooperative network. At the same time, if all agents in the network are similar, then the network reduces to a single-task cooperative network. Agents should not be aware of the clustered structure beforehand, and the clustering scheme should adapt to changes and also accommodate any new agents. Clustered multi-task learning offers significant practical benefits. For instance, in a mobile application network, people with similar age or profession may exhibit similar preferences for particular content. Further, such preferences may deviate from the preferences of people with different age or profession. Advertising companies, in particular, use such strategies to better target and manage their advertisements. Adaptive clustering allows agents to perform local optimization tasks while learning from neighbors with similar objectives. As a result, an agent is allocated to an appropriate cluster without knowing the clustering structure a priori [60, 97, 103, 104, 189]. Agents can then cooperate with others in the same cluster, thus, making cooperation more beneficial and meaningful. However, as we show in Section 7.4, existing methods typically rely on comparing the Euclidean distance between model parameters to measure their similarities and could result in two agents from different clusters starting cooperation during learning. In particular, such cooperation may continue and agents could be driven to converge to a wrong cluster. Our evaluation in Section 7.5 also shows that such an approach could fail in learning the correct clustering structure.

To address such problems, we propose an *adaptive clustering method by comparing the losses of two agents*. The main idea is that *agents sharing similar objectives have similar underlying data distributions, and hence their models should fit each other's data distributions.* Moreover, to make the cooperation lead to a better learning performance (smaller loss), it is appropriate for an agent to cooperate only with those neighbors who incur a smaller loss than the agent itself. To measure similarity between an agent and its neighbor, we propose that an agent computes the loss by fitting its neighbor's model to its data and then compares this loss to the one obtained by fitting the data to its own model. The agent cooperates with a neighbor only if the loss with the neighbor's model is no greater than the agent's own loss. In doing so, it is guaranteed that the loss is reduced in expectation by the cooperation, and agents converge to the objective of the cluster.

The main contributions of the chapter are:

- We propose an adaptive clustering method in distributed multi-task learning networks that allows agents to perform the optimization task while simultaneously learn which neighbors are suitable for cooperation.
- We analyze the convergence and learning performance for the proposed method.
- We propose the optimal combination weights for aggregating the neighbors' model parameters after recognizing which neighbors to cooperate with using the proposed clustering method that optimize the network learning performance.
- We evaluate the proposed method for both linear regression and classification problems and compare the results with existing distributed clustering methods. The evaluation results show that the proposed method significantly improves the learning performance compared to the non-cooperative approach by correctly estimating the clustering structure. In contrast, other methods fail to achieve the clustering information and exhibit inferior learning performance.

7.2 Related Work

Multi-Task Learning. Multi-task learning (MTL) deals with the problem of solving multiple related optimization tasks simultaneously to improve the overall performance of the models learned by each task with other auxiliary tasks [85]. There is a large body of related work in the area of MTL with different variations. One area of work is related to deep learning, with the aim of learning multiple objectives from a shared representation by sharing layers and splitting architecture in the deep neural networks [87–90]. Such a framework usually assumes that data sets are collected from all the tasks in a central server and uses a single model to do the learning. An example is to learn the depth and semantics from RGB images simultaneously [86]. In contrast, in this chapter, we consider distributed multi-task learning where networked agents maintain separate data sets and models without a central server [7, 182]. While the first MTL framework is widely used in deep learning, e.g., computer vision and natural language processing, the second framework considered in this chapter is naturally suited for distributed learning in multi-agent systems, such as mobile phones, autonomous vehicles, and smart cities [48–51].

Distributed Clustering over Networks. Clustering is a well-known unsupervised learning technique for grouping a set of data points [102]. In contrast, this chapter deals with the distributed clustering problem of a set of networked agents running individual optimization tasks [60, 97, 103, 104]. In such methods, agents perform local tasks while simultaneously learning which neighbors they should cooperate with by measuring their relatedness. Compared to traditional clustering, distributed clustering is more challenging since any

clustering error could lead an agent towards an undesired model. Measuring the Euclidean distance between the model parameters of agents is a principle method used in distributed clustering. For example, in [103], if the Euclidean distance is less than a pre-defined threshold, then the two agents will be clustered into the same group. Similarly, in [104], an accumulated Euclidean distance within a time-based sliding window is used. Adaptive weights based on Euclidean distance are used in [97] and [60] for distributed clustering. As we discuss in Section 7.4, measuring similarities relying on comparing the Euclidean distance between two model parameters could lead agents to converge to a wrong cluster, as illustrated in our evaluation (Section 7.5) as well. In contrast, the proposed method can capture the accurate clustering information and guarantees that agents cooperate only with the neighbors sharing the same task. This, in turn, ensures that agents converge to their global minimizers resulting in an improved overall learning performance than without cooperation.

7.3 Clustered Multi-Task Network

We consider a network of N agents modeled by an *undirected graph* $G = (V, E)$, where V represents agents and E represents interactions between agents. A bi-directional edge $(l, k) \in E$ means that agents k and l can exchange information with each other. Note that $(k, k) \in E, \forall k \in V$. The *neighborhood* of k is the set $\mathcal{N}_k = \{l \in V | (l, k) \in E\}$. Every agent is associated with a loss function $r_k(\theta) : \mathbb{R}^{d\times 1} \to \mathbb{R}$. It is assumed that each $r_k(\theta)$ is strongly-convex and is minimized at the unique point θ_k^*. Agents can then be categorized into $Q \in [1, N]$ mutually-exclusive clusters $\{\mathcal{C}_q; q = 1, 2, \ldots, Q\}$, such that agents in the same cluster share the same minimizer for their individual loss functions. Denote the unique minimizer for cluster $\mathcal{C}_q$ as θ_q^o, then $\theta_k^* = \theta_q^o$ for all $k \in \mathcal{C}_q$. We also assume that the underlying network topology is connected and clusters are inter-connected so that agents may have neighbors from different clusters. Note that agents do not know which of the neighbors belong to the same cluster as themselves. Agents are interested in solving the following optimization problem:

$$\min_{\{\theta_q\}_{q=1}^Q} \sum_{q=1}^{Q} \sum_{k\in\mathcal{C}_q} r_k(\theta_q). \tag{7.1}$$

To solve (7.1) in a distributed and cooperative way, we use the *adapt-then-combine* (ATC) diffusion algorithm [182] with a clustering step in-between, which takes the following iterative steps for an agent k at iteration i:

$$\hat{\theta}_{k,i} = \theta_{k,i-1} - \mu_k \widehat{\nabla r_k}(\theta_{k,i-1}), \qquad \text{(adaptation)} \tag{7.2}$$

$$\text{Obtain } \mathcal{N}_{k,i}^+ \text{ by clustering,} \qquad \text{(clustering)} \tag{7.3}$$

$$\theta_{k,i} = \sum_{l\in\mathcal{N}_{k,i}^+} a_{lk}(i)\hat{\theta}_{l,i}, \qquad \text{(combination)} \tag{7.4}$$

where μ_k is the step-size, $\mathcal{N}_{k,i}^+$ is the set of neighbors belonging to the same cluster as agent k at iteration i, $\widehat{\nabla r_k}(\theta_{k,i-1})$ is the stochastic gradient of the loss function $r_k(\cdot)$ at $\theta_{k,i-1}$, and $a_{lk}(i)$ denotes the weight assigned by agent k to l at iteration i that satisfies the following constraints:

$$\sum_{l\in\mathcal{N}_{k,i}^+} a_{lk}(i) = 1, a_{lk}(i) \geq 0, a_{lk}(i) = 0 \text{ if } l \notin \mathcal{N}_{k,i}^+. \tag{7.5}$$

The ATC algorithm indicates that at each iteration i, agent k minimizes the individual loss function using stochastic gradient descent (SGD) given local data followed by a combination step that aggregates the model parameters of the neighbors in the same cluster according to the weights assigned to them.

The goal of this chapter is to solve the optimization problem (7.1) in a distributed and cooperative way using the ATC diffusion algorithm in (7.2)-(7.4) by designing the clustering and optimal combination weights (7.5) that agents use for cooperation. Cooperation among agents can improve learning if they share common objectives. However, when agents pursue different objectives, indiscriminate cooperation leads to undesired outcomes [105]. Therefore, it is important for the agents to use an accurate clustering method that allows them to learn which neighbors are sharing similar objectives. By doing so, agents only cooperate with the neighbors in the same cluster and stop cooperating with those from different clusters, thus ensuring that the cooperation is beneficial.

7.4 Adaptive Clustering

In this section, we first propose a distributed clustering method that allows agents to learn which neighbors are in the same cluster. Next, we analyze the convergence and learning performance using the proposed clustering method. Finally, we propose the optimal combination weights to aggregate the model parameters of the neighbors belonging to the same cluster that solves (7.1).

7.4.1 Clustering Hypothesis

Distributed clustering methods are based on measuring similarities among agents. Existing methods rely on comparing the Euclidean distance between two agents' model parameters to determine whether they belong to the same cluster [60, 97, 103, 104]. For example, in [103], the following hypothesis test is used for agent k to determine whether a neighbor l belongs to the same cluster:

$$\|\hat{\theta}_{l,i} - \hat{\theta}_{k,i}\|^2 \underset{\mathbb{H}_1}{\overset{\mathbb{H}_0}{\lessgtr}} d_{k,l}^2, \tag{7.6}$$

where $\mathbb{H}_0$ denotes the hypothesis that $l \in \mathcal{N}_{k,i}^+$ and $\mathbb{H}_1$ denotes the hypothesis that $l \notin \mathcal{N}_{k,i}^+$, and $d_{k,l} > 0$ is a predefined threshold.

We find that methods based on Euclidean distance between model parameters may lead agents to cooperate with neighbors from another cluster which could prevent agents from converging. Consider the example shown in Figure 7.1. Assume that k has only one neighbor l, and at iteration i, agent k identifies l to be in the same cluster as $\|\hat{\theta}_{l,i} - \hat{\theta}_{k,i}\|^2 < d_{k,l}^2$. If $\|\hat{\theta}_{l,i} - \hat{\theta}_k^*\|^2 > \|\hat{\theta}_{k,i} - \theta_k^*\|^2$, then $\theta_{k,i}$ as a combination of $\hat{\theta}_{k,i}$ and $\hat{\theta}_{l,i}$ (given (7.4)) will move away from θ_k^* rendering $r_k(\theta_{k,i}) > r_k(\hat{\theta}_{k,i})$. If l is from another cluster that moves away from θ_k^* yet keeps falling into hypothesis $\mathbb{H}_0$ for agent k, then k will continue to cooperate with l and will fail to converge to θ_k^*.

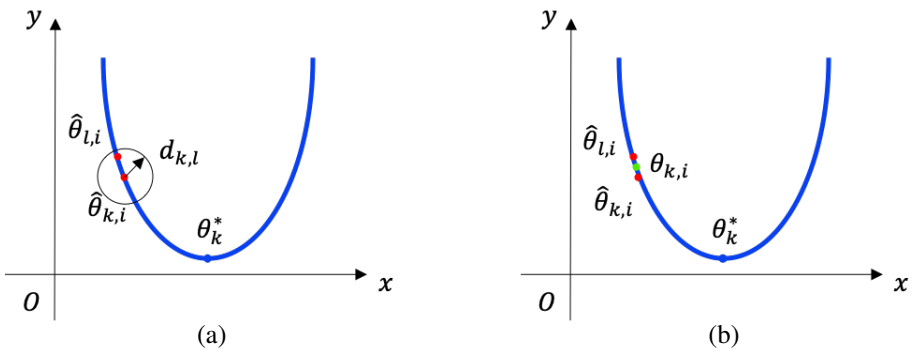


Figure 7.1: Example of an undesired outcome due to clustering using hypothesis testing (7.6): (a) k identifies l to be in the same cluster; (b) $\theta_{k,i}$ as a combination of $\hat{\theta}_{k,i}$ and $\hat{\theta}_{l,i}$ moves away from $\theta_{k,i}^*$ rendering $r_k(\theta_{k,i}) > r_k(\hat{\theta}_{k,i})$.

To ensure that agents converge to their global minimizers using clustering-based cooperation, we propose an alternative hypothesis test as follows:

$$\widehat{r_k}(\hat{\theta}_{l,i}) - \widehat{r_k}(\hat{\theta}_{k,i}) \underset{\mathbb{H}_1}{\overset{\mathbb{H}_0}{\lessgtr}} 0, \tag{7.7}$$

where $\mathbb{H}_0$ denotes the hypothesis that $l \in \mathcal{N}_{k,i}^+$ and $\mathbb{H}_1$ denotes the hypothesis that $l \notin \mathcal{N}_{k,i}^+$, and $\widehat{r_k}(\cdot)$ is the approximation of $r_k(\cdot)$ with $\mathbb{E}\left[\widehat{r_k}(\theta)\right] = \mathbb{E}\left[r_k(\theta)\right]$. The approximation $\widehat{r_k}(\cdot)$ can be performed using stacked data received in the previous iterations. Alternatively, if we use (mini-) batch gradient descent in the adaptation step, then the batch loss can be used as $\widehat{r_k}(\cdot)$.

The hypothesis test (7.7) indicates that the clustering is based on measuring the loss by fitting the neighbor's model to an agent's data distribution and comparing the loss with its own loss. Therefore, we have

$$\mathbb{E}\left[\widehat{r_k}(\theta_{k,i})\right] = \mathbb{E}\left[\widehat{r_k}\left(\sum_{l \in \mathcal{N}_{k,i}^+} a_{lk}(i)\hat{\theta}_{l,i}\right)\right] \leq \mathbb{E}\left[\sum_{l \in \mathcal{N}_{k,i}^+} a_{lk}(i)\widehat{r_k}(\hat{\theta}_{l,i})\right] \leq \mathbb{E}\left[\widehat{r_k}(\hat{\theta}_{k,i})\right], \tag{7.8}$$

where the first inequality follows by Jensen's inequality assuming $\widehat{r_k}(\cdot)$ is convex, and the second inequality follows by the proposed clustering hypothesis test (7.7). Since we assume that $\mathbb{E}\left[\widehat{r_k}(\theta)\right] = \mathbb{E}\left[r_k(\theta)\right]$, it follows from (7.8) that

$$\mathbb{E}\left[r_k(\theta_{k,i})\right] \leq \mathbb{E}\left[r_k(\hat{\theta}_{k,i})\right]. \tag{7.9}$$

Thus, clustering using the hypothesis test (7.7) results in a reduced loss in expectation. Later in Theorem 7.1, we will show that this results in the convergence of every agent towards their objectives.

7.4.2 Convergence and Learning Performance

We first prove the convergence of every agent k towards its objective θ_k^* using the clustering hypothesis test (7.7), which ensures that agent k converges to the true cluster. Next, we analyze the learning performance over the network using (7.7). Below, we list the necessary assumptions for the loss functions and gradient noise used in our results that are standard in the literature [73, 103, 156].

Assumption 7.1. (Loss functions)

(a) Different clusters have distinct minimizers $\theta_q^o \neq \theta_r^o$ if $q \neq r$. Further, the difference between every two minimizers of two distinct clusters are sufficiently large such that $r_k(\theta_q^o) < r_k(\theta_r^o)$ if $k \in \mathcal{C}_q$ and $k \notin \mathcal{C}_r$.

(b) The sequence $\{\theta_{k,i}\}$ for every agent k is contained in an open set over which $r_k(\cdot)$ is always upper-bounded by a scalar r_{inf}.

(c) Each individual loss function $r_k(\theta)$ is strongly convex, twice-differentiable, and has bounded Hessian matrix function, which also implies that it has a Lipschitz continuous gradient, i.e.,

$$r_k(\theta_2) \geq r_k(\theta_1) + \langle \nabla r_k(\theta_1), \theta_2 - \theta_1 \rangle + \frac{m}{2}\|\theta_2 - \theta_1\|^2, \forall \theta_1, \theta_2, \text{ for some } m > 0.$$

$$LI_d \leq \nabla^2 r_k(\theta) \leq UI_d, \forall \theta, \text{ for some } 0 \leq L \leq U < \infty.$$

$$\|\nabla r_k(\theta_1) - \nabla r_k(\theta_2)\| \leq U\|\theta_1 - \theta_2\|, \forall \theta_1, \theta_2.$$

(d) Denote the network Hessian function

$$\nabla^2 \mathcal{R}(\Theta) \triangleq \text{diag}\{r_1(\theta_1), \ldots, r_N(\theta_N)\},$$

where $\Theta \triangleq \text{col}\{\theta_1, \ldots, \theta_N\} \in \mathbb{R}^{Nd\times 1}$. It is assumed that $\nabla^2\mathcal{R}(\Theta)$ satisfies the Lipschitz condition:

$$\left\|\nabla^2\mathcal{R}(\Theta_1) - \nabla^2\mathcal{R}(\Theta_2)\right\| \leq \kappa \left\|\Theta_1 - \Theta_2\right\|,$$

for any $\Theta_1, \Theta_2 \in \mathbb{R}^{Nd\times 1}$ and some $\kappa \geq 0$.

Let's denote the stochastic gradient noise as

$$s_{k,i}(\theta_{k,i-1}) \triangleq \widehat{\nabla r_k}(\theta_{k,i-1}) - \nabla r_k(\theta_{k,i-1}). \tag{7.10}$$

We stack the noise of every agent into a vector and obtain the network noise denoted by

$$\mathcal{S}_i(\Theta_{i-1}) \triangleq \text{col}\{s_{1,i}(\theta_{1,i-1}), \ldots, s_{N,i}(\theta_{N,i-1})\}.$$

We use the filtration $\{\mathbb{F}_i; i \geq 0\}$ to represent the information flow that is available up to the i-th iteration of the learning process. Then, the conditional covariance of $\mathcal{S}_i(\Theta_{i-1})$ is denoted by

$$\mathcal{V}_{s,i}(\Theta_{i-1}) \triangleq \mathbb{E}\left[\mathcal{S}_i(\Theta_{i-1})\mathcal{S}_i^{\top}(\Theta_{i-1})|\mathbb{F}_{i-1}\right]. \tag{7.11}$$

Assumption 7.2. (Gradient noise) The gradient noise satisfies the following properties:

(a) The stochastic approximations are unbiased estimates of gradients such that

$$\mathbb{E}\left[\mathcal{S}_i(\Theta_{i-1})|\mathbb{F}_{i-1}\right] = 0.$$

(b) The second-order moment of the stochastic gradient process satisfies:

$$\mathbb{E}\left[\left\|\widehat{\nabla r_k}(\theta_{i-1})\right\|^2 \middle| \mathbb{F}_{i-1}\right] \leq \alpha\|\nabla r_k(\theta_{i-1})\|^2 + \sigma_k^2.$$

for some $\alpha \geq 1, \sigma_s > 0$.

(c) The conditional covariance function satisfies the Lipschitz condition:

$$\|\mathcal{V}_{s,i}(\Theta^*) - \mathcal{V}_{s,i}(\Theta_{i-1})\| \leq \beta\left\|\Theta^* - \Theta_{i-1}\right\|^{\gamma}, \tag{7.12}$$

for some $\beta \geq 0, 0 < \gamma \leq 4$, with

$$\Theta^* \triangleq \text{col}\{\theta_1^*, \ldots, \theta_N^*\} = \text{col}\{\mathbb{1}_{|\mathcal{C}_q|} \otimes \theta_q^o; q = 1, \ldots, Q\}.$$

(d) The conditional covariance matrix at convergence $\mathcal{V}_s \triangleq \lim_{i\to\infty} \mathcal{V}_{s,i}(\Theta^*) > 0$ is symmetric and positive definite.

Theorem 7.1. *Under Assumptions 1-2, given sufficiently small step-sizes $\mu_k \in (0, \frac{1}{U\alpha}]$, every normal agent k using the hypothesis test* (7.7) *for clustering converges towards θ_k^* with*

$$\lim_{i\to\infty} \sup \mathbb{E}\left[r_k\left(\theta_{k,i}\right) - r_k(\theta_k^*)\right] = O(\mu_k). \tag{7.13}$$

Proof. Let $\delta_k(\theta_{k,i}) = \mathbb{E}\left[r_k(\theta_{k,i}) - r_k(\theta_k^*)\right]$. By recursion (7.2)-(7.4) and Assumptions 1-2, using constant step-size $\mu_k \in (0, \frac{1}{U\alpha}]$, it follows from [156](Theorem 4.6) that the following error recursion holds for the SGD step (7.2):

$$\delta_k(\hat{\theta}_{k,i}) - \frac{\mu_k U\sigma_k^2}{2m} \leq (1 - \mu_k m)\left(\delta_k(\theta_{k,i-1}) - \frac{\mu_k U\sigma_k^2}{2m}\right).$$

The hypothesis test (7.7) requires that k cooperates only with the neighbors incurring a loss $\widehat{r_k}\left(\hat{\theta}_{l,i}\right) \leq \widehat{r_k}\left(\hat{\theta}_{k,i}\right)$, which results in (7.9). Thus, it follows that $\delta_k(\theta_{k,i}) \leq \delta_k(\hat{\theta}_{k,i})$. Thus,

$$\delta_k(\theta_{k,i}) - \frac{\mu_k U\sigma_k^2}{2m} \leq (1 - \mu_k m)\left(\delta_k(\theta_{k,i-1}) - \frac{\mu_k U\sigma_k^2}{2m}\right). \tag{7.14}$$

Given $\mu_k \in (0, \frac{1}{U\alpha}]$, with $\alpha \geq 1$, $m \leq U$ (given the property of m-strongly convex and U-Lipschitz continuous gradient), it holds that $(1 - \mu_k m) \in [0, 1)$. Applying (7.14) repeatedly through iteration $i \in \mathbb{N}$, we obtain

$$\delta_k(\theta_{k,i}) \leq \frac{\mu_k U\sigma_k^2}{2m} + (1 - \mu_k m)^i \left(\delta_k(\theta_{k,0}) - \frac{\mu_k U\sigma_k^2}{2m}\right) \overset{i\to\infty}{\longrightarrow} \frac{\mu_k U\sigma_k^2}{2m}, \tag{7.15}$$

and (7.13) holds accordingly. □

Theorem 7.1 indicates that every agent converges to the objective of its cluster $\theta_q^o = \theta_k^*$ if $k \in \mathcal{C}_q$. Given Assumption 1(a), at convergence, $r_k(\theta_q^o) < r_k(\theta_r^o)$ if $k \in \mathcal{C}_q$ and $k \notin \mathcal{C}_r$ for any $r \neq q$. For any agent $l \notin \mathcal{C}_q$, suppose $l \in \mathcal{C}_r$, then it holds that $r_k(\theta_k^*) < r_k(\theta_l^*)$. Then, following hypothesis test (7.7), at convergence, agent k will not cooperate with any neighbor l from a different cluster. Therefore, the network is separated into distinct connected sub-networks (groups) where agents in the same group are from the same cluster.

Assume that there are $G \in [Q, N]$ groups, with N_g agents in group $\mathcal{G}_g$ such that $\sum_{g=1}^{G} N_g = N$. Then, the clustered multi-task network can be reduced to multiple single-task networks (groups). In each group,

let A_g be the $N_g \times N_g$ weight matrix with $A_g \triangleq [\lim_{i\to\infty} a_{lk}(i); l,k \in \mathcal{G}_g]$. Since agents have non-trivial self-loops ($a_{kk}(i) > 0$), it follows that every weight matrix A_g is a left-stochastic and primitive matrix [70]. Then, by the Perron-Frobenius theorem [73, 190], such matrix has a simple eigenvalue at one and all other eigenvalues have magnitude strictly less than one. Moreover, let p_g denote the right eigenvector of A_g that is associated with the eigenvalue at one and normalize its entries to add up to one. Then, the entries of p_g are positive and smaller than one, such that

$$A_g p_g = p_g,\ p_g^\top \mathbb{1} = 1,\ 0 < p_{g,k} < 1,\ \text{with } p_g = [p_{g,k}; k = 1,2,\ldots,|p_g|]\,. \tag{7.16}$$

We can then obtain the average learning performance over the network as described in the following theorem.

Theorem 7.2. *Under Assumptions 1-2, given sufficiently small step-sizes $\mu_k \in (0, \frac{1}{U\alpha}]$ and the clustering hypothesis test* (7.7)*, the average learning performance of the network measured by the mean-squared-deviation (MSD) of the estimated parameters is given by*

$$\lim_{i\to\infty} \sup \frac{1}{N}\mathbb{E}\left\|\widetilde{\Theta}_i\right\|^2 = \frac{1}{2N}\sum_{g=1}^{G} N_g \text{Tr}\left[\left(\sum_{k\in\mathcal{G}_g} p_{g,k}\mu_k H_k\right)^{-1}\left(\sum_{k\in\mathcal{G}_g} p_{g,k}^2\mu_k^2 V_k\right)\right], \tag{7.17}$$

where $p_{g,k}$ subjects to (7.16)*, and*

$$\widetilde{\Theta}_i \triangleq \Theta^* - \Theta_i = \text{col}\{\widetilde{\theta}_{1,i},\ldots,\widetilde{\theta}_{N,i}\},$$

$$H_k \triangleq \nabla^2 r_k(\theta_k^*), \quad V_k \triangleq \lim_{i\to\infty} \mathbb{E}\left[s_{k,i}(\theta_k^*)s_{k,i}(\theta_k^*)^\top|\mathbb{F}_{i-1}\right].$$

Proof. The average MSD over the network is an average of the MSD of each groups. Then, (7.17) can be easily derived from the results of [73] (Theorem 11.3). □

7.4.3 Optimal Combination Weights

Next, we consider how to optimize the weights $a_{lk}(i)$ in order to optimize (7.1), which equivalently optimize each individual loss function. Given the clustering hypothesis test (7.7), at each iteration i, an agent k clusters neighbors in $\mathbb{H}_0$ as $\mathcal{N}_{k,i}^+$ and cooperates only with agents in $\mathcal{N}_{k,i}^+$. Using (7.4), we get an equivalent problem:

$$\min_{A_k} \left\|\sum_{l\in\mathcal{N}_{k,i}^+} a_{lk}(i)\hat{\theta}_{l,i} - \theta_k^*\right\|^2,\ \text{subject to (7.5)},$$

where $A_k = [a_{1k}(i), \ldots, a_{|\mathcal{N}_{k,i}^+|k}(i)] \in \mathbb{R}^{1\times|\mathcal{N}_{k,i}^+|}$. As in a typical approximation approach [97], we consider

$$\left\| \sum_{l\in\mathcal{N}_{k,i}^+} a_{lk}(i)\hat{\theta}_{l,i} - \theta_k^* \right\|^2 \approx \sum_{l\in\mathcal{N}_{k,i}^+} a_{lk}(i)^2 \left\|\hat{\theta}_{l,i} - \theta_k^*\right\|^2. \tag{7.18}$$

Since the loss functions r_k are assumed to be m-strongly convex, it follows that

$$\|\hat{\theta}_{l,i} - \theta_k^*\|^2 \leq \frac{2}{m}\left(r_k(\hat{\theta}_{l,i}) - r_k(\theta_k^*)\right). \tag{7.19}$$

Instead of directly minimizing the right side of (7.18), we consider minimizing its upper bound given in (7.19). Hence, by combining (7.18) and (7.19), we obtain: $\min_{A_k} \sum_{l\in\mathcal{N}_{k,i}^+} a_{lk}^2(i)\left(r_k(\hat{\theta}_{l,i}) - r_k(\theta_k^*)\right)$. Since $r_k(\theta_k^*)$ is small compared to $r_k(\theta_{l,i})$, we consider the following minimization problem:

$$\min_{A_k} \sum_{l\in\mathcal{N}_{k,i}^+} a_{lk}^2(i) r_k(\hat{\theta}_{l,i}), \text{ subject to (7.5).} \tag{7.20}$$

Using the Lagrangian relaxation and the approxmation value $\widehat{r_k}(\cdot)$ to replace $r_k(\cdot)$, we obtain the optimal solution of (7.20) as

$$a_{lk}(i) = \frac{\widehat{r_k}(\hat{\theta}_{l,i})^{-1}}{\sum_{p\in\mathcal{N}_{k,i}^+} \widehat{r_k}(\hat{\theta}_{p,i})^{-1}}. \tag{7.21}$$

Combined with (7.7), we conclude the algorithm for the learning and clustering over networks in Algorithm 4.

Algorithm 4: Distributed learning and clustering over networks

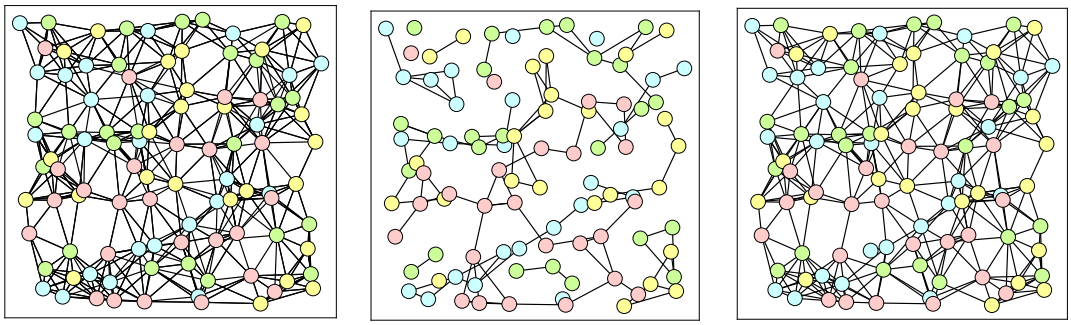
Input: Initialize $w_{k,-1}$ for $k \in \mathcal{C}_q$ and $q = 1, 2, \ldots, Q$.

for $i \geq 0$ **do**
- **for** *every agent* k **do**
  - Update $\hat{\theta}_{k,i}$ according to (7.2).
  - Send $\hat{\theta}_{k,i}$ and receive $\hat{\theta}_{l,i}$ from neighbors.
  - Cluster neighbors using (7.7) and obtain $\mathcal{N}_{k,i}^+$.
  - Assign weights to neighbors in $\mathcal{N}_{k,i}^+$ according to (7.21).
  - Aggregate neighbors' model parameters according to (7.4).

7.5 Evaluation

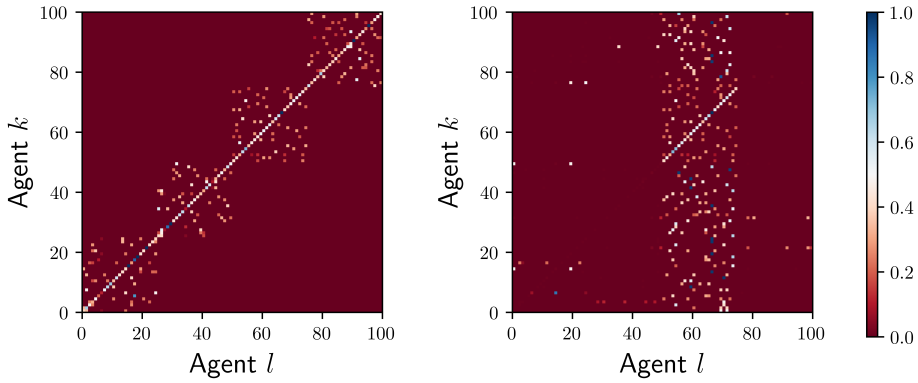
In this section, we evaluate the proposed distributed clustering method with optimal weights computed using Algorithm 4 and compare the results with the case where the agents (1) do not cooperate with each other

(non-cooperative case), (2) cooperate using the average weights ($a_{lk} = \frac{1}{|\mathcal{N}_k|}$), and (3) cooperate using the quadratic distance-based clustering hypothesis test. To avoid the selection of the user-defined parameter $d_{k,l}$ in (7.6), we instead use $\|\hat{\theta}_{l,i} - \theta_{k,i-1}\|^2 \underset{\mathbb{H}_1}{\overset{\mathbb{H}_0}{\lessgtr}} \|\hat{\theta}_{k,i} - \theta_{k,i-1}\|^2$ as proposed in [60]. We consider two distributed learning examples: (1) target localization and (2) digit classification. For the classification problem, we use convolutional neural networks (CNNs) that are non-convex models. We show empirically that Algorithm 4 results in better learning performance than the other methods with correct clustering structure being learned, for both convex and non-convex models (such as CNNs).



(a) (b) (c)

Figure 7.2: Target localization (nodes in the same color share the same target): (a) Initial network; (b) Final network by Algorithm 4; (c) Final network by the distance-based clustering method.



(a) Loss-based (b) Distance-based

Figure 7.3: Target localization: average weight matrix over time $\frac{1}{T+1}\sum_{i=0}^{T} a_{lk}(i)$.

7.5.1 Target Localization

Target localization is a linear regression problem in which the objective is to estimate the location of a target by minimizing the squared error loss of noisy streaming sensor data [185]. We consider a net-

work of 100 agents randomly distributed in a planar region $\mathcal{W} = [0, 10] \times [0, 10] \in \mathbb{R}^2$ as shown in Figure 7.2(a). An edge between two agents means that they are neighbors. Agents with index numbers from 0-24, 25-49, 50-75, 75-99 share the same target, which are indicated using the same color. However, the agents do not know this clustering information beforehand. The four target locations in $\mathbb{R}^2$ are: $(100, 200), (200, 100), (100, 100), (200, 200)$ respectively. At each iteration, every agent k has a noisy observation (streaming data) of the distance $d_k(i)$ and the unit direction vector $u_{k,i}$ pointing from x_k to its target based on built-in sensors. Let $\theta_k \in \mathbb{R}^2$ denote the estimation of the target location for agent k, then the loss is given by $r_k(\theta_{k,i}) = \mathbb{E}\left[\|d_k(i) - (\theta_{k,i} - x_k)^\top u_{k,i}\|^2\right]$. The approximation $\widehat{r_k}(\theta_{k,i})$ is computed by the last 10 stacked streaming datapoints. The distance measurement data has noise variance $\sigma_{d,k}^2 \in [0.1, 1]$, and the unit direction vector has additive white Guassian noise with diagnonal covariance matrices $R_{u,k} = \sigma_{u,k}^2 I_2$, with $\sigma_{u,k}^2 \in [0.01, 0.1]$ for different k. The step-size is set to be $\mu = 0.1$ for every agent.

The results are given in Figure 7.2 - Figure 7.5. From Figure 7.2, the loss-based clustering method (Algorithm 4) results in a clustered network at the end of the simulation (Figure 7.2(b)), and no agents in two different clusters end up being neighbors of each other. On the other hand, the distance-based method fails to correctly capture the clustering information (Figure 7.2(c)). Further, as shown in Figure 7.3, in the proposed method agents cooperate only with neighbors in the same cluster. Using distance-based clustering, the agents failed to identify those belonging to the same cluster. Figure 7.4 shows the learning loss using different clustering methods, where solid lines are the average losses over the network and the shadow area is the range between the minimum and the maximum loss of the networked agents. Our method achieves the minimum average loss and the learning performance is better than the non-cooperative case. Figure 7.5 shows the target's estimation as a function of time. It can be found that our method achieves smoother and more accurate estimations than the non-cooperative case, whereas the average and distance-based methods fail to learn the correct targets.

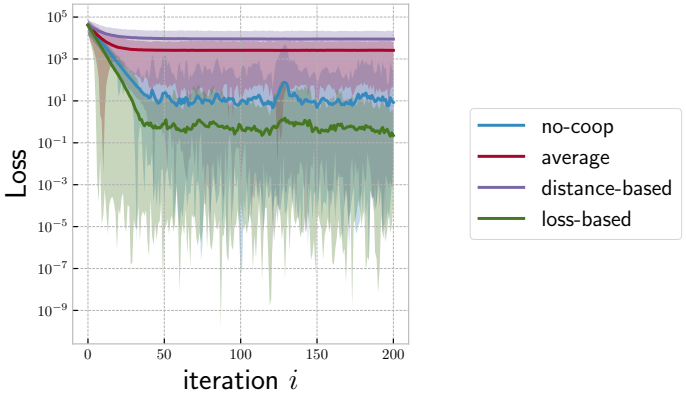
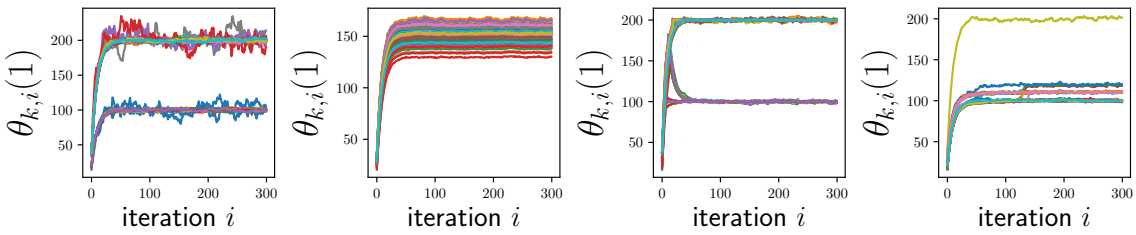


Figure 7.4: Target localization: learning loss for different methods.



(a) No-cooperation (b) Average (c) Loss-based (d) Distance-based

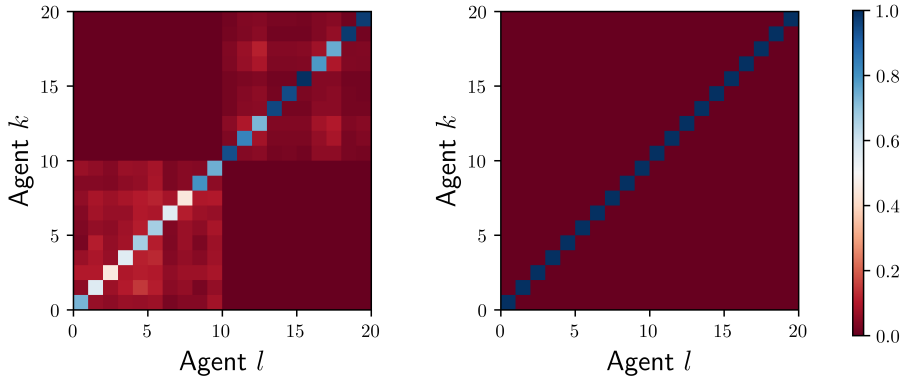
Figure 7.5: Target localization: estimation $\theta_{k,i}$ (1st dimension) of every agent k (each line represents an agent) for different methods.

7.5.2 Digit Classification

We consider a network of twenty agents performing digit classification tasks. We use a complete graph to model the network topology such that every agent is connected to all the other agents. An agent does not know which of its neighbors are performing the same task as the agent itself. Agents indexed by 0-9 have access to the MNIST dataset[1] [186] and agents indexed by 10-19 have access to the synthetic dataset[2] that is composed by the generated images of digits embedded on random backgrounds [187]. All the images are preprocessed to be 28×28 grayscale images. We use a CNN model of the same architecture for each agent and cross-entropy-loss. The CNN architecture we use is the same as in [7]. We consider that agents have access to uneven sizes of training data such that each agent receives $100 - 1000$ training data and 200 testing data from the corresponding dataset for each iteration, similar to the real-world examples. We use mini-batch gradient descent with batch size of 64 for MNIST and 128 for the synthetic digit dataset [156]. The approximation $\widehat{r_k}(\cdot)$ is computed using the mini-batch loss. The step-size $\mu = 0.001$ is set for every agent.

The results are given in Figure 7.6 - Figure 7.7. From Figure 7.6, using the proposed method, the clustering structure is correctly learned and agents cooperate only with neighbors in the same cluster and stop cooperating with the agents from the other cluster. In the case of distance-based clustering, agents do not cooperate with any neighbors and task is reduced to the non-cooperative case. Figure 7.7 shows the training and testing loss using different clustering methods, where solid lines are the average losses over the network and the shadow area is the range between the minimum and the maximum loss of the networked agents. The proposed method achieves the minimum average training and testing loss, and the learning performance is better than the non-cooperative case.

[1] http://yann.lecun.com/exdb/mnist
[2] https://www.kaggle.com/prasunroy/synthetic-digits



(a) Loss-based (b) Distance-based

Figure 7.6: Digit classification: average weight matrix over time $\frac{1}{T+1}\sum_{i=0}^{T} a_{lk}(i)$.

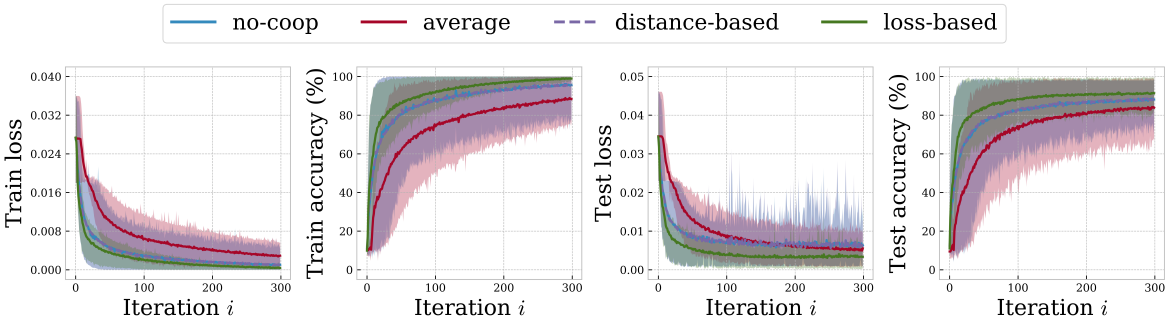


Figure 7.7: Digit classification: learning performance for different methods.

7.6 Conclusion

Accurately capturing the clustering structure in a distributed multi-task learning network has great significance in practice. For example, advertising companies can use such information to target clusters of potential customers who may be interested in a particular product. This chapter proposes an adaptive clustering method for distributed clustered multi-task learning network with fully decentralized agents. We prove the convergence of agents using the proposed method for clustering towards their objectives and analyze the learning performance. We also present the optimal combination weights for aggregating the neighbors' model parameters to optimize network learning performance. In the evaluation, we show that existing clustering methods fail to capture the correct clustering structure and result in worse learning performance. In contrast, the proposed method accurately learns the clustering structure and cooperation within such clusters improves the network learning performance compared to the case where agents do not cooperate with each other.

Chapter 8

Byzantine Resilient Aggregation in Distributed Reinforcement Learning

Recent distributed reinforcement learning techniques utilize networked agents to accelerate exploration and speed up learning. However, such techniques are not resilient in the presence of Byzantine agents which can disturb convergence. In this chapter, we present a Byzantine resilient aggregation rule for distributed reinforcement learning with networked agents that incorporates the idea of optimizing the objective function in designing the aggregation rules. We evaluate our approach using multiple reinforcement learning environments for both value-based and policy-based methods with homogeneous and heterogeneous agents. The results show that cooperation using the proposed approach exhibits better learning performance than the non-cooperative case and is resilient in the presence of an arbitrary number of Byzantine agents.

8.1 Introduction

Due to the growth of machine learning (ML) applications and increasing volumes of data, distributed learning and adaptation methods have been receiving greater attention. In such methods, multiple agents operate in a distributed and cooperative manner to achieve a common learning task. Typically, agents adapt their models using local data and interact with neighbors for model aggregation. Such cooperation has been demonstrated to help boost sample data for each agent and improve learning performance over the network [16]. Distributed reinforcement learning (RL), in particular, has been widely studied and applied in many applications, such as sensor networks, multi-robot networks, mobile phone networks, intelligent transportation systems, especially combined with deep neural networks [125, 191, 192].

Although cooperation in a distributed multi-agent network helps improve the learning performance, such methods are vulnerable to Byzantine attacks. It has been shown that a single Byzantine agent could disturb convergence of the entire network by sending malicious information to its neighbors [2, 18]. To address this challenge, there is considerable recent research focusing on the resilient aggregation of distributed learning algorithms in the presence of Byzantine agents. Many resilient aggregation methods for distributed learning have been developed based on geometric properties of the model parameters such as coordinate-wise median, trimmed-mean, geometric median, Krum, and centerpoint, among many others [4, 18, 52, 57, 62]. One limitation of such approaches is that they are only resilient to a bounded number (usually less than half) of Byzantine neighbors.

Although research in Byzantine resilient aggregation for distributed ML is very broad, studies focusing on resilient distributed RL are limited. The recent studies in [126] and [127] use trimmed mean to achieve resilience for distributed actor-critic and Q-learning algorithms when a bounded number of agents are Byzantine. In this chapter, we propose a Byzantine resilient aggregation rule for distributed RL. Compared to the existing methods that rely on the geometric properties to achieve resilience, the proposed method incorporates the idea of optimizing the objective function in designing the aggregation rule, and does not require a tailored upper bound of the Byzantine agents. In order to maximize the networked rewards, agents assign larger weights to neighbors incurring a larger reward, and stop cooperation with those incurring a smaller reward. Byzantine agents try to disturb the convergence of normal agents by sharing model parameters resulting in a small reward and thus are not taken into account by normal agents. The effectiveness of the proposed method is well validated by the evaluation results using multiple RL tasks for both value-based and policy-based distributed RL, such as distributed deep Q-learning and distributed Deep Deterministic Policy Gradient (DDPG). The evaluation results show that the proposed method exhibits better or similar learning performance (measured by the accumulated reward over the network) than no-cooperation in the presence of an arbitrary number of Byzantine agents.

8.2 Related Work

The technique of training multiple RL agents distributedly for a common objective has been extensively studied in recent years. Related work in distributed RL can be broadly grouped into two categories. In the first category, multiple agents operate in similar but independent Markov decision processes (MDPs) whose actions do not affect each other [108, 109]. Such an approach is widely used in recent RL techniques for parallel exploration and computation to accelerate exploration and speed up learning, especially combined with deep neural networks. It is also naturally suited to be used in multi-agent networks where networked agents perform similar RL tasks in independent environments. The second category considers training RL algorithms with multiple agents in a single MDP [110]. This chapter focuses on the first paradigm.

A major body of related work in training multiple agents in independent MDPs considers using a centralized parameter server for model updates and multiple workers to execute in multiple instances of the environment in parallel to collect state-action pairs and compute gradients of the model. Examples include the distributed deep Q-network [109] and A3C (Asynchronous Advantage Actor-Critic) [111]. Distributed RL in fully-decentralized networks has also been studied in the literature [108, 112–114]. For example, [112] proposes a distributed implementation of Q-learning called QD-learning where every normal agent collaboratively updates tabular Q-values that being shared with their neighbors. Further, [108] proposes a distributed

RL method for policy evaluation with linear value function approximation. Moreover, [113] proposes a distributed actor-critic framework that aims to learn a policy that performs well on average for the whole set of tasks.

Resilient aggregation in distributed learning is a very active research area in recent years. Many consensus-based methods have been proposed to address the vulnerabilities of the cooperation to Byzantine attacks. Examples include coordinate-wise median, trimmed-mean, geometric median, Krum, and centerpoint, among many others [4, 18, 52, 57, 62]. In addition, methods based on measuring the similarities between agents are also considered in the related work, which can be found in [2, 7, 54, 74]. Although research in Byzantine resilient aggregation for distributed learning algorithms is very broad, studies focusing on resilient distributed RL are limited. A recent work presented in [126] uses trimmed-mean to achieve resilience, where a centralized server exists in the network. In addition, a resilient version of QD-learning in a full-decentralized network has been proposed in [127], which is also based on the trimming approach.

8.3 Background

Markov decision processes (MDPs) are widely used for modeling RL problems, which can be described formally as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, R \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ denote the (finite) state and action spaces, $\mathcal{P}(s'|s,a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is a state transition probability and $R(s,a,s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function defined by $R(s,a,s') = \mathbb{E}[r_{t+1}|s_t = s, a_t = a, s_{t+1} = s']$ with r_{t+1} being the immediate reward received at time t. The probability of taking action a in state s is defined by the policy $\pi(a|s) : \mathcal{S} \times \mathcal{A} \to [0,1]$. Moreover, denote the state-value function $V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s, \pi\right]$, with $\gamma \in (0,1)$ as the discounted factor that determines how much future rewards are counted, and the action-value function $Q_\pi(s,a) = \sum_{s'} P^a_{ss'} \left(R(s,a,s') + \gamma V_\pi(s')\right)$, with $P^a_{ss'} = \mathcal{P}(s'|s,a)$. The objective is to learn an optimal policy π^* that maximizes the expected long-term reward given $a_t \sim \pi(\cdot|s_t)$ and $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$:

$$\max_{\pi} \left\{ J(\pi) = \mathbb{E}_s \left[V_\pi(s)\right] \right\}, \tag{8.1}$$

To ensure the existence of solution to (8.1), bounded rewards are assumed for any time-step as $|r_{t+1}| \leq r_{\max} < \infty, \forall t.$ for some scalar $r_{\max}$.

RL algorithms can be broadly categorized into value-based and policy-based. In the following, we briefly introduce the main algorithms for the two types for solving (8.1).

Value-based methods aim to find a good estimate of the Q-function, and indirectly extract the optimal policy by selecting the greedy action in each state according to the estimates of the Q-values. One of the

most popular value-based RL algorithms is Q-learning [193], which uses the Bellman equation as an iterative update. Suppose $Q_\pi(s,a)$ can be parametrized by some parameter w as $Q(s,a;w) \approx Q_\pi(s,a)$. Then w can be updated by performing a gradient descent step on $\min_w \mathbb{E}\left[(y_t - Q(s_t,a_t;w_t))^2\right]$ where $y_t = \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1},a';w_{t-1})|s_t,a_t\right]$ [194].

Policy-based methods directly search over the policy space to find the optimal policy instead of relying on the Q-function. One of the most popular policy-based RL algorithms is the Policy Gradient (PG) method [195]. In PG, the policy is parametrized by some parameter θ, and is updated by performing a gradient descent step on $\max_\theta J(\theta)$ with $\nabla J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla \log \pi_\theta(a_t|s_t)\right]$. Ψ_t can be expressed, for example, as the Q-function $Q_\pi(s_t,a_t)$ or the advantage function $A_\pi(s_t,a_t) = Q_\pi(s_t,a_t) - V_\pi(s_t)$, and can be estimated using Monte-Carlo evaluation [196].

8.4 Problem Formulation

Consider a network of $N+b$ agents operating in parallel based on similar but *independent* MDPs $\mathcal{M}_k = \langle \mathcal{S}_k, \mathcal{A}_k, \mathcal{P}_k, R_k \rangle$. The agents are connected by an *undirected graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ represents the agents and $\mathcal{E}$ represents interactions between agents. We assume that there are $b \geq 0$ Byzantine agents and $N \geq 1$ normal agents. *Normal* agents are those who strictly follow the prescribed algorithm in a network; and *Byzantine* agents are those who do not follow the algorithm and could send arbitrary different information to different neighbors usually with a malicious goal of disrupting the network's convergence. Note that Byzantine agents are *indistinguishable*. A bi-directional edge $(l,k) \in \mathcal{E}$ means that agents k and l can exchange information with each other. The *neighborhood* of k is the set $\mathcal{N}_k = \{l \in \mathcal{V} | (l,k) \in \mathcal{E}\} \cup \{k\}$. Agents share the same state and action spaces $\mathcal{S}_k$ and $\mathcal{A}_k$ but the transition probabilities $\mathcal{P}_k$ and the reward function R_k could be different among agents. Since agents are based on independent MDPs, their actions do not influence each other. Let J_k be the expected long-term reward associated with agent k. The goal is to cooperatively learn the optimal policies π_k for each normal agent k that maximize the global average reward:

$$\max_{\{\pi_k\}_{k=1}^N} \left\{ \frac{1}{N} \sum_{k=1}^{N} J_k(\pi_k) \right\}. \tag{8.2}$$

It is assumed that each normal agent k maintains its own parameter w_k (or θ_k), and uses $Q_k(s,a;w_k)$ (or $J_k(\theta_k)$) to be the local estimates of $Q_{\pi_k}(s,a)$ (or $J_k(\pi_k)$), when running value-based RL (or policy-based RL). Agents share their local estimates of such parameters with neighbors and aggregate the estimates from their neighbors to facilitate their learning. In this chapter, we consider that the aggregation steps take place

after each learning episode[1]. The algorithm used by each normal agent k with value-based or policy-based method for solving (8.2) is given in *Algorithm 5*[2].

Since Byzantine agents could disturb the convergence of normal agents through exchanging malicious messages, we are interested in finding a Byzantine resilient aggregation rule that solves (8.2) in the presence of Byzantine agents.

Algorithm 5: Distributed reinforcement learning with model aggregation

Input: Initialize $w_{k,0}^0$ for value-based RL or $\theta_{k,0}^0$ for policy-based RL

for *episode* i = *0,* M **do**

  Initialize state $s_{k,0}^i$, set $t = 0$;

  while $s_{k,t}^i$ *is not terminal* **do**

    `/* Exploration and learning */`

    Select $a_{k,t}^i \sim \pi_k(\cdot|s_{k,t}^i)$) ; Execute $a_{k,t}^i$, observe $r_{k,t+1}^i$ and $s_{k,t}^i$;

    Update $w_{k,t}^i$ if using value-based method or $\theta_{k,t}$ if using policy-based method[3];

    $t = t + 1$;

  `/* Model parameters aggregation */`

  Set $w_{k,\infty}^i = w_{k,t}^i$ or $\theta_{k,\infty}^i = \theta_{k,t}^i$; Exchange $w_{k,\infty}^i$ or $\theta_{k,\infty}^i$ with neighbors ;

  Assign weights $c_{lk}(i)$ according to $w_{l,\infty}^i$ or $\theta_{l,\infty}^i$;

  Aggregate estimates $w_{k,0}^{i+1} = \sum_{l\in\mathcal{N}_k} c_{lk}(i) w_{l,\infty}^i$ or $\theta_{k,0}^{i+1} = \sum_{l\in\mathcal{N}_k} c_{lk}(i)\theta_{l,\infty}^i$;

8.5 Resilient Aggregation in Distributed RL

In this section, we present resilient aggregation rules applicable to both value-based and policy-based distributed RL. The goal is to design non-negative weights $C_k(i) = [c_{1k}(i), \ldots, c_{(N+b)k}(i)] \in \mathbb{R}^{1\times(N+b)}$ for every normal agent k, with $\sum_{l\in\mathcal{N}_k} c_{lk}(i) = 1$, and $c_{lk}(i) = 0$ if $l \notin \mathcal{N}_k$, such that by using $C_k(i)$ in *Algorithm 5*, and in the presence of an arbitrary number of Byzantine neighbors, (8.2) is solved. We incorporate the idea of optimizing the objective function in designing the aggregation weights. Besides, a softmax layer is applied to the weights in order to make the weights to be non-negative. The weights are assigned as follows: if $l \notin \mathcal{N}_k^{\geq}$, then $c_{lk}(i) = 0$, otherwise,

$$c_{lk}(i) = \frac{e^{\hat{J}_k(\pi_l^i)}}{\sum_{p\in\mathcal{N}_k^{\geq}} e^{\hat{J}_k(\pi_p^i)}}, \tag{8.3}$$

where $l \in \mathcal{N}_k^{\geq}$ if $l \in \mathcal{N}_k$ and $\hat{J}_k(\pi_l^i) \geq \hat{J}_k(\pi_k^i)$; $\hat{J}_k(\cdot)$ is an approximation of $J_k(\cdot)$ with $\mathbb{E}_s[\hat{J}_k(\cdot)] = J_k(\cdot)$. For example, $\hat{J}_k(\pi)$ can be computed by the simulated long-term reward of one-shot Monte-Carlo policy

[1] An episode is a sequence of states from the start state to a terminal state.

[2] During learning, t increases while i remains the same; and during model aggregation, i increases while t changes from ∞ to 0. To simplify, hereafter, we omit the subscripts of t for cooperation and the superscripts of i for learning.

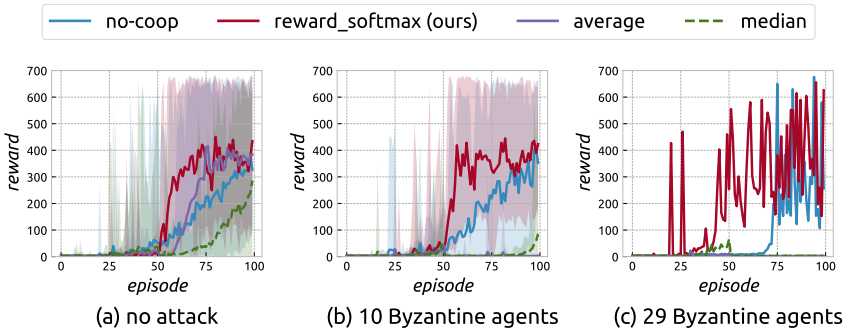
[3] Methods for updating these parameters are discussed in Section 8.3.

evaluation using policy π on the MDP of k. Note that π_l^i can be either extracted from w_l^i when using value-based RL or can be parametrized by θ_l^i when using policy-based RL. Obviously, $\sum_{l\in\mathcal{N}_k} c_{lk}(i) = 1$ holds using weights (8.3). The intuition behind (8.3) is the following. One agent k can evaluate the policy of a neighbor l on its own MDP, and a larger long-term reward resulted by a neighbor's policy on k's MDP implies a better approximation of the policy and agent k should assign larger weights to such policies. In addition, it is reasonable to cooperate with neighbors incurring better approximations but unnecessary to cooperate with those resulting in worse approximations than oneself. As a result, we come up with the aggregation rule given in (8.3).

Discussion of the convergence. Convergence for RL algorithms is hard to be guaranteed, especially when combined with neural networks. In the convergence analysis of RL algorithms, it is often assumed that the value or policy functions can be parametrized by a class of linear functions. If this is the case, the aggregation using (8.3) results in $\hat{J}_k(\pi_k^{i+1}) \geq \hat{J}_k(\pi_k^i)$, given that the agent cooperates only with the neighbors having a larger approximate reward $\hat{J}_k(\pi_l^i) \geq \hat{J}_k(\pi_k^i)$, and $\hat{J}_k(\pi_k^{i+1})$ is a linear combination of $\hat{J}_k(\pi_l^i)$. This means that the aggregation using (8.3) always results in a larger approximate long-term reward, i.e., a better approximation of the policy. Further, the presence of Byzantine agents does not affect this improvement. As a result, if agents converge to the optimal policy without cooperation, they also converge in the cooperative case in the presence of byzantine agents. If agents do not converge to the optimal policy without cooperation, cooperation using (8.3) could help improve their learned policy.

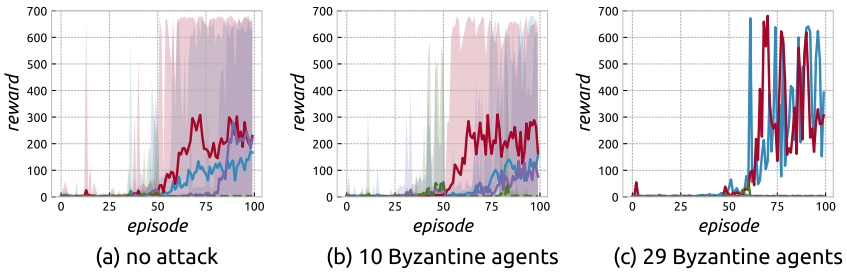
8.6 Evaluation

In this section, we evaluate the proposed resilient aggregation method for both value-based and policy-based RL algorithms. We also compare the approach with the average- and median-based aggregation rules as well as the non-cooperative case. Note that median is a special case of trimmed-mean when half of the smallest and largest values are trimmed. In all the examples, our approach exhibits better or similar learning performance than the non-cooperative case measured by the averaged long-term rewards over the network, in the presence of an arbitrary number of Byzantine neighbors. When all the neighbors are Byzantine, the approach is reduced to a non-cooperative algorithm. Whereas in the same scenarios, average and median-based methods may exist worse learning performance than the non-cooperative case, showing the vulnerabilities of such aggregation methods in Byzantine systems.



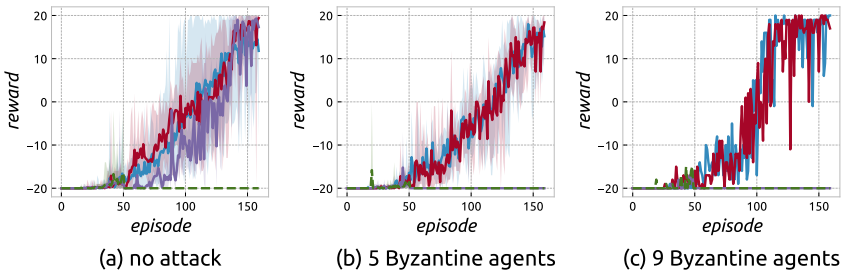
(a) no attack (b) 10 Byzantine agents (c) 29 Byzantine agents

Figure 8.1: 30 homogeneous agents running DQN for Cartpole.



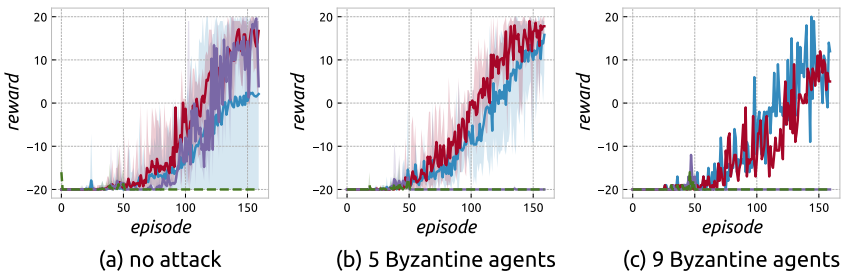
(a) no attack (b) 10 Byzantine agents (c) 29 Byzantine agents

Figure 8.2: 30 heterogeneous agents running DQN for Cartpole.



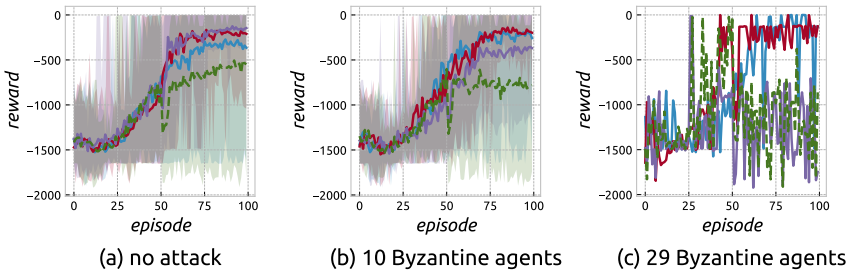
(a) no attack (b) 5 Byzantine agents (c) 9 Byzantine agents

Figure 8.3: 10 homogeneous agents running DQN for Pong.



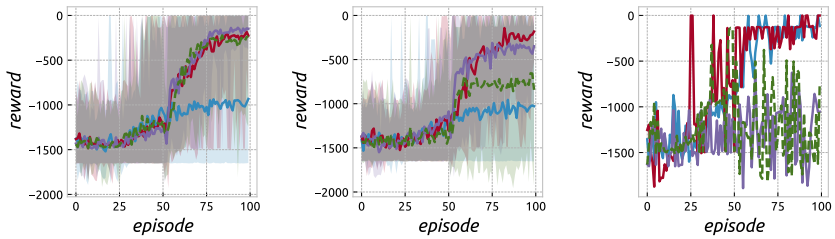
(a) no attack (b) 5 Byzantine agents (c) 9 Byzantine agents

Figure 8.4: 10 heterogeneous agents running DQN for Pong.



(a) no attack (b) 10 Byzantine agents (c) 29 Byzantine agents

Figure 8.5: 30 homogeneous agents running DDPG for Pendulum.



(a) no attack (b) 10 Byzantine agents (c) 29 Byzantine agents

Figure 8.6: 30 heterogeneous agents running DDPG for Pendulum.

8.6.1 Simulation Setup

Tasks. The tasks we consider are the classic control problems Cartpole and Pendulum, and the Atari game Pong based on the OpenAI Gym [197]. Specifically, we evaluate both the value-based RL algorithm deep Q-networks (DQN) [194] for Cartpole and Pong, and policy-based algorithm DDPG [198] for Pendulum.

Network. For the Cartpole and Pendulum tasks, we consider a network of 30 agents; and for Pong, we consider a network of 10 agents. The connectivity of the agents is determined by their geographical location, which is randomly drawn from a $[0, 3] \times [0, 3]$ plane. The average degree $\frac{1}{N}\sum_{k=1}^{N} |\mathcal{N}_k|$ of the connectivity graph with 30 agents is approximately 8.2. The network with 10 agents is modeled by a complete graph.

Attacks. Byzantine agents are designed to send random values (for each dimension) from the interval [0, 1] to all of its neighbors.

Hyper-parameters. In all the tasks, we use ADAM optimizer [199], $\gamma = 0.99$, and ε-greedy exploration strategy with ε annealed from 1 to 0.01 over the first 1e5 learning steps [109] for DQN. The aggregation between the agents starts after the agents run 50 episodes' exploration and learning independently. For Cartpole, the batch-size is 32 and the neural network model for each agent has one hidden layer of 50 neurons. For Pong, the batch-size is 32 and we use CNN with 3 convolutional layers of output shape $[32, 20, 20]$, $[64, 9, 9]$ and $[64, 7, 7]$, as well as a fully-connected layers of 512 neurons and an output layer. For Pendulum, the batch-size is 128 and the actor network has 3 hidden layers with 256/128/64 neurons; The critic network has two layers with 256/128 neurons for the state input and one layer with 128 neurons for the action input, which then concatenates the state/action output of 256 neurons followed by a fully-connected layer of 128 neurons and an output layer. The activation functions are all ReLU. In a connected network, every agent shares the same model architecture (and the same state/action spaces).

8.6.2 Simulation Results

We consider both homogeneous networks where agents use the same learning rate and environment as well as heterogeneous networks where the agents have different learning rates and/or random noisy data being added to each state.

Homogeneous agents. We set the same learning rate and other parameters for each normal agent. The learning rate is 0.01 for Cartpole, 0.001 for Pendulum, and 0.0001 for Pong. Figure 8.1, 8.3 and 8.5 show the mean and range of the evaluated accumulated reward for every normal agent in the network, in the case of no attack, with 5/10 Byzantine agents, and with 9/29 Byzantine agents. We find that the proposed aggregation method is resilient in all scenarios (even when there is only one normal agent in the network and the other agents are all Byzantine) and exhibits better or similar learning performance as the non-cooperative case. The average and median rules fail to converge in the presence of Byzantine agents. It should be noted that the median-based aggregation may fail to converge even without Byzantine agents in the network.

Heterogeneous agents. In the heterogeneous networks, we consider a random learning rate sampled from $(0, 0.1]$ for Cartpole; from $[0.0001, 0.0002]$ for Pong; and from $(0, 0.01]$ for Pendulum. We also consider random noise sampled from $(0, 0.02]$ being added to each element of the state for Cartpole and Pendulum. Figure 8.2, 8.4 and 8.6 show the results. The proposed method outperforms no-cooperation, average, and median as measured by the average accumulated rewards over the network, with and without Byzantine agents. Comparing to the results of the homogeneous setting, we find that in heterogeneous networks, agents that are not able to reach a good policy by themselves could greatly improve their learning performance by cooperating with neighbors. In general, the averaged accumulated reward over the network is greatly improved by model aggregation using the proposed weights compared to the non-cooperative case.

8.7 Conclusion

In this chapter, we present a Byzantine resilient aggregation rule for distributed reinforcement learning with networked agents. In order to maximize the networked rewards, agents assign larger weights to neighbors incurring a larger reward and reduce cooperation with those incurring a smaller reward. Byzantine agents try to disturb the convergence of normal agents and share a value function or policy resulting in small rewards, and thus, they are not included in the cooperation with normal agents. We analyze the approach in the case of linear function approximations to guarantee convergence in the presence of Byzantine agents. Finally, we evaluate our approach using multiple RL environments, for both value- and policy- based methods, with homogeneous and heterogeneous agents. The simulation results validate the effectiveness of our approach.

Chapter 9

Adaptive Learning from Peers for Distributed Actor-Critic Algorithms

Face recognition, health tracker, and recommender system are just a few of the machine learning applications built upon distributed devices. Training machine learning models over a network of users in a fully decentralized network has great potentials in dealing with the massive amounts of data generated by such distributed devices. Due to the privacy and security concerns, models are usually trained in a distributed fashion using individual data, yet cooperation among agents by sharing and aggregating their model parameters has been demonstrated to benefit the learning performance over the network. In this chapter, we consider the problem of training distributed reinforcement learning models collaboratively using actor-critic algorithms. We propose an efficient adaptive cooperation strategy by promoting the similarities among agents and assigning adaptive weights in aggregating the parameters from neighbors. Further, we analyze the convergence of the proposed method when linear function approximations are applied. Finally, extensive simulation results are provided to validate the proposed method, showing the effectiveness of the proposed method in improving the learning performance over the network in the case where agents are performing the same task, multi-task, as well as when some of the agents are attacked. In contrast, other aggregation methods, such as average and median, result in inferior learning performance in certain scenarios.

9.1 Introduction

Distributed learning has attracted increasing attention due to the growth of Machine Learning (ML) applications in multi-agent networks, such as mobile phones, wearable devices, and smart homes [48–51]. In such networks, multiple agents operate in a distributed and cooperative manner to achieve a common learning task. Compared to federated learning [188], distributed learning with networked agents is fully-decentralized, without the need of a central server, thus addresses the problem of single-point-of-failure and the scalability issues. For example, consider learning the behavior of users in a cellular network based on data generated using various mobile applications. Each user may generate data that follows a distinct distribution and it is common to learn separate models for each user. However, people may exhibit similar behaviors and relatedness among models commonly exists [15]. Hence, cooperation among agents could be leveraged to promote the learning performance over the network. Due to privacy and security concerns, agents typically communicate their model parameters with their local neighbors without sharing the user data. Such cooperation has

been demonstrated to help improve the learning performance over the network while not compromising data privacy [16].

Although personalizing customer interactions at scale through the data analysis of users' online behavior patterns has been realized by machine learning, special learning tasks, such as news recommendation is still challenging. The reasons are that user preferences in topics change frequently and the features of those contents are dynamic by nature and become rapidly irrelevant. Reinforcement Learning (RL), in particular, can be applied to address such challenges [115–117]. Fully-decentralized multi-agent RL (MARL) is still a nascent field compared to distributed learning. Previous studies in MARL focused on the case where agents learn in a shared Markov Decision Process (MDP) and the policy developed by each agent should take into account the joint action made by all the other agents [125, 200]. In contrast, we are interested in the scenario where agents learn in similar and *independent* MDPs. They make individual decisions and their actions do not influence each other. The cooperation is achieved by exchanging and aggregating model parameters from neighbors, which promotes the similarities among agents and benefits learning.

In this chapter, we consider the problem of training multiple RL agents in a fully-decentralized network using actor-critic algorithms. Agents make individual decisions and they share their model parameters (both actor and critic) with local neighbors for consensus that updates their parameters as a linear combination of their neighbors' model parameters. Average and median are commonly used as aggregation functions in model consensus [125, 126]. However, these methods cannot be used to infer the underlying relatedness among agents and could do harm to the overall learning performance when agents are performing different tasks or when some agents are learning from untrusted sources. There are metrics that can be used to promote the similarity among networked agents such as Kullback-Leibler (KL) divergence and Wasserstein distance [56, 201]. However, these metrics are used for measuring how one probability distribution is different from another and is restricted to stochastic policies that model actions by a probability distribution. Yet they are not applicable to deterministic policies that map each state to a deterministic action.

To address the problem, we propose a distributed actor-critic algorithm using adaptive weights for aggregating the model parameters from neighbors that incorporates the idea of optimizing the objective function in designing the combination weights. To measure the similarity among agents, we fit the model parameters of neighbors into the sampled data of agents. In the case of linear function approximations, the gradient norm of the neighbors' model parameters on one's sampled data measures the proximity of the neighbor's estimate to the optimum point. As the estimate approaches the optimum point, the gradient norm converges to zero. Measuring the gradient norm is not applicable to neural network models since many local optimums exist in a neural network. Hence, when using neural network models, we instead compute the loss of the actor and critic network using neighbor's models and one's sampled data. A smaller loss indicates a better

estimate and is assigned a larger weight to aggregate with. To illustrate, for the Deep Deterministic Policy Gradient (DDPG) algorithm, the actor aims to minimize the negative expected action-value function and the critic aims to minimize the temporal difference (TD) error. Our method scores the action-value generated by neighbor's policies by one's own critic given sampled states and assigns large weights if the negative action-value computed by the agent is small. For the critic, TD errors of neighbors are computed using the agent's own target critic network and neighbor's critic network given sampled data. Larger weights are assigned to those incurring smaller TD errors.

The contributions of the chapter are:

- We propose efficient adaptive learning methods for distributed actor-critic algorithms that allow agents to learn which neighbors to cooperate with in a fully-decentralized network.
- We analyze the convergence of distributed actor-critic algorithms with the proposed method when linear function approximations are applied.
- We evaluate the proposed method for multiple actor-critic algorithms including DDPG, Soft Actor-Critic (SAC), Twin Delayed DDPG (TD3). The evaluation results show that the proposed method greatly improves the learning performance than the non-cooperative case in all the scenarios. Besides, when all the agents share the common learning task, the efficiency and performance of the proposed method matches that of averaging method. Moreover, the proposed method results in superior performance in multi-task networks and in the presence of attacked agents – it is resilient even when all one's neighbors are attacked, which matches the performance of the non-cooperative case.

9.2 Distributed Actor-Critic in Multi-Agent Networks

Markov Decision Processes (MDPs) can be characterized as a tuple $\langle \mathcal{S}, \mathcal{A}, P, r \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ denote the finite state and action spaces, respectively, $P(s'|s,a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is the state transition probability from state s to the next state s' determined by action a, and $R(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function defined as $R(s,a) = \mathbb{E}\left[r_{t+1}|s_t = s, a_t = a\right]$, with r_{t+1} being the immediate reward received at time t. The agent's action is characterized by a policy π given as a mapping $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$, representing the probability of choosing action a at state s.

Consider a network of N connected agents modeled by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that could be time-varying, where the set of nodes $\mathcal{V}$ represents the agents, and the set of edges $\mathcal{E}$ represents the interaction between them. An edge $(l,k) \in \mathcal{E}, l,k \in \mathcal{V}$ signifies that agents k and l exchange information with each other. The neighborhood of agent k is the set of agents that it interacts with including itself, and is

represented as $\mathcal{N}_k = \{l \in \mathcal{V} | (l,k) \in \mathcal{E}\} \cup \{k\}$, which can be time-varying depending on the underlying graph. Agents operate in similar and *independent* MDPs each can be modeled by $\mathcal{M}_k = \langle \mathcal{S}, \mathcal{A}, P^k, r^k \rangle$ for $k \in \{1, 2, \ldots, N\}$. The state action spaces $\mathcal{S}, \mathcal{A}$ are the same for every agent, but the transition probabilities and the reward function can be different. Since agents operate in independent environments, their actions do not influence each other [108], which is a valid setup in reality, e.g., in a mobile application network. The expected time-average return of policy π for agent k is defined as

$$J_k(\pi) = \lim_T \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}(r_{t+1}^k) = \sum_{s \in \mathcal{S}} d_\pi^k(s) \sum_{a \in \mathcal{A}} \pi(s,a) R^k(s,a) \tag{9.1}$$

where $d_\pi^k = \lim_t P^k(s_t = s|\pi)$ is the stationary distribution of the Markov chain under policy π for agent k. Further, we define the action-value associated with a state-action pair (s,a) under policy π for agent k as $Q_\pi^k(s,a) = \sum_t \mathbb{E}\left[r_{t+1}^k - J_k(\pi) | s_0 = s, a_0 = a, \pi\right]$. It is usually assumed that $Q_\pi^k(s,a)$ can be approximated by some parametrized functions $Q^k(\cdot, \cdot; w_k)$ with parameters w_k. And the policy π can also be parametrized by π_{θ_k} with parameter θ_k. The objective for the networked agents is to cooperatively learn the optimal policy that maximizes the following optimization function:

$$\max_{\theta_1, \ldots, \theta_N} \left\{ \frac{1}{N} \sum_{k=1}^{N} J_k(\theta_k) \right\}. \tag{9.2}$$

We consider each agent runs a separate actor-critic algorithm with a consensus step to solve (9.2). A common actor-critic algorithm based on action-value function approximation with consensus is given below, with the critic step (9.3) and the actor step (9.4):

$$\mu_{t+1}^k = (1 - \beta_{w,t}^k) \cdot \mu_t^k + \beta_{w,t}^k \cdot r_{t+1}^k,\ \tilde{w}_t^k = w_t^k + \beta_{w,t}^k \cdot \delta_t^k \cdot \nabla_w Q_t^k(w_t^k),\ w_{t+1}^k = \sum_{l \in \mathcal{N}_k} c_t(k,l) \cdot \tilde{w}_t^l. \tag{9.3}$$

$$\tilde{\theta}_t^k = \theta_t^k + \beta_{\theta,t}^k \cdot Q_t^k(w_t^k) \cdot \psi_t^k, \quad \theta_{t+1}^k = \sum_{l \in \mathcal{N}_k} b_t(k,l) \cdot \tilde{\theta}_t^l. \tag{9.4}$$

Here, $Q_t^k(w) \triangleq Q^k(s_t^k, a_t^k; w)$, $\delta_t^k = r_{t+1}^k - \mu_t^k + Q_t^k(w_t^k) - Q_t^k(w_t^k)$ denotes the action-value temporal difference (TD) error, and $\psi_t^k = \nabla_\theta \log \pi_{\theta_t^k}(s_t^k, a_t^k)$. Besides, $\beta_{w,t}^k, \beta_{\theta_t}^k > 0$ are stepsizes. The term $Q_t^k(w_t^k) \cdot \psi_t^k$ and $\delta_t^k \cdot \nabla_w Q_t^k(w_t^k)$ can be regarded as the gradient of the actor and critic parameters. Note that the critic step operates at a faster time scale to estimate the action-value function $Q_t^k(w_t^k)$ under policy $\pi_{\theta_t^k}$. And the actor step improves the policy by gradient ascent at a slower time scale. There is a combination step in both the actor and the critic updates that aggregates the neighbors' parameter estimates with the weight matrix $C_t = [c_t(k,l)]_{N \times N}$ and $B_t = [b_t(k,l)]_{N \times N}$, where $c_t(k,l)$ and $b_t(k,l)$ are the row-stochastic weights assigned from k to l at time t for critic and actor's parameters aggregation, respectively. One can easily find

that (9.2) can be optimized by individual agent without cooperation. However, cooperation among agents is beneficial in improving the network learning performance [16]. In this chapter, we are interested in finding the optimal combination matrix B_t and C_t that solves (9.2) in a cooperative manner.

9.3 Adaptive Learning in Distributed Actor-Critic Algorithms

In this section, we propose the optimal combination weights for solving (9.2). It has been demonstrated in the literature that when agents share common objectives, the cooperation among them will promote the learning performance over the network. However, when agents pursue different objectives, indiscriminate cooperation leads to undesired outcomes [103, 105]. Therefore, the combination weight matrix should be designed such that it accurately captures the similarity among agents. Metrics such as KL divergence and Wasserstein distance can be used to promote the similarity among probability distributions [56, 201]. However, these metrics are restrictive to stochastic policies that model actions by a probability distribution, but not applicable to deterministic policies that map each state to a deterministic action. The intuition behind our method is that when two agents share similar objectives, their estimated model parameters should be able to fit to the sampled data of one another. The better one's model fits to the other's data, the more the model is related to the other's underlying task. In the following, we discuss the method for measuring how a model fits to a sampled data of another, for linear function approximations and neural network (nonlinear) function approximations, respectively.

9.3.1 In the Case of Linear Function Approximations

To introduce the proposed method, we make the following assumptions about the action-value and policy functions.

Assumption 9.1. *For each agent k, the action-value function is parameterized by a class of linear functions: $Q_t^k(w) = w^\top \phi_w^k(s_t, a_t)$ with the uniformly bounded feature vector $\phi_w^k(s,a) = \left[\phi_{w,1}^k(s,a), \ldots, \phi_{w,d_w}^k(s,a)\right]^\top \in \mathbb{R}^{d_w}$ for any $s \in \mathcal{S}, a \in \mathcal{A}$. In addition, the feature matrix $\Phi_w^k \in \mathbb{R}^{|\mathcal{S}|\cdot|\mathcal{A}|\times d_w}$ has full column rank, where the j-th column of Φ_w^k is $[\phi_{w,j}^k(s,a), s \in \mathcal{S}, a \in \mathcal{A}]^\top$, for $j \in [d]$. And for any $u \in \mathbb{R}^{d_w}$, $\Phi_w^k u \neq \mathbb{1}$.*

Assumption 9.2. *For each agent k, the policy function is modeled by a Gaussian function $\pi_{\theta_k}(s,a) = \frac{1}{\sigma_k\sqrt{2\pi}}\exp(-\frac{(a-\mu)^2}{2\sigma_k^2})$ with $\mu_k(s) = \theta_k^\top \phi_\pi^k(s)$ as the mean, $\sigma_k > 0$ as the constant variance, and $\phi_\pi^k(s) \in \mathbb{R}^{d_\theta}$ as the uniformly bounded feature vector.*

Define $G_{k,t}^w(w) \triangleq \delta_t^k \cdot \nabla_w Q(s_t, a_t; w)$ and $G_{k,t}^\theta(\theta) \triangleq Q_t^k(w_t^k) \cdot \nabla_\theta \log \pi_\theta(s_t, a_t)$ as the gradient of the action-value and policy functions. Given the above assumptions, we can write the $G_{k,t}^w(w)$ and $G_{k,t}^\theta(\theta)$ as

linear functions, as described below.

Lemma 9.1. [1] *Under Assumption 9.1 and 9.2, we can write $G^w_{k,t}(w)$ and $G^\theta_{k,t}(\theta)$ as:*

$$G^w_{k,t}(w) = \gamma^k_t + w^\top \phi^k_t, \quad G^\theta_{k,t}(\theta) = \xi^k_t - \theta^\top \zeta^k_t,$$

where $\gamma^k_t \triangleq \big(r^k_{t+1} - \mu^k_t\big)\cdot\phi^k_w(s_t,a_t)$, $\phi^k_t \triangleq \big(\phi^k_w(s_{t+1},a_{t+1}) - \phi^k_w(s_t,a_t)\big)\cdot\phi^k_w(s_t,a_t)$. *And* $\xi^k_t \triangleq a_t\phi^k_\pi(s_t)\cdot \frac{Q^k_t(w^k_t)}{\sigma^2_k}$, $\zeta^k_t \triangleq \phi^k_\pi(s_t)\phi^k_\pi(s_t)\cdot\frac{Q^k_t(w^k_t)}{\sigma^2_k}$.

Observe that at convergence, the gradient norm of the actor and critic parameters converges to zero in expectation, i.e.,

$$\lim_{t\to\infty}\mathbb{E}\left[\left\|G^w_{k,t}(w^k_t)\right\|\right] = 0, \quad \lim_{t\to\infty}\mathbb{E}\left[\left\|G^\theta_{k,t}(\theta^k_t)\right\|\right] = 0. \tag{9.5}$$

Since gradients can be expressed as linear functions, the solution to (9.5) is unique. Therefore, we can use the gradient norm as a measure of the proximity of a model estimate to the optimum point. Models with smaller gradient norms are considered to be closer to the optimum point. Note that the idea of measuring the proximity of a point to the convergence point using gradient norm is also considered in [74]. Commonly used norms are the ℓ_1, ℓ_2, and the ℓ_∞ norm. In this chapter, we adopt ℓ_2 norm, but the other norms can also be used. We derive the adaptive combination weights for the critic and actor step as follows.

$$c_{lk}(t) = \frac{\left\|G^w_{k,t}(\tilde{w}^l_t)\right\|^{-1}}{\sum\limits_{p\in\mathcal{N}^{(w)\leq}_k}\left\|G^w_{k,t}(\tilde{w}^p_t)\right\|^{-1}},\ \text{for } l\in\mathcal{N}^{(w)\leq}_k,\ b_{lk}(t) = \frac{\left\|G^\theta_{k,t}(\tilde{\theta}^l_t)\right\|^{-1}}{\sum\limits_{p\in\mathcal{N}^{(\theta)\leq}_k}\left\|G^\theta_{k,t}(\tilde{\theta}^p_t)\right\|^{-1}},\ \text{for } l\in\mathcal{N}^{(\theta)\leq}_k. \tag{9.6}$$

Here, $l\in\mathcal{N}^{(w)\leq}_k$ if $\|G^w_{k,t}(\tilde{w}^l_t)\| \leq \|G^w_{k,t}(\tilde{w}^k_t)\|$; And $l\in\mathcal{N}^{(\theta)\leq}_k$ if $\|G^\theta_{k,t}(\tilde{\theta}^l_t)\| \leq \|G^\theta_{k,t}(\tilde{\theta}^k_t)\|$. For $l\notin\mathcal{N}^{(w)\leq}_k$, $c_{lk}(t)=0$; And for $l\notin\mathcal{N}^{(\theta)\leq}_k$, $b_{lk}(t)=0$. The proposed method in (9.6) enables agents to weight neighbors based on how close the neighbors' estimates are to the optimum point of the agent using the gradient norm measured by the sampled data of the agent and the model parameters of its neighbors. A better estimate will have a smaller gradient norm and be assigned a larger weight. And cooperation using such combination weights contributes in converging to the optimum point, making the cooperation beneficial to the overall performance of the network. We will prove the convergence of the distributed actor-critic algorithm in (9.3) and (9.4) with (9.6) in Section 9.4.

9.3.2 Generalize the Method to Neural Networks

The method in (9.6) assigns adaptive weights based on the gradient norm when gradient functions are parametrized by a class of linear functions. For neural networks, using the method in (9.6) could cause

[1] All the proofs are given in the Appendix.

problems since parameters in a local optimum also have zero gradient norm. To address this issue, we consider assigning adaptive weights based on the loss values instead of the gradient norms. In the case of linear function approximations, the loss is minimized when the gradient norm is zero. Yet for neural networks, a smaller loss means a better estimation, whereas the gradient norm is not a clear measurement of how good the estimation is given the existence of many local optimums. As a result, the method in (9.6) can be seen as a special example of (9.7) when using linear function approximations.

With neural network-based function approximations, the adaptive weights can be assigned as

$$c_{lk}(t) = \frac{L^w_{k,t}(\tilde{w}^l_t)^{-1}}{\sum\limits_{p\in\mathcal{N}_k} L^w_{k,t}(\tilde{w}^p_t)^{-1}}, \quad b_{lk}(t) = \frac{\exp(-L^\theta_{k,t}(\tilde{\theta}^l_t))}{\sum\limits_{p\in\mathcal{N}_k}\exp(-L^\theta_{k,t}(\tilde{\theta}^p_t))}, \text{ for } l \in \mathcal{N}_k, \tag{9.7}$$

where $L^w_{k,t}(w)$ and $L^\theta_{k,t}(\theta)$ are the losses for the critic and actor for agent k at time t, with model parameter w and θ respectively. Since $L^w_{k,t}(\cdot)$ often uses mean-squared TD-error, it is guaranteed to be non-negative. However, $L^\theta_{k,t}(\cdot)$ could be negative values and thus we apply softmax function to it which guarantees the weights to be non-negative. Note that in (9.6), only neighbors resulting in a better estimation are assigned non-negative weights, which is a key step in guaranteeing the convergence of the distributed actor-critic algorithm. However, in practice, neighbors may not be able to achieve smaller loss values than the agent itself, rendering the weights assigned to neighbors to be zero. To ensure the cooperation takes place, we remove the constraints that agents only cooperate with those with better estimation in (9.7). As an example, consider the case for DDPG. In DDPG, the actor function $\mu(s|\theta)$ specifies the current policy by deterministically mapping a state to a specific action. The critic parameters are updated by minimizing the mean-squared TD error; And the actor parameters are updated by maximizing the expected action-value $\mathbb{E}[Q(s,\mu(s))]$, or minimizing $-\mathbb{E}[Q(s,\mu(s))]$, i.e.,

$$L^w_{k,t}(w) = \frac{1}{B}\sum_{i=1,\ldots,B}(y^k_i - Q^k_i(w))^2, \quad L^\theta_{k,t}(\theta) = \frac{1}{B}\sum_{i=1,\ldots,B}(-Q^k_i(s_i,\mu^k(s_i|\theta)),$$

where $y^k_i = r^k_i + \gamma Q'^k(s_{i+1}, \mu'^k(s_{i+1}|\theta'^k_t); w'^k_t)$, with Q'^k and μ'^k as the target value function and policy function parametrized by w'^k_t and θ'^k_t, respectively; and B is the size of the sampled minibatch of data (s_i, a_i, r_i, s_{i+1}) at time t. And the weights can be assigned using (9.7) accordingly.

Time complexity. In order to compute the combination weights in (9.6) or (9.7), one needs to compute the gradient norm, or the batch loss, for each neighbor with the sampled data. The computational time is linear in the dimension of the model parameters and the neighborhood size given the size of the batch data B as a constant. Thus, the time complexity is $O(d\cdot|\mathcal{N}_k|)$ for agent k, where $d = d^w$ for critic and $d = d^\theta$ for actor. Thus, the methods proposed in (9.6) and (9.7) are as efficient as averaging.

9.4 Convergence Analysis

In this section, we provide a convergence analysis of the proposed method (9.6) when linear function approximation is used. Note that the TD-learning-based policy evaluation with nonlinear function approximations may fail to converge [202]. As a result, the convergence of actor-critic algorithms with nonlinear function approximations is not clear. However, the proposed method (9.7) is applicable to nonlinear function approximators including neural networks, as we demonstrated in our evaluation.

The standard practice in proving the convergence of actor-critic algorithms is to use the two-time-scale stochastic approximation approach [203]. In this approach, first the convergence of the critic step is analyzed on a faster time scale, where the policy is assumed to be fixed. Then the convergence of the policy parameters is analyzed, upon the convergence of the critic step. Following this idea, we show the convergence of the overall algorithm by proving the convergence of the critic step and the actor step separately. Before diving into the convergence analysis, there are a few standard assumptions in the literature for analyzing the convergence of actor-critic methods, as listed below.

Assumption 9.3. *The transition matrix of the Markov chain $\{s_t\}_{t\geq 0}$ induced by policy π_{θ_k}, i.e., $P^{\theta_k}(s'|s) = \sum_{a\in\mathcal{A}} \pi_{\theta_k}(s,a)\cdot P^k(s'|s,a), \forall s,s'\in\mathcal{S}$, is irreducible and aperiodic under any π_{θ_k} for $k\in[N]$. Further, $\pi_{\theta_k}(s,a)>0$ for any $s\in\mathcal{S}$, $a\in\mathcal{A}$, θ_k, and $\pi_{\theta_k}(s,a)$ is continuously differentiable with respect to θ_k.*

Assumption 9.4. *The instantaneous reward r_t^k is uniformly bounded for any $k\in[N]$ and $t\geq 0$.*

Assumption 9.5. *For every normal agent $k\in[N]$, the stepsizes $\beta_{w,t}^k$ and $\beta_{\theta,t}^k$ satisfy*

$$\sum_t \beta_{w,t}^k = \sum_t \beta_{\theta,t}^k = \infty, \quad \sum_t {\beta_{w,t}^k}^2 + {\beta_{\theta,t}^k}^2 < \infty, \quad \frac{\beta_{\theta,t}^k}{\beta_{w,t}^k}\to 0, \quad \lim_{t\to\infty} \beta_{w,t+1}^k\cdot {\beta_{w,t}^k}^{-1} = 1.$$

Assumption 9.6. *Through the whole history of the algorithm, θ_t^k belongs to a compact set for all k and t. And this compact set contains at least one local optimum of the problem* (9.2)*.*

Let $\chi_k(\theta)$ denote a Lipschitz continous function that maps the policy parameter θ to the optimal value function parameter by means of the policy evaluation algorithm.

Lemma 9.2. (In [204]). *Under Assumption 9.1, 9.3-9.5, let policy parameter θ_k at agent k be fixed, with $\{w_t^k\}$ generated from* (9.3) *using identity weight matrix $C_t = I_N$ to evaluate this policy, i.e., the non-cooperative case. Then, w_k converges to $\chi_k(\theta)$ almost surely (a.s.).*

Lemma 9.2 establishes that each agent is able to converge to the optimal value function without the model consensus step. Next, we show that with the consensus step, agents are still able to converge to the optimum point. We first provide the following intermediate results.

Lemma 9.3. *Given non-negative values $x_1, \ldots, x_n$, and weight parameters $c_1, \ldots, c_n$, if $c_i = \frac{{x_i}^{-1}}{\sum_{p=1,\ldots,n} {x_p}^{-1}}$ for $i = 1, \ldots, n$, then $\sum_{i=1,\ldots,n} c_i x_i \leq \frac{1}{n} \sum_{i=1,\ldots,n} x_i$.*

Given Lemma 9.3, we are able to show that the gradient norm will be reduced as a result of aggregation. The idea is that since a smaller gradient norm means the model is closer to the optimum point, the aggregated model becomes a better estimation than the model before the aggregation. Since without cooperation, the gradient norm will converge to zero, the gradient with cooperation will also converge to zero and therefore the estimation will converge to the optimum point. We establish the convergence of the critic step as follows.

Theorem 9.1. *Under Assumption 9.1, 9.3-9.5, for any policy π_θ, with $\{w_t^k\}$ generated from* (9.3) *using the proposed weights in* (9.6)*, we have $\lim_{t\to\infty} w_t^k = \chi_k(\theta)$ almost surely for any $k \in [N]$.*

Next, we analyze the convergence of the actor step upon the convergence of the action-value function. First, define the set $\Lambda_k = \{\theta_k : \mathbb{E}_{s_t \sim d_{\theta_k}, a_t \sim \pi_{\theta_k}} [G_{k,t}^{\theta}(\theta_k))] = 0\}$.

Lemma 9.4. *(In [125]) Under Assumption 9.1, 9.3-9.6, with $\{\theta_t^k\}$ generated from* (9.4) *using identity weight matrix $B_t = I_N$, i.e., the non-cooperative case, θ_t^k converges almost surely to a point in the set Λ_k for any $k \in \mathcal{N}$.*

Note that converging to a stationary point in the set Λ_k is the best one can achieve in the actor step [125]. Next, similar to the critic updates, we show that the aggregation step results in the gradient norm smaller than that before the aggregation. The convergence of the actor-step is established below.

Theorem 9.2. *Under Assumption 9.1-9.6, with $\{\theta_t^k\}$ generated from* (9.4) *using the proposed weights in* (9.6)*, we have θ_t^k converges almost surely to a point in the set Λ_k for any $k \in [N]$.*

9.5 Evaluation

In this section, we evaluate the proposed adaptive learning method in (9.7) for three state-of-the-art off-policy actor-critic algorithms – DDPG [205], TD3 [206], and SAC [207] – using the suite of MuJoCo continuous control tasks [208] via OpenAI Gym interface [197], including HalfCheetah, Walker2d, Ant, and Reacher. The goal of the evaluation is to understand how the proposed cooperative learning scheme can improve the overall learning performance over the network than the non-cooperative case under different conditions, including the multi-task network and the attacked scenario. We consider a network of $N = 8$ agents. The average degree of the connectivity graph is approximately $\frac{1}{N} \sum_{k=1}^{N} n_k = 5.75$. We compare our method to the baselines of 1) no cooperation is placed throughout the simulation, 2) aggregation by averaging, and 3) aggregation by median. We also consider three scenarios where the networked agents are performing

1) the same task, 2) multi-tasks, and 3) the same task when some of the agents are under attack. We find that the proposed method helps improve the overall learning performance over the network in all the scenarios. In particular, in the case when any of the agents in the network cannot learn a good policy alone, cooperation using the proposed method helps networked agents learn a better policy than the best policy any of the agents can achieve by itself. In contrast, cooperation using the average and median-based aggregation may lead to worse average learning performance over the network when agents are performing multiple tasks or some of the agents are under attack (i.e., learning from untrusted sources).

Table 9.1: Max average return for TD3.

Task	Noncoop	Average	Median	AdaLearn(Ours)
HalfCheetah	6706.49±774.00	**10350.08±82.14**	9626.16±80.44	**10249.95±84.18**
Walker2d	4180.99±463.20	4988.75±41.84	4507.01±86.61	**5835.56±213.38**
Ant	1490.81 ±671.22	**3588.90 ±78.99**	2964.49 ±70.53	**3531.44 ±76.57**
Reacher	-6.31±0.24	**-5.85±0.03**	**-5.87±0.06**	**-5.95±0.06**
Multitask	4507.75±1289.29	3314.79±2336.07	2901.95±1923.04	**6708.19±2184.79**
HalfCheetah (25% fgsm)	6596.01 ±622.46	348.31 ±191.54	9283.08 ±69.68	**9771.80 ±63.28**
HalfCheetah (50% fgsm)	6467.72 ±704.65	510.94 ±47.32	3873.77 ±117.46	**9488.09 ±192.31**
HalfCheetah (only 1 normal)	**5963.01 ±0.00**	608.72 ±0.00	-2.50 ±0.00	**5891.17 ±0.00**
Waler2d (25% fgsm)	2648.39 ±803.81	415.75 ±300.81	4348.52 ±54.55	**4398.11 ±545.71**
Waler2d (50% fgsm)	2906.32 ±1596.26	157.65 ±19.40	1115.38 ±47.98	**5345.96 ±163.44**
Waler2d (only 1 normal)	1467.60 ±0.00	-16.81 ±0.00	-9.35 ±0.00	**4458.48 ±0.00**
Ant (25% fgsm)	2433.60 ±1440.88	990.60 ±5.12	3037.39 ±56.06	**3690.67 ±16.59**
Ant (50% fgsm)	1414.63 ±861.81	982.06 ±3.37	2104.45 ±49.55	**3654.57 ±147.10**
Ant (only 1 normal)	2488.00 ±0.00	878.62 ±0.00	952.24 ±0.00	**3085.57 ±0.00**
Reacher (25% fgsm)	-6.42 ±0.38	-12.92 ±1.51	-6.36 ±0.08	**-6.10 ±0.08**
Reacher (50% fgsm)	-6.52 ±0.12	-14.71 ±2.82	-11.89 ±0.52	**-6.14 ±0.16**
Reacher (only 1 normal)	**-6.85 ±0.00**	-62.43 ±0.00	-25.73 ±0.00	**-7.17 ±0.00**

Table 9.2: Max average return for DDPG.

Task	Noncoop	Average	Median	AdaLearn(Ours)
HalfCheetah	4728.48 ±432.39	6968.39 ±65.85	6637.61 ±69.86	**7734.40 ±97.89**
Walker2d	1598.72 ±437.28	**3278.27 ±184.70**	1041.99 ±122.27	**3219.94 ±189.01**
Reacher	-6.09 ±0.18	**-5.86 ±0.02**	-6.19 ±0.11	**-5.87 ±0.02**
Multitask	3302.70 ±1779.20	2933.86 ±1597.24	1058.48 ±346.85	**5365.06 ±2538.90**
HalfCheetah (25% attacked)	4899.95 ±258.43	-52.64 ±29.20	6045.27 ±49.68	**7668.68 ±836.77**
HalfCheetah (50% attacked)	4876.08 ±184.65	-71.81 ±43.45	1939.30 ±90.08	**6361.74 ±727.24**
HalfCheetah (only 1 normal)	**4742.29 ±0.00**	375.79 ±0.00	-47.44 ±0.00	**4353.34 ±0.00**
Waler2d (25% attacked)	1233.25 ±174.05	319.87 ±78.40	1542.28 ±278.48	**2647.08 ±236.87**
Waler2d (50% attacked)	1774.96 ±506.91	-7.49 ±4.30	1123.09 ±33.55	**2422.36 ±757.20**
Waler2d (only 1 normal)	1237.56 ±0.00	-10.31 ±0.00	-15.30 ±0.00	**1644.62 ±0.00**
Reacher (25% attacked)	-6.18 ±0.22	-16.99 ±2.31	-6.07 ±0.06	**-5.92 ±0.16**
Reacher (50% attacked)	**-6.24 ±0.24**	-102.32 ±6.59	-9.12 ±0.20	**-6.62 ±0.17**
Reacher (only 1 normal)	**-6.03 ±0.0**	-109.00 ±0.00	-13.45 ±0.00	**-6.10 ±0.00**

In all the tasks, we use ADAM optimizer [199], $\gamma = 0.99$, learning rate is 0.001, $\tau = 0.005$, batch-size is 256. The other hyper-parameters and neural network architectures are the same as given in the original algorithms [205–207]. The simulation code is based on the author's publication in https://github.com/sfujim/TD3 and https://github.com/haarnoja/sac.

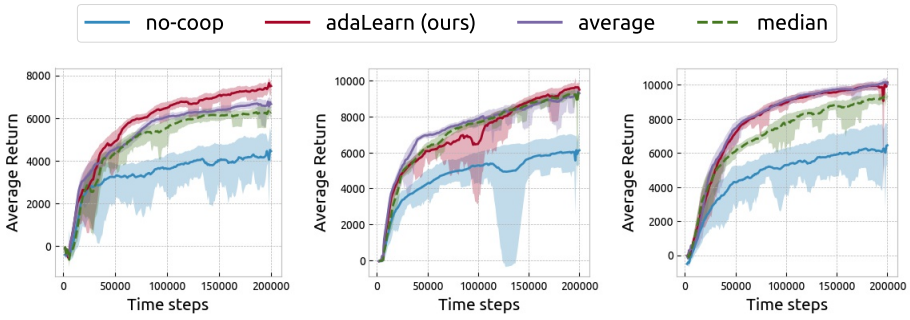
Table 9.3: Max average return for SAC.

Task	Noncoop	Average	Median	AdaLearn(Ours)
HalfCheetah	6399.85 ±673.81	9614.02 ±107.70	9588.57 ±79.58	**9931.12 ±125.80**
Walker2d	3019.59 ±718.71	**3662.94 ±60.70**	3289.44 ±93.99	3398.29 ±96.57
Ant	1979.03 ±348.19	3720.35 ±62.03	3146.27 ±67.65	**3776.56 ±64.48**
Reacher	-6.33 ±0.29	-6.50 ±0.07	-6.50 ±0.12	**-6.20 ±0.13**
Multitask	3938.91 ±1632.00	3973.77 ±1337.77	2958.70 ±1853.09	**4857.24 ±3433.90**
HalfCheetah (25% attacked)	6163.24 ±823.36	261.09 ±157.26	8616.53 ±89.98	**8742.69 ±63.35**
HalfCheetah (50% attacked)	5755.53 ±659.42	1003.12 ±349.58	3725.91 ±82.78	**7885.69 ±39.13**
HalfCheetah (only 1 normal)	**6710.47 ±0.00**	-5.97 ±0.00	668.58 ±0.00	**6666.83 ±0.00**
Waler2d (25% attacked)	3396.36 ±440.52	315.70 ±62.68	2781.80 ±63.98	**3531.32 ±36.73**
Waler2d (50% attacked)	**3132.66 ±812.49**	3.16 ±4.63	959.56 ±10.18	**3115.22 ±47.73**
Waler2d (only 1 normal)	2284.98 ±0.00	1.25 ±0.00	-6.22 ±0.00	**2695.72 ±0.00**
Ant (25% attacked)	2198.53 ±254.48	994.86 ±0.91	3022.71 ±71.98	**3578.94 ±54.02**
Ant (50% attacked)	1803.09 ±258.45	999.50 ±1.15	1114.05 ±56.17	**3483.97 ±70.63**
Ant (only 1 normal)	1383.55 ±0.00	692.08 ±0.00	576.88 ±0.00	**1750.23 ±0.00**
Reacher (25% attacked)	-6.60 ±0.72	-14.09 ±1.10	-6.22 ±0.05	**-6.17 ±0.02**
Reacher (50% attacked)	**-6.35 ±0.21**	-31.41 ±13.13	**-6.47 ±0.09**	**-6.50 ±0.26**
Reacher (only 1 normal)	**-6.24 ±0.00**	-70.99 ±0.00	-101.27 ±0.00	**-6.59 ±0.00**

Agents learn in independent environments and they start with random environment seeds in OpenAI Gym. Agents exchange and aggregate the model (actor/critic) parameters with neighbors after each learning episode. If the algorithm includes target networks, then the target networks are updated using soft update with $\tau = 0.001$ [205]. The learning results for TD3, DDPG and SAC are given in Table 9.1, 9.2 and 9.3, where maximum value for each task is bolded, and $\pm$ corresponds to a single standard deviation over the network. In the tables, the results are when the agents are trained for 10^5 steps for the tasks Reacher and the attacked Reacher, and 2×10^5 steps for the other tasks. Note that DDPG does not converge for the task Ant and is not listed in Table 9.2. Training curves are given in Figure 9.1-Figure 9.5. Note that in the figures, values are computed as an average of the 10 nearest values to make the curves smooth. Also, to make the discrepancies between normal and attacked agents larger, we consider an enlarged reward for the task Reacher in the training steps (not in the evaluation), that is, the new reward is $10\times$ the original reward given in OpenAI Gym.

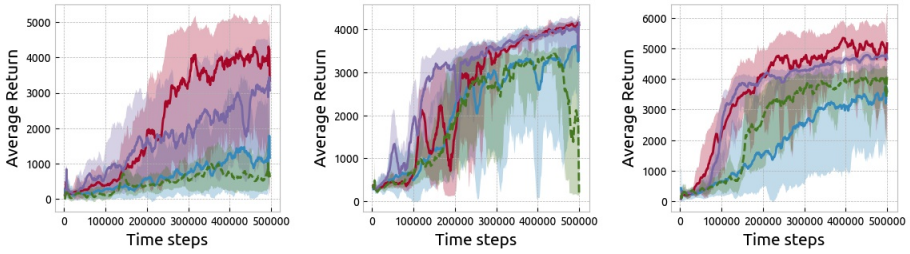
Single-task network. In this case, all the agents in the network perform the same task. The results are given in Figure 9.1 and Figure 9.2, where the lines represent the average return over the networked agents and the shaded area represents the range. We find the proposed method greatly improves the learning performance over the network as measured by the average return compared to the non-cooperative case. It also outperforms median. Besides, the performance of the proposed method matches that of the average-based aggregation. In certain examples, the proposed method exhibits better performance than averaging.

Multi-task network. In the Multi-Task Learning (MTL) example, 8 agents perform two different tasks – HalfCheetah (agents 0-3) and Walker2d (agents 4-7). The MTL problem has been extensively studied in distributed multi-agent networks [182], which is a powerful tool to model the case where agents share distinct



(a) DDPG (b) SAC (c) TD3

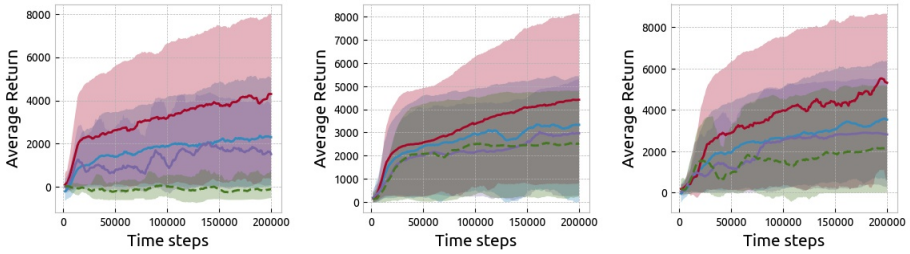
Figure 9.1: Training curves for DDPG, SAC, and TD3 (HalfCheetah).



(a) DDPG (b) SAC (c) TD3

Figure 9.2: Training curves for DDPG, SAC, and TD3 (Walker2d).

and related tasks, e.g., in a network of users for a certain mobile application. Averaging and median are not suitable for MTL. However, as the proposed method promotes the similarity among agents, it can be used for such tasks. Figure 9.3 shows the learning results in the MTL network. It can be observed that the proposed method greatly improves the performance, whereas averaging and median result in worse performance than the non-cooperative case. The results demonstrate that the proposed method effectively promotes the similarity among the networked agents and the cooperation by promoting such similarity improves the overall learning performance.

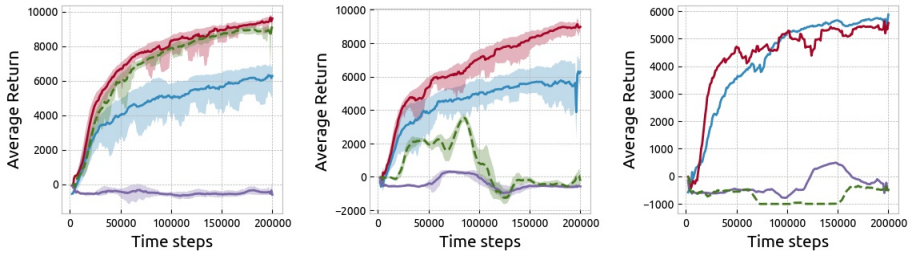


(a) DDPG (b) SAC (c) TD3

Figure 9.3: Training curves for DDPG, SAC, and TD3 (Multitask).

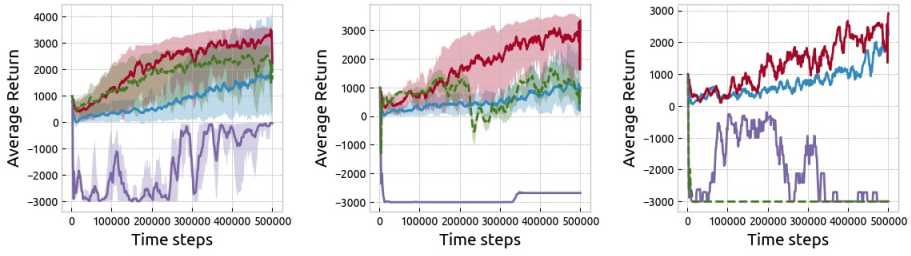
Attacked network. In this scenario, we consider the case where some of the networked agents are under

attack, yet still continue to exchange parameters with the neighbors. Cooperation with such attacked agents will cause deterioration in the learning performance, as the attacked agents are indistinguishable from the normal agents. Average and median based model aggregation are vulnerable to such attacks. However, using the proposed method, attacked agents will be assigned smaller weights than the normal ones given the discrepancies in their estimates, and thus ensuring the integrity of the cooperation. We consider that the attacked agents are receiving adversarial states by adding perturbation to their states using FGSM (Fast Gradient Sign Method) [209]. The multiplier of the perturbation is selected to be 0.005. Figure 9.4 and Figure 9.5 show the results when 2 (25%), 4 (50%), and 7 (only 1 normal) agents are under attack. It can be found that averaging results in inferior performance than non-cooperation in all the attacked cases. In addition, when the number of attacked agents is less than half of the total number of agents, median is robust to such attack, but it fails when the number of attacked agents increases beyond that. In contrast, the proposed method achieves better performance even in the case of increased number of attacked agents. Note that when 7 out of 8 agents are under attack, there is only one normal agent in the network and cooperation using the proposed method results in a similar performance to the non-cooperative case, showing the resilience of the proposed method.



(a) 2 agents under attack (b) 4 agents under attack (c) 7 agents under attack

Figure 9.4: Training curves for normal agents when certain agents are under FGSM attack (TD3, HalfCheetah).



(a) 2 agents under attack (b) 4 agents under attack (c) 7 agents under attack

Figure 9.5: Training curves for normal agents when certain agents are under FGSM attack (TD3, Ant).

9.6 Conclusion

In this chapter, we study the problem of distributed reinforcement learning using actor-critic algorithms in a fully decentralized network. Networked agents make individual decisions and combine the models from their neighbors for a better learning performance. We propose an efficient cooperation strategy that adaptively assigns weights to neighbors by measuring the similarities among agents and demonstrate the convergence of the actor-critic algorithms using the proposed method when linear function approximations are applied. Further, extensive simulation results are presented showing that cooperation by promoting the similarity among agents improves the learning performance over the network for actor-critic algorithms, even in the case where agents are performing different tasks and when some of the agents are under attack (i.e, learning from untrusted resources).

9.A Proofs

9.A.1 Proof for Lemma 9.1

Proof. Given Assumption 9.1,

$$\begin{aligned} G^w_{k,t}(w) &= \left(r^k_{t+1} - \mu^k_t + Q^k_{t+1}(w) - Q^k_t(w)\right) \cdot \phi^k_w(s_t, a_t) \\ &= \left(r^k_{t+1} - \mu^k_t + w^\top(\phi^k_w(s_{t+1}, a_{t+1}) - \phi^k_w(s_t, a_t)\right) \cdot \phi^k_w(s_t, a_t) \\ &= \left(r^k_{t+1} - \mu^k_t\right) \cdot \phi^k_w(s_t, a_t) + w^\top \left[\left(\phi^k_w(s_{t+1}, a_{t+1}) - \phi^k_w(s_t, a_t)\right) \cdot \phi^k_w(s_t, a_t)\right]. \end{aligned}$$

Define γ^k_t and ϕ^k_t as

$$\gamma^k_t \triangleq \left(r^k_{t+1} - \mu^k_t\right) \cdot \phi^k_w(s_t, a_t), \quad \phi^k_t \triangleq \left(\phi^k_w(s_{t+1}, a_{t+1}) - \phi^k_w(s_t, a_t)\right) \cdot \phi^k_w(s_t, a_t).$$

We can write $G^w_{k,t}(w)$ as

$$G^w_{k,t}(w) = \gamma^k_t + w^\top \phi^k_t. \tag{8}$$

Given Assumption 9.2, the derivation becomes $\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \theta^\top \phi^k_\pi(s))\phi^k_\pi(s)}{\sigma^2_k}$. Since the policy function is updated in a slower time scale in actor-critic algorithms, we can consider $Q^k_t(w^k_t)$ as a fixed value. Hence,

$$\begin{aligned} G^\theta_{k,t}(\theta) &= Q^k_t(w^k_t) \cdot \nabla_\theta \log \pi_\theta(s_t, a_t) \\ &= Q^k_t(w^k_t) \cdot \frac{(a_t - \theta^\top \phi^k_\pi(s_t))\phi^k_\pi(s_t)}{\sigma^2_k} \\ &= a_t \phi^k_\pi(s_t) \cdot \frac{Q^k_t(w^k_t)}{\sigma^2_k} - \theta^\top \phi^k_\pi(s_t)\phi^k_\pi(s_t) \cdot \frac{Q^k_t(w^k_t)}{\sigma^2_k} \end{aligned}$$

Define ξ^k_t and ζ^k_t as

$$\xi^k_t \triangleq a_t \phi^k_\pi(s_t) \cdot \frac{Q^k_t(w^k_t)}{\sigma^2_k}, \quad \zeta^k_t = \phi^k_\pi(s_t)\phi^k_\pi(s_t) \cdot \frac{Q^k_t(w^k_t)}{\sigma^2_k}.$$

We can write $G^\theta_{k,t}(\theta)$ as

$$G^\theta_{k,t}(\theta) = \xi^k_t - \theta^\top \zeta^k_t. \tag{9}$$

□

9.A.2 Proof for Lemma 9.3

Proof. To prove

$$\sum_{i=1,\ldots,n} \frac{{x_t}^{-1}}{\sum_{p=1,\ldots,n} {x_p}^{-1}} x_t \leq \frac{1}{n} \sum_{i=1,\ldots,n} x_t$$

It is equivalent to prove that

$$\frac{n}{\sum_{p=1,\ldots,n} {x_p}^{-1}} \leq \frac{1}{n} \sum_{i=1,\ldots,n} x_t, \text{ or } \sum_{p=1,\ldots,n} {x_p}^{-1} \sum_{i=1,\ldots,n} x_t \geq n^2. \tag{10}$$

We prove (10) by induction. Consider the base case where n = 2. It holds that

$$(\frac{1}{x_1} + \frac{1}{x_2})(x_1 + x_2) = 1 + \frac{x_2}{x_1} + \frac{x_1}{x_2} + 1 \geq 4 = 2^2.$$

Assume that for some $n = k \geq 2$, (10) holds, i.e.,

$$(\frac{1}{x_1} + \ldots + \frac{1}{x_k})(x_1 + \ldots + x_k) \geq k^2$$

Then, when $n = k + 1$, it follows that:

$$\begin{aligned}
&(\frac{1}{x_1} + \ldots + \frac{1}{x_{k+1}})(x_1 + \ldots + x_{k+1}) \\
=&(\frac{1}{x_1} + \ldots + \frac{1}{x_k})(x_1 + \ldots + x_k) + x_{k+1}(\frac{1}{x_1} + \ldots + \frac{1}{x_k}) + \frac{1}{x_{k+1}}(x_1 + \ldots + x_k) + 1 \\
\geq&(\frac{1}{x_1} + \ldots + \frac{1}{x_k})(x_1 + \ldots + x_k) + 2\sqrt{(\frac{1}{x_1} + \ldots + \frac{1}{x_k})(x_1 + \ldots + x_k)} + 1 \\
\geq&k^2 + 2k + 1 = (k+1)^2
\end{aligned}$$

That is, the statement also holds true for $n = k + 1$, establishing the inductive step, which completes the proof. □

9.A.3 Proof for Theorem 9.1

Proof. Given Lemma 9.1, we can write $G^w_{k,t}(w)$ as

$$G^w_{k,t}(w) = \gamma^k_t + w^\top \phi^k_t,$$

where

$$\gamma^k_t \triangleq \left(r^k_{t+1} - \mu^k_t\right) \cdot \phi^k_w(s_t, a_t), \quad \phi^k_t \triangleq \left(\phi^k_w(s_{t+1}, a_{t+1}) - \phi^k_w(s_t, a_t)\right) \cdot \phi^k_w(s_t, a_t).$$

Given the combination step $w_{t+1}^k = \sum_{l\in\mathcal{N}_k} c_t(k,l)\cdot \tilde{w}_t^l$, it follows that

$$\gamma_t^k + {w_{t+1}^k}^\top \phi_t^k = \gamma_t^k + \sum_{l\in\mathcal{N}_k} c_t(k,l)\cdot \tilde{w}_t^{l\top}\phi_t^k = \sum_{l\in\mathcal{N}_k} c_t(k,l)\cdot\left(\gamma_t^k + \tilde{w}_t^{l\top}\phi_t^k\right).$$

Thus, given (8), we have

$$G_{k,t}^w(w_{t+1}^k) = \sum_{l\in\mathcal{N}_k} c_t(k,l)\cdot G_{k,t}^w(\tilde{w}_t^l),$$

Hence,

$$\|G_{k,t}^w(w_{t+1}^k)\| \le \sum_{l\in\mathcal{N}_k} c_t(k,l)\cdot \|G_{k,t}^w(\tilde{w}_t^l)\|.$$

Put weights (9.6) into the above formula and given Lemma 9.3, it yields that

$$\|G_{k,t}^w(w_{t+1}^k)\| \le \frac{1}{|\mathcal{N}_k^{\le}|}\sum_{l\in\mathcal{N}_k^{\le}} \|G_{k,t}^w(\tilde{w}_t^l)\| \le \|G_{k,t}^w(\tilde{w}_t^k)\| \tag{11}$$

Using the proposed weights (9.6), (11) holds. It can be found that $\|G_{k,t}^w(w_{t+1}^k)\|$ is bounded by $\|G_{k,t}^w(\tilde{w}_t^k)\|$ at each step. Given Lemma 9.2, $\lim_{t\to\infty} w_t^k = \chi_k(\theta)$ a.s., for all k, without cooperation, i.e, when $w_{t+1}^k = \tilde{w}_t^k$ and $G_{k,t}^w(w_{t+1}^k) = G_{k,t}^w(\tilde{w}_t^k)$. As a result, $\lim_{t\to\infty}\|G_{k,t}(\tilde{w}_k^t)\| = 0$ a.s. Combining with (11), we conclude that with $\{w_t^k\}$ generated from (9.3) using the proposed weights (9.6), $\lim_{t\to\infty}\|G_{k,t}(w_t^k)\| = 0$ a.s. Thus, $\lim_{t\to\infty} w_t^k = \chi_k(\theta)$ a.s. □

9.A.4 Proof for Theorem 9.2

Proof. Given Lemma 9.1, we can write $G_{k,t}^\theta(\theta)$ as

$$G_{k,t}^\theta(\theta) = \xi_t^k - \theta^\top \zeta_t^k,$$

where

$$\xi_t^k \triangleq a_t\phi_\pi^k(s_t)\cdot\frac{Q_t^k(w_t^k)}{\sigma_k^2}, \quad \zeta_t^k \triangleq \phi_\pi^k(s_t)\phi_\pi^k(s_t)\cdot\frac{Q_t^k(w_t^k)}{\sigma_k^2}.$$

Given the combination step $\theta_{t+1}^k = \sum_{l\in\mathcal{N}_k} b_t(k,l)\cdot\tilde{\theta}_t^l$, it follows that

$$\xi_t^k - {\theta_t^l}^\top\zeta_t^k = \xi_t^k - \sum_{l\in\mathcal{N}_k} b_t(k,l)\cdot\tilde{\theta}_t^{l\top}\zeta_t^k = \sum_{l\in\mathcal{N}_k} b_t(k,l)\cdot\left(\xi_t^k - \tilde{\theta}_t^{l\top}\zeta_t^k\right).$$

Thus, given (9), we have

$$G^{\theta}_{k,t}(\theta^k_{t+1}) = \sum_{l \in \mathcal{N}_k} b_t(k,l) \cdot G^{\theta}_{k,t}(\tilde{\theta}^k_t),$$

Hence,

$$\|G^{\theta}_{k,t}(\theta^k_{t+1})\| \leq \sum_{l \in \mathcal{N}_k} b_t(k,l) \cdot \|G^{\theta}_{k,t}(\tilde{\theta}^k_t)\|.$$

Put weights (9.6) into the above formula and given Lemma 9.3, it yields that

$$\|G^{\theta}_{k,t}(\theta^k_{t+1})\| \leq \frac{1}{|\mathcal{N}^{\leq}_k|} \sum_{l \in \mathcal{N}^{\leq}_k} \|G^{\theta}_{k,t}(\tilde{\theta}^l_t)\| \leq \|G^{\theta}_{k,t}(\tilde{\theta}^k_t)\| \tag{12}$$

Using the proposed weights (9.6), (12) holds. It can be found that $\|G^{\theta}_{k,t}(\theta^k_{t+1})\|$ is bounded by $\|G^{\theta}_{k,t}(\tilde{\theta}^k_t)\|$ at each step. Given Lemma 9.4, θ^l_t converges to a point in the set Λ_k a.s., for all k, without cooperation, i.e, when $\theta^k_{t+1} = \tilde{\theta}^k_t$ and $G^{\theta}_{k,t}(\theta^k_{t+1}) = G^{\theta}_{k,t}(\tilde{\theta}^k_t)$. As a result, $\lim_{t\to\infty} \|G^{\theta}_{k,t}(\tilde{\theta}^k_t)\| = 0$ a.s. Combining with (12), we conclude that with $\{\theta^k_t\}$ generated from (9.4) using the proposed weights (9.6), $\lim_{t\to\infty} \|G^{k,k}_{\theta,t}\| = 0$ a.s. Thus, θ^k_t converges a.s. to a point in the set Λ_k. □

Chapter 10

Conclusion

Distributed consensus, learning, and optimization algorithms are vulnerable to cyber-attacks. A single adversarial agent exchanging malicious information with normal agents may ruin the integrity of the entire network. This work proposes multiple solutions to address the vulnerabilities of such distributed algorithms and provides guarantees for the resilient operation in multi-agent systems. Throughout the thesis, we study the resilient vector consensus problem and propose a centerpoint-based consensus method that generalizes the resilience property of the median into higher dimensions, which offers more advantages in terms of computational complexity and characterization than the existing methods. We extend this consensus method into distributed learning and optimization problems, particularly for robotic applications, where the position vector is in two or three dimensions, and a centerpoint can be computed efficiently in $O(n)$ and $O(n^2)$ time complexity. For high-dimensional distributed learning problems, we incorporate the idea of optimizing the objective function in designing the cooperation strategy, which is both efficient and guarantees the resilient cooperation among normal agents in the presence of an arbitrary number of adversarial agents, without the need of a tailored upper-bound of the adversaries. Further, we apply such an approach into distributed multi-task learning, clustering, and reinforcement learning problems. Our results are supported by theoretical analysis and well validated by numerical implementations and practical experiments. We hope our results could assist people in the design and implementation of real-world applications in distributed multi-agent systems.

Bibliography

[1] Jiani Li and Xenofon D. Koutsoukos. “Resilient Distributed Diffusion for Multi-task Estimation”. In: *14th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. 2018, pp. 93–102. DOI: 10.1109/DCOSS.2018.00020. URL: https://doi.org/10.1109/DCOSS.2018.00020.

[2] Jiani Li, Waseem Abbas, and Xenofon Koutsoukos. “Resilient Distributed Diffusion in Networks With Adversaries”. In: *IEEE Transactions on Signal and Information Processing over Networks* 6 (2020), pp. 1–17. ISSN: 2373-7778. DOI: 10.1109/TSIPN.2019.2957731.

[3] Mudassir Shabbir, Jiani Li, Waseem Abbas, and Xenofon D. Koutsoukos. “Resilient Vector Consensus in Multi-Agent Networks Using Centerpoints”. In: *2020 American Control Conference (ACC)*. 2020, pp. 4387–4392. DOI: 10.23919/ACC45564.2020.9147441.

[4] Jiani Li, Waseem Abbas, Mudassir Shabbir, and Xenofon Koutsoukos. “Resilient Distributed Diffusion for Multi-Robot Systems Using Centerpoint”. In: *Proceedings of Robotics: Science and Systems (RSS)*. Corvalis, Oregon, USA, July 2020. DOI: 10.15607/RSS.2020.XVI.021.

[5] Jiani Li, Waseem Abbas, Mudassir Shabbir, and Xenofon Koutsoukos. “Resilient Multi-Robot Target Pursuit”. In: *Hot Topics in the Science of Security Symposium (HotSoS ’20)*. 2020.

[6] Waseem Abbas, Mudassir Shabbir, Jiani Li, and Xenofon D. Koutsoukos. “Interplay Between Resilience and Accuracy in Resilient Vector Consensus in Multi-Agent Networks”. In: *59th IEEE Conference on Decision and Control, CDC 2020, Jeju Island, South Korea, December 14-18, 2020*. IEEE, 2020, pp. 3127–3132. URL: https://doi.org/10.1109/CDC42340.2020.9304098.

[7] Jiani Li, Waseem Abbas, and Xenofon D. Koutsoukos. “Byzantine Resilient Distributed Multi-Task Learning”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems (NeurIPS) 2020, December 6-12*. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/d37eb50d868361ea729bb4147eb3c1d8-Abstract.html.

[8] Feiyang Cai, Jiani Li, and X. Koutsoukos. “Detecting Adversarial Examples in Learning-Enabled Cyber-Physical Systems using Variational Autoencoder for Regression”. In: *2020 IEEE Security and Privacy Workshops (SPW)* (2020), pp. 208–214.

[9] Xingyu Zhou, Yi Li, Carlos A. Barreto, Jiani Li, Peter Volgyesi, Himanshu Neema, and Xenofon Koutsoukos. “Evaluating Resilience of Grid Load Predictions under Stealthy Adversarial Attacks”. In: *2019 Resilience Week (RWS)*. Vol. 1. 2019, pp. 206–212. DOI: 10.1109/RWS47064.2019.8971816.

[10] S. Kar and J. M. F. Moura. “Sensor Networks With Random Links: Topology Design for Distributed Consensus”. In: *IEEE Transactions on Signal Processing* 56.7 (July 2008), pp. 3315–3326. ISSN: 1053-587X. DOI: 10.1109/TSP.2008.920143.

[11] S. Chouvardas, K. Slavakis, and S. Theodoridis. “Adaptive Robust Distributed Learning in Diffusion Sensor Networks”. In: *IEEE Transactions on Signal Processing* 59.10 (Oct. 2011), pp. 4692–4707. ISSN: 1053-587X. DOI: 10.1109/TSP.2011.2161474.

[12] S. Markovich-Golan, A. Bertrand, M. Moonen, and S. Gannot. “Optimal distributed minimum-variance beamforming approaches for speech enhancement in wireless acoustic sensor networks”. In: *Signal Processing* 107 (2015), pp. 4–20.

[13] Le Xie, Dae-Hyun Choi, and Soummya Kar. “Cooperative distributed state estimation: Local observability relaxed”. In: *Power and Energy Society General Meeting, 2011 IEEE*. IEEE. 2011, pp. 1–11.

[14] Yuan Chen, Soummya Kar, and José M. F. Moura. “Resilient Distributed Estimation: Exponential Convergence Under Sensor Attacks”. In: *57th IEEE Conference on Decision and Control*. Miami, FL, December 17-19, 2018, pp. 7275–7282.

[15] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S. Talwalkar. “Federated Multi-Task Learning”. In: *Advances in Neural Information Processing Systems, Long Beach, CA, USA*. 2017, pp. 4424–4434. URL: http://papers.nips.cc/paper/7029-federated-multi-task-learning.

[16] Ali H. Sayed, Sheng-Yuan Tu, Jianshu Chen, Xiaochuan Zhao, and Zaid J. Towfic. “Diffusion Strategies for Adaptation and Learning over Networks: An Examination of Distributed Strategies and Network Behavior.” In: *IEEE Signal Processing Magazine* 30.3 (2013), pp. 155–171.

[17] Heath LeBlanc, Haotian Zhang, Xenofon D. Koutsoukos, and Shreyas Sundaram. “Resilient Asymptotic Consensus in Robust Networks”. In: *IEEE Journal on Selected Areas in Communications* 31.4 (2013), pp. 766–781.

[18] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. “Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent”. In: *Annual Conference on Neural Information Processing Systems*. 2017, pp. 118–128.

[19] F. Pasqualetti, A. Bicchi, and F. Bullo. “Consensus Computation in Unreliable Networks: A System Theoretic Approach”. In: *IEEE Transactions on Automatic Control* 57.1 (Jan. 2012), pp. 90–104. ISSN: 2334-3303. DOI: 10.1109/TAC.2011.2158130.

[20] Iman Shames, AndrÃ© M.H. Teixeira, Henrik Sandberg, and Karl H. Johansson. “Distributed fault detection for interconnected second-order systems”. In: *Automatica* 47.12 (2011), pp. 2757 –2764.

[21] Yuan Chen, Soummya Kar, and José M. F. Moura. “Adversary Detection and Resilient Distributed Estimation of Parameters from Compact Sets”. In: *IEEE Transactions on Signal Processing*. 2016. URL: https://arxiv.org/abs/1701.00878.

[22] Nitin H. Vaidya and Vijay K. Garg. “Byzantine Vector Consensus in Complete Graphs”. In: *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2013, pp. 65–73.

[23] Hammurabi Mendes, Maurice Herlihy, Nitin H. Vaidya, and Vijay K. Garg. “Multidimensional agreement in Byzantine systems”. In: *Distributed Comput.* 28.6 (2015), pp. 423–441.

[24] Xavier Défago, Maria Gradinariu, Stéphane Messika, and Philippe Raipin-Parvédy. “Fault-Tolerant and Self-stabilizing Mobile Robots Gathering”. In: *Distributed Computing*. Ed. by Shlomi Dolev. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 46–60. ISBN: 978-3-540-44627-9.

[25] S. Sundaram and C. N. Hadjicostis. “Distributed Function Calculation via Linear Iterative Strategies in the Presence of Malicious Agents”. In: *IEEE Transactions on Automatic Control* 56.7 (July 2011), pp. 1495–1508. ISSN: 2334-3303. DOI: 10.1109/TAC.2010.2088690.

[26] Nitin H. Vaidya, Lewis Tseng, and Guanfeng Liang. “Iterative approximate byzantine consensus in arbitrary directed graphs”. In: *PODC ’12*. 2012.

[27] Lewis Tseng and Nitin Vaidya. “Iterative approximate byzantine consensus under a generalized fault model”. In: *International Conference on Distributed Computing and Networking*. Springer. 2013, pp. 72–86.

[28] Hammurabi Mendes and Maurice Herlihy. “Multidimensional approximate agreement in Byzantine asynchronous systems”. In: *45th Annual ACM Symposium on Theory of Computing*. 2013, pp. 391–400.

[29] El Mahdi El Mhamdi, Rachid Guerraoui, Arsany Guirguis, and Sebastien Rouault. “SGD: Decentralized Byzantine Resilience”. In: *CoRR* abs/1905.03853 (2019). arXiv: 1905.03853. URL: http://arxiv.org/abs/1905.03853.

[30] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. “Reaching approximate agreement in the presence of faults”. In: *J. ACM* 33.3 (1986), pp. 499–516. DOI: 10.1145/5925.5931. URL: https://doi.org/10.1145/5925.5931.

[31] H. Zhang and S. Sundaram. “A simple median-based resilient consensus algorithm”. In: *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. Oct. 2012, pp. 1734–1741. DOI: 10.1109/Allerton.2012.6483431.

[32] Hyongju Park and Seth A Hutchinson. “Fault-tolerant rendezvous of multirobot systems”. In: *IEEE Transactions on Robotics* 33 (2017), pp. 565–582.

[33] Nitin H Vaidya. “Iterative byzantine vector consensus in incomplete graphs”. In: *International Conference on Distributed Computing and Networking*. Springer. 2014, pp. 14–28.

[34] Lili Su and Nitin H Vaidya. “Fault-tolerant multi-agent optimization: optimal iterative distributed algorithms”. In: *ACM Symposium on Principles of Distributed Computing*. 2016, pp. 425–434.

[35] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. “Randomized gossip algorithms”. In: *IEEE Transactions on Information Theory* 52.6 (June 2006), pp. 2508–2530. ISSN: 1557-9654. DOI: 10.1109/TIT.2006.874516.

[36] Waseem Abbas, Aron Laszka, and Xenofon Koutsoukos. “Improving network connectivity and robustness using trusted nodes with application to resilient consensus”. In: *IEEE Transactions on Control of Network Systems* 5.4 (2018), pp. 2036–2048.

[37] Faiq Ghawash and Waseem Abbas. “Leveraging Diversity for Achieving Resilient Consensus in Sparse Networks”. In: *8th IFAC Workshop on Distributed Estimation and Control in Networked Systems (NecSys)*. 2019.

[38] David Saldana, Amanda Prorok, Mario FM Campos, and Vijay Kumar. “Triangular Networks for Resilient Formations”. In: *13th International Symposium on Distributed Autonomous Robotic Systems*. 2016.

[39] A. Pilloni, A. Pisano, M. Franceschelli, and E. Usai. “Robust distributed consensus on the median value for networks of heterogeneously perturbed agents”. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. Dec. 2016, pp. 6952–6957. DOI: 10.1109/CDC.2016.7799340.

[40] Mauro Franceschelli, Alessandro Giua, and Alessandro Pisano. “Finite-Time Consensus on the Median Value With Robustness Properties”. In: *IEEE Transactions on Automatic Control* 62 (2017), pp. 1652–1667.

[41] Luis Guerrero-Bonilla, Amanda Prorok, and Vijay Kumar. “Formations for Resilient Robot Teams”. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 841–848.

[42] Kelsey Saulnier, David Saldana, Amanda Prorok, George J. Pappas, and Vijay Kumar. “Resilient Flocking for Mobile Robot Teams”. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 1039–1046. DOI: 10.1109/LRA.2017.2655142.

[43] Wolfgang Mulzer and Daniel Werner. “Approximating Tverberg Points in Linear Time for Any Fixed Dimension”. In: *CoRR* abs/1107.0104 (2011). arXiv: 1107.0104. URL: http://arxiv.org/abs/1107.0104.

[44] Xuan Wang, Shaoshuai Mou, and Shreyas Sundaram. “A resilient convex combination for consensus-based distributed algorithms”. In: *Numerical Algebra, Control & Optimization* 9 (2019), p. 269.

[45] S. M. Dibaji, H. Ishii, and R. Tempo. “Resilient randomized quantized consensus”. In: *2016 American Control Conference (ACC)*. July 2016, pp. 5118–5123. DOI: 10.1109/ACC.2016.7526165.

[46] Marc Casas, Wilfried N Gansterer, and Elias Wimmer. “Resilient gossip-inspired all-reduce algorithms for high-performance computing: Potential, limitations, and open questions”. In: *The International Journal of High Performance Computing Applications* 33.2 (2019), pp. 366–383. DOI: 10.1177/1094342018762531. eprint: https://doi.org/10.1177/1094342018762531. URL: https://doi.org/10.1177/1094342018762531.

[47] Wilfried N. Gansterer, Gerhard Niederbrucker, Hana StrakovÃ¡, and Stefan Schulze Grotthoff. “Scalable and fault tolerant orthogonalization based on randomized distributed data aggregation”. In: *Journal of Computational Science* 4.6 (2013). Scalable Algorithms for Large-Scale Systems Workshop (ScalA2011), Supercomputing 2011, pp. 480 –488. ISSN: 1877-7503. DOI: https://doi.org/10.1016/j.jocs.2013.01.006. URL: http://www.sciencedirect.com/science/article/pii/S1877750313000185.

[48] Jakub Konecný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. “Federated Learning: Strategies for Improving Communication Efficiency”. In: *CoRR* abs/1610.05492 (2016). arXiv: 1610.05492. URL: http://arxiv.org/abs/1610.05492.

[49] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. “A Public Domain Dataset for Human Activity Recognition using Smartphones”. In: *21st European Symposium on Artificial Neural Networks, ESANN 2013, Bruges, Belgium, April 24-26, 2013*. 2013. URL: http://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2013-84.pdf.

[50] Kai Yang, Tao Jiang, Yuanming Shi, and Zhi Ding. “Federated Learning via Over-the-Air Computation”. In: *CoRR* abs/1812.11750 (2018). arXiv: 1812.11750. URL: http://arxiv.org/abs/1812.11750.

[51] Yiqiang Chen, Jindong Wang, Chaohui Yu, Wen Gao, and Xin Qin. “FedHealth: A Federated Transfer Learning Framework for Wearable Healthcare”. In: *CoRR* abs/1907.09173 (2019). arXiv: 1907.09173. URL: http://arxiv.org/abs/1907.09173.

[52] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter L. Bartlett. “Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018, pp. 5636–5645. URL: http://proceedings.mlr.press/v80/yin18a.html.

[53] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. “Generalized Byzantine-tolerant SGD”. In: *CoRR* abs/1802.10116 (2018). arXiv: 1802.10116. URL: http://arxiv.org/abs/1802.10116.

[54] Cong Xie, Sanmi Koyejo, and Indranil Gupta. “Zeno: Distributed Stochastic Gradient Descent with Suspicion-based Fault-tolerance”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 2019, pp. 6893–6901. URL: http://proceedings.mlr.press/v97/xie19b.html.

[55] Nikola Konstantinov and Christoph Lampert. “Robust Learning from Untrusted Sources”. In: *ICML*. 2019.

[56] Changjian Shui, Mahdieh Abbasi, Louis-Émile Robitaille, Boyu Wang, and Christian Gagné. “A Principled Approach for Learning Task Similarity in Multitask Learning”. In: *IJCAI*. 2019.

[57] Zhixiong Yang and Waheed U. Bajwa. “ByRDiE: Byzantine-Resilient Distributed Coordinate Descent for Decentralized Learning”. In: *IEEE Trans. Signal and Information Processing over Networks* 5.4 (2019), pp. 611–627. DOI: 10.1109/TSIPN.2019.2928176. URL: https://doi.org/10.1109/TSIPN.2019.2928176.

[58] Venkata Krishna Pillutla, Sham M. Kakade, and Zaïd Harchaoui. “Robust Aggregation for Federated Learning”. In: *CoRR* abs/1912.13445 (2019). arXiv: 1912.13445.

[59] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. “Clustered Federated Learning: Model-Agnostic Distributed Multi-Task Optimization under Privacy Constraints”. In: *ArXiv* abs/1910.01991 (2019).

[60] X. Zhao and A. H. Sayed. “Clustering via diffusion adaptation over networks”. In: *2012 3rd International Workshop on Cognitive Information Processing*. May 2012, pp. 1–6. DOI: 10.1109/CIP.2012.6232902.

[61] Anusha Lalitha, Osman Cihan Kilinc, Tara Javidi, and Farinaz Koushanfar. “Peer-to-peer Federated Learning on Graphs”. In: *ArXiv* abs/1901.11173 (2019).

[62] Yudong Chen, Lili Su, and Jiaming Xu. “Distributed Statistical Machine Learning in Adversarial Settings: Byzantine Gradient Descent”. In: *Proc. ACM Meas. Anal. Comput. Syst.* 1.2 (Dec. 2017), 44:1–44:25. ISSN: 2476-1249. DOI: 10.1145/3154503. URL: http://doi.acm.org/10.1145/3154503.

[63] Haibo Yang, Xin Zhang, Minghong Fang, and Jia Liu. “Byzantine-Resilient Stochastic Gradient Descent for Distributed Learning: A Lipschitz-Inspired Coordinate-wise Median Approach”. In: *CoRR* abs/1909.04532 (2019). arXiv: 1909.04532. URL: http://arxiv.org/abs/1909.04532.

[64] Xiangyi Chen, Tiancong Chen, Haoran Sun, Zhiwei Steven Wu, and Mingyi Hong. “Distributed Training with Heterogeneous Data: Bridging Median- and Mean-Based Algorithms”. In: *CoRR* abs/1906.01736 (2019). arXiv: 1906.01736. URL: http://arxiv.org/abs/1906.01736.

[65] Jeff Daily, Abhinav Vishnu, Charles Siegel, Thomas Warfel, and Vinay Amatya. “GossipGraD: Scalable Deep Learning using Gossip Communication based Asynchronous Gradient Descent”. In: *CoRR* abs/1803.05880 (2018). arXiv: 1803.05880. URL: http://arxiv.org/abs/1803.05880.

[66] Cong Xie, Sanmi Koyejo, and Indranil Gupta. “Asynchronous Federated Optimization”. In: *CoRR* abs/1903.03934 (2019). arXiv: 1903.03934. URL: http://arxiv.org/abs/1903.03934.

[67] José Pereira and Laura Ricci, eds. *Distributed Applications and Interoperable Systems - 19th IFIP WG 6.1 International Conference, DAIS 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17-21, 2019, Proceedings*. Vol. 11534. Lecture Notes in Computer Science. Springer, 2019. ISBN: 978-3-030-22495-0. DOI: 10.1007/978-3-030-22496-7. URL: https://doi.org/10.1007/978-3-030-22496-7.

[68] Paul Vanhaesebrouck, Aurélien Bellet, and Marc Tommasi. “Decentralized Collaborative Learning of Personalized Models over Networks”. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*. 2017, pp. 509–517. URL: http://proceedings.mlr.press/v54/vanhaesebrouck17a.html.

[69] Ali H. Sayed, Sheng-Yuan Tu, Jianshu Chen, Xiaochuan Zhao, and Zaid J. Towfic. “Diffusion Strategies for Adaptation and Learning over Networks: An Examination of Distributed Strategies and Network Behavior”. In: *IEEE Signal Processing Magazine* 30.3 (2013), pp. 155–171. DOI: 10.1109/MSP.2012.2231991.

[70] Ali H. Sayed. “Adaptive Networks”. In: *Proceedings of the IEEE* 102.4 (2014), pp. 460–497. DOI: 10.1109/JPROC.2014.2306253.

[71] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*. 2017, pp. 1273–1282. URL: http://proceedings.mlr.press/v54/mcmahan17a.html.

[72] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf.

[73] Ali H. Sayed. "Adaptation, Learning, and Optimization over Networks". In: *Found. Trends Mach. Learn.* 7.4-5 (2014), pp. 311–801. DOI: 10.1561/2200000051. URL: https://doi.org/10.1561/2200000051.

[74] Nirupam Gupta and Nitin H. Vaidya. "Byzantine Fault Tolerant Distributed Linear Regression". In: *CoRR* abs/1903.08752 (2019). arXiv: 1903.08752. URL: http://arxiv.org/abs/1903.08752.

[75] L. Su and N. H. Vaidya. "Byzantine-Resilient Multi-Agent Optimization". In: *IEEE Transactions on Automatic Control* (2020).

[76] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. "The Hidden Vulnerability of Distributed Learning in Byzantium". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018, pp. 3518–3527. URL: http://proceedings.mlr.press/v80/mhamdi18a.html.

[77] El-Mahdi El-Mhamdi and Rachid Guerraoui. "Fast and Secure Distributed Learning in High Dimension". In: *CoRR* abs/1905.04374 (2019). arXiv: 1905.04374. URL: http://arxiv.org/abs/1905.04374.

[78] Gilad Baruch, Moran Baruch, and Yoav Goldberg. "A Little Is Enough: Circumventing Defenses For Distributed Learning". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. 2019, pp. 8632–8642. URL: http://papers.nips.cc/paper/9069-a-little-is-enough-circumventing-defenses-for-distributed-learning.

[79] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. "Fall of Empires: Breaking Byzantine-tolerant SGD by Inner Product Manipulation". In: *Proceedings of the Thirty-Fifth Conference on Uncertainty*

in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019. 2019, p. 83. URL: http://auai.org/uai2019/proceedings/papers/83.pdf.

[80] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. "Local Model Poisoning Attacks to Byzantine-Robust Federated Learning". In: *CoRR* abs/1911.11815 (2019). arXiv: 1911.11815. URL: http://arxiv.org/abs/1911.11815.

[81] Lingjiao Chen, Hongyi Wang, Zachary B. Charles, and Dimitris S. Papailiopoulos. "DRACO: Byzantine-resilient Distributed Training via Redundant Gradients". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018, pp. 902–911. URL: http://proceedings.mlr.press/v80/chen18l.html.

[82] Shashank Rajput, Hongyi Wang, Zachary B. Charles, and Dimitris S. Papailiopoulos. "DETOX: A Redundancy-based Framework for Faster and More Robust Gradient Aggregation". In: *CoRR* abs/1907.12205 (2019). eprint: 1907.12205.

[83] Deepesh Data, Linqi Song, and Suhas N. Diggavi. "Data Encoding Methods for Byzantine-Resilient Distributed Optimization". In: *IEEE International Symposium on Information Theory (ISIT)*. 2019, pp. 2719–2723. DOI: 10.1109/ISIT.2019.8849857.

[84] Liping Li, Wei Xu, Tianyi Chen, Georgios B. Giannakis, and Qing Ling. "RSA: Byzantine-Robust Stochastic Aggregation Methods for Distributed Learning from Heterogeneous Datasets". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. 2019, pp. 1544–1551. DOI: 10.1609/aaai.v33i01.33011544. URL: https://doi.org/10.1609/aaai.v33i01.33011544.

[85] Rich Caruana. "Multitask Learning". In: *Mach. Learn.* 28.1 (1997), pp. 41–75. DOI: 10.1023/A:1007379606734. URL: https://doi.org/10.1023/A:1007379606734.

[86] Sebastian Ruder. "An Overview of Multi-Task Learning in Deep Neural Networks". In: *CoRR* abs/1706.05098 (2017). arXiv: 1706.05098. URL: http://arxiv.org/abs/1706.05098.

[87] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Philip S. Yu. "Learning Multiple Tasks with Multilinear Relationship Networks". In: *Advances in Neural Information Processing Systems, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett. 2017, pp. 1594–1603. URL: http://papers.nips.cc/paper/6757-learning-multiple-tasks-with-multilinear-relationship-networks.

[88] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. “Cross-Stitch Networks for Multi-task Learning”. In: *IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA*. 2016, pp. 3994–4003. DOI: 10.1109/CVPR.2016.433. URL: https://doi.org/10.1109/CVPR.2016.433.

[89] Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. “A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Copenhagen, Denmark*. Ed. by Martha Palmer, Rebecca Hwa, and Sebastian Riedel. 2017, pp. 1923–1933. DOI: 10.18653/v1/d17-1206. URL: https://doi.org/10.18653/v1/d17-1206.

[90] Alex Kendall, Yarin Gal, and Roberto Cipolla. “Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics”. In: *IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA*. IEEE Computer Society, 2018, pp. 7482–7491. DOI: 10.1109/CVPR.2018.00781. URL: http://openaccess.thecvf.com/content_cvpr_2018/html/Kendall_Multi-Task_Learning_Using_CVPR_2018_paper.html.

[91] Theodoros Evgeniou and Massimiliano Pontil. “Regularized multi–task learning”. In: *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA*. 2004, pp. 109–117. DOI: 10.1145/1014052.1014067. URL: https://doi.org/10.1145/1014052.1014067.

[92] Jiayu Zhou, Jianhui Chen, and Jieping Ye. “Clustered Multi-Task Learning Via Alternating Structure Optimization”. In: *Advances in Neural Information Processing Systems, Granada, Spain*. 2011, pp. 702–710. URL: http://papers.nips.cc/paper/4292-clustered-multi-task-learning-via-alternating-structure-optimization.

[93] Jianhui Chen, Jiayu Zhou, and Jieping Ye. “Integrating low-rank and group-sparse structures for robust multi-task learning”. In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA*. 2011, pp. 42–50. DOI: 10.1145/2020408.2020423. URL: https://doi.org/10.1145/2020408.2020423.

[94] Laurent Jacob, Francis R. Bach, and Jean-Philippe Vert. “Clustered Multi-Task Learning: A Convex Formulation”. In: *Advances in Neural Information Processing Systems, Vancouver, British Columbia, Canada*. 2008, pp. 745–752. URL: http://papers.nips.cc/paper/3499-clustered-multi-task-learning-a-convex-formulation.

[95] Yu Zhang and Dit-Yan Yeung. "A Convex Formulation for Learning Task Relationships in Multi-Task Learning". In: *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA*. 2010, pp. 733–442. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1\&smnu=2\&article_id=2117\&proceeding_id=26.

[96] Avishek Saha, Piyush Rai, Hal Daumé III, and Suresh Venkatasubramanian. "Online Learning of Multiple Tasks and Their Relationships". In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, USA*. 2011, pp. 643–651. URL: http://proceedings.mlr.press/v15/saha11b/saha11b.pdf.

[97] J. Chen, C. Richard, and A. H. Sayed. "Diffusion LMS Over Multitask Networks". In: *IEEE Transactions on Signal Processing* 63.11 (June 2015), pp. 2733–2748. ISSN: 1053-587X. DOI: 10.1109/TSP.2015.2412918.

[98] Keerthiram Murugesan, Hanxiao Liu, Jaime G. Carbonell, and Yiming Yang. "Adaptive Smoothed Online Multi-Task Learning". In: *Advances in Neural Information Processing Systems, Barcelona, Spain*. 2016, pp. 4296–4304. URL: http://papers.nips.cc/paper/6434-adaptive-smoothed-online-multi-task-learning.

[99] Keerthiram Murugesan and Jaime G. Carbonell. "Active Learning from Peers". In: *Advances in Neural Information Processing Systems, Long Beach, CA, USA*. 2017, pp. 7008–7017. URL: http://papers.nips.cc/paper/7276-active-learning-from-peers.

[100] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. "Domain-Adversarial Training of Neural Networks". In: *J. Mach. Learn. Res.* 17.1 (Jan. 2016), 2096–2030. ISSN: 1532-4435.

[101] Yitong Li, michael Murias, geraldine Dawson, and David E Carlson. "Extracting Relationships by Multi-Domain Matching". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 6798–6809. URL: http://papers.nips.cc/paper/7913-extracting-relationships-by-multi-domain-matching.pdf.

[102] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. "An Efficient k-Means Clustering Algorithm: Analysis and Implementation". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 24.7 (2002), pp. 881–892. DOI: 10.1109/TPAMI.2002.1017616. URL: https://doi.org/10.1109/TPAMI.2002.1017616.

[103] X. Zhao and A. H. Sayed. “Distributed Clustering and Learning Over Networks”. In: *IEEE Transactions on Signal Processing* 63.13 (July 2015), pp. 3285–3300. ISSN: 1053-587X. DOI: 10.1109/TSP.2015.2415755.

[104] S. Khawatmi, A. M. Zoubir, and A. H. Sayed. “Decentralized clustering over adaptive networks”. In: *2015 23rd European Signal Processing Conference (EUSIPCO)*. Aug. 2015, pp. 2696–2700. DOI: 10.1109/EUSIPCO.2015.7362874.

[105] J. Chen, C. Richard, and A. H. Sayed. “Multitask Diffusion Adaptation Over Networks”. In: *IEEE Transactions on Signal Processing* 62.16 (Aug. 2014), pp. 4129–4144. ISSN: 1053-587X. DOI: 10.1109/TSP.2014.2333560.

[106] Mingbao Lin, Rongrong Ji, Shen Chen, Feng Zheng, Xiaoshuai Sun, Baochang Zhang, Liujuan Cao, Guodong Guo, and Feiyue Huang. “Supervised Online Hashing via Similarity Distribution Learning”. In: *CoRR* abs/1905.13382 (2019). arXiv: 1905.13382. URL: http://arxiv.org/abs/1905.13382.

[107] A. Feinberg. “Markov Decision Processes: Discrete Stochastic Dynamic Programming (Martin L. Puterman)”. In: *SIAM Review* 38.4 (1996), p. 689. DOI: 10.1137/1038137. URL: https://doi.org/10.1137/1038137.

[108] Sergio Valcarcel Macua, Jianshu Chen, Santiago Zazo, and Ali H. Sayed. “Distributed Policy Evaluation Under Multiple Behavior Strategies”. In: *IEEE Trans. Automat. Contr.* 60.5 (2015), pp. 1260–1274. DOI: 10.1109/TAC.2014.2368731. URL: https://doi.org/10.1109/TAC.2014.2368731.

[109] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. “Massively Parallel Methods for Deep Reinforcement Learning”. In: *CoRR* abs/1507.04296 (2015). arXiv: 1507.04296. URL: http://arxiv.org/abs/1507.04296.

[110] Kaiqing Zhang, Zhuoran Yang, and Tamer Basar. “Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms”. In: *CoRR* abs/1911.10635 (2019). arXiv: 1911.10635. URL: http://arxiv.org/abs/1911.10635.

[111] Maria-Florina Balcan and Kilian Q. Weinberger, eds. *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016. URL: http://proceedings.mlr.press/v48/.

[112] Soummya Kar, José M. F. Moura, and H. Vincent Poor. "QD-Learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement Learning Through *Consensus* + *Innovations*". In: *IEEE Trans. Signal Process.* 61.7 (2013), pp. 1848–1862. DOI: 10.1109/TSP.2013.2241057. URL: https://doi.org/10.1109/TSP.2013.2241057.

[113] Sergio Valcarcel Macua, Aleksi Tukiainen, Daniel García-Ocaña Hernández, David Baldazo, Enrique Munoz de Cote, and Santiago Zazo. "Diff-DAC: Distributed Actor-Critic for Multitask Deep Reinforcement Learning". In: *CoRR* abs/1710.10363 (2017). arXiv: 1710.10363. URL: http://arxiv.org/abs/1710.10363.

[114] Yee Whye Teh, Victor Bapst, Wojciech M. Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. "Distral: Robust multitask reinforcement learning". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 2017, pp. 4496–4506. URL: http://papers.nips.cc/paper/7036-distral-robust-multitask-reinforcement-learning.

[115] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. "DRN: A Deep Reinforcement Learning Framework for News Recommendation". In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*. Ed. by Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis. ACM, 2018, pp. 167–176. DOI: 10.1145/3178876.3185994. URL: https://doi.org/10.1145/3178876.3185994.

[116] A. Bodas, B. Upadhyay, C. Nadiger, and S. Abdelhak. "Reinforcement learning for game personalization on edge devices". In: *2018 International Conference on Information and Computer Technologies (ICICT)*. 2018, pp. 119–122. DOI: 10.1109/INFOCT.2018.8356853.

[117] M. Mehdi Afsar, Trafford Crump, and Behrouz H. Far. "Reinforcement learning based recommender systems: A survey". In: *CoRR* abs/2101.06286 (2021). arXiv: 2101.06286. URL: https://arxiv.org/abs/2101.06286.

[118] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. "Multiagent Cooperation and Competition with Deep Reinforcement Learning". In: *CoRR* abs/1511.08779 (2015). arXiv: 1511.08779. URL: http://arxiv.org/abs/1511.08779.

[119] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 De-*

cember 2017, Long Beach, CA, USA. 2017, pp. 6379–6390. URL: http://papers.nips.cc/paper/7217-multi-agent-actor-critic-for-mixed-cooperative-competitive-environments.

[120] Jakob N. Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. "Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017, pp. 1146–1155. URL: http://proceedings.mlr.press/v70/foerster17b.html.

[121] Jiachen Yang, Alireza Nakhaei, David Isele, Hongyuan Zha, and Kikuo Fujimura. "CM3: Cooperative Multi-goal Multi-stage Multi-agent Reinforcement Learning". In: *CoRR* abs/1809.05188 (2018). arXiv: 1809.05188. URL: http://arxiv.org/abs/1809.05188.

[122] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. "Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks". In: *CoRR* abs/1602.02672 (2016). arXiv: 1602.02672. URL: http://arxiv.org/abs/1602.02672.

[123] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. "Learning to Communicate with Deep Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016, pp. 2137–2145. URL: http://papers.nips.cc/paper/6042-learning-to-communicate-with-deep-multi-agent-reinforcement-learning.

[124] Kaiqing Zhang, Zhuoran Yang, and Tamer Basar. "Networked Multi-Agent Reinforcement Learning in Continuous Spaces". In: *57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018*. 2018, pp. 2771–2776. DOI: 10.1109/CDC.2018.8619581. URL: https://doi.org/10.1109/CDC.2018.8619581.

[125] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. "Fully Decentralized Multi-Agent Reinforcement Learning with Networked Agents". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018, pp. 5867–5876. URL: http://proceedings.mlr.press/v80/zhang18n.html.

[126] Yixuan Lin, Shripad Gade, Romeil Sandhu, and Ji Liu. "Toward Resilient Multi-Agent Actor-Critic Algorithms for Distributed Reinforcement Learning". In: *2020 American Control Conference, ACC 2020, Denver, CO, USA, July 1-3, 2020*. IEEE, 2020, pp. 3953–3958. DOI: 10.23919/ACC45564.2020.9147381. URL: https://doi.org/10.23919/ACC45564.2020.9147381.

[127] Yijing Xie, Shaoshuai Mou, and Shreyas Sundaram. “Towards Resilience for Multi-Agent QD-Learning”. In: *CoRR* abs/2104.03153 (2021). arXiv: 2104.03153. URL: https://arxiv.org/abs/2104.03153.

[128] Waseem Abbas, Aron Laszka, and Xenofon Koutsoukos. “Improving network connectivity and robustness using trusted nodes with application to resilient consensus”. In: *IEEE Transactions on Control of Network Systems* 5.4 (2017), pp. 2036–2048.

[129] Lili Su and Nitin Vaidya. “Multi-agent optimization in the presence of byzantine adversaries: Fundamental limits”. In: *2016 American Control Conference (ACC)*. IEEE. 2016, pp. 7183–7188.

[130] Heath J LeBlanc, Haotian Zhang, Xenofon Koutsoukos, and Shreyas Sundaram. “Resilient asymptotic consensus in robust networks”. In: *IEEE Journal on Selected Areas in Communications* 31.4 (2013), pp. 766–781.

[131] Mudassir Shabbir. “Some results in computational and combinatorial geometry”. PhD thesis. Rutgers University-New Brunswick, 2014.

[132] Nabil H Mustafa, Saurabh Ray, and Mudassir Shabbir. “k-Centerpoints Conjectures for Pointsets in d”. In: *International Journal of Computational Geometry & Applications* 25.03 (2015), pp. 163–185.

[133] Helge Tverberg. “A generalization of Radon’s theorem”. In: *Journal of the London Mathematical Society* 1.1 (1966), pp. 123–128.

[134] John R. Reay. “An extension of Radon’s theorem”. In: *Illinois J. Math.* 12.2 (June 1968), pp. 184–189. DOI: 10.1215/ijm/1256054209.

[135] Jean-Pierre Roudneff. “New cases of Reay’s conjecture on partitions of points into simplices with k-dimensional intersection”. In: *European Journal of Combinatorics* 30.8 (2009), pp. 1919–1943.

[136] Jean-Pierre Roudneff. “Partitions of points into intersecting tetrahedra”. In: *Discrete Mathematics* 81.1 (1990), pp. 81–86.

[137] Wolfgang Mulzer and Daniel Werner. “Approximating Tverberg points in linear time for any fixed dimension”. In: *Discrete & Computational Geometry* 50.2 (2013), pp. 520–535.

[138] Richard Rado. “A theorem on general measure”. In: *Journal of the London Mathematical Society* 1.4 (1946), pp. 291–300.

[139] Jiří Matoušek. *Lectures on discrete geometry*. Vol. 108. Springer, 2002.

[140] Shreesh Jadhav and Asish Mukhopadhyay. “Computing a centerpoint of a finite planar set of points in linear time”. In: *Discrete & Computational Geometry* 12.3 (1994), pp. 291–312.

[141] Nimrod Megiddo. “Partitioning with two lines in the plane”. In: *Journal of Algorithms* 6.3 (1985), pp. 430 –433. ISSN: 0196-6774.

[142] Timothy M Chan. “An optimal randomized algorithm for maximum Tukey depth”. In: *Proceedings of the 15th annual ACM-SIAM Symposium on Discrete Slgorithms (SODA)*. SIAM. 2004, pp. 430–436.

[143] Bernard Chazelle. “Cutting hyperplanes for divide-and-conquer”. In: *Discrete & Computational Geometry* 9.2 (1993), pp. 145–158.

[144] Gary L Miller and Donald R Sheehy. “Approximate centerpoints with proofs”. In: *Computational Geometry* 43.8 (2010), pp. 647–654.

[145] Anusha Lalitha, Osman Cihan Kilinc, Tara Javidi, and Farinaz Koushanfar. “Peer-to-peer Federated Learning on Graphs”. In: *CoRR* abs/1901.11173 (2019). arXiv: 1901.11173. URL: http://arxiv.org/abs/1901.11173.

[146] Jorge Plata-Chaves, Nikola Bogdanovic, and Kostas Berberidis. “Distributed Diffusion-Based LMS for Node-Specific Adaptive Parameter Estimation”. In: *IEEE Trans. Signal Processing* 63.13 (2015), pp. 3448–3460.

[147] Reza Abdolee, Stephan Saur, Benoît Champagne, and Ali H. Sayed. “Diffusion LMS localization and tracking algorithm for wireless cellular networks”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing, (ICASSP)*. 2013, pp. 4598–4602.

[148] Xiaochuan Zhao and Ali H. Sayed. “Clustering via diffusion adaptation over networks”. In: *3rd International Workshop on Cognitive Information Processing (CIP)*. 2012, pp. 1–6. DOI: 10.1109/CIP.2012.6232902.

[149] S. Tu and A. H. Sayed. “Mobile Adaptive Networks”. In: *IEEE Journal of Selected Topics in Signal Processing* 5.4 (Aug. 2011), pp. 649–664. DOI: 10.1109/JSTSP.2011.2125943.

[150] Roger M. Kieckhafer and Mohammad H. Azadmanesh. “Reaching Approximate Agreement with Mixed-Mode Faults”. In: *IEEE Trans. Parallel Distrib. Syst.* 5.1 (1994), pp. 53–63. DOI: 10.1109/71.262588. URL: https://doi.org/10.1109/71.262588.

[151] Nitin H. Vaidya, Lewis Tseng, and Guanfeng Liang. “Iterative approximate byzantine consensus in arbitrary directed graphs”. In: *ACM Symposium on Principles of Distributed Computing, PODC ’12, Funchal, Madeira, Portugal, July 16-18, 2012*. 2012, pp. 365–374. DOI: 10.1145/2332432.2332505. URL: https://doi.org/10.1145/2332432.2332505.

[152] Hyongju Park and Seth Hutchinson. “Fault-Tolerant Rendezvous of Multirobot Systems”. In: *IEEE Trans. Robotics* 33.3 (2017), pp. 565–582.

[153] Zang Li, Wade Trappe, Yanyong Zhang, and Badri Nath. “Robust statistical methods for securing wireless localization in sensor networks”. In: *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005, April 25-27, 2005, UCLA, Los Angeles, California, USA*. 2005, pp. 91–98. DOI: 10.1109/IPSN.2005.1440903. URL: https://doi.org/10.1109/IPSN.2005.1440903.

[154] Yingpei Zeng, Jiannong Cao, Jue Hong, Shigeng Zhang, and Li Xie. “Secure localization and location verification in wireless sensor networks: a survey”. In: *J. Supercomput.* 64.3 (2013), pp. 685–701. DOI: 10.1007/s11227-010-0501-4. URL: https://doi.org/10.1007/s11227-010-0501-4.

[155] Zhixiong Yang, Arpita Gang, and Waheed U. Bajwa. “Adversary-resilient Inference and Machine Learning: From Distributed to Decentralized”. In: *CoRR* abs/1908.08649 (2019). arXiv: 1908.08649. URL: http://arxiv.org/abs/1908.08649.

[156] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. “Optimization Methods for Large-Scale Machine Learning”. In: *SIAM Rev.* 60.2 (2018), pp. 223–311. DOI: 10.1137/16M1080173. URL: https://doi.org/10.1137/16M1080173.

[157] Jesús De Loera, Xavier Goaoc, Frédéric Meunier, and Nabil Mustafa. “The discrete yet ubiquitous theorems of Carathéodory, Helly, Sperner, Tucker, and Tverberg”. In: *Bulletin of the American Mathematical Society* 56.3 (2019), pp. 415–511.

[158] Wolfgang Mulzer and Daniel Werner. “Approximating Tverberg Points in Linear Time for Any Fixed Dimension”. In: *Discret. Comput. Geom.* 50.2 (2013), pp. 520–535. DOI: 10.1007/s00454-013-9528-7. URL: https://doi.org/10.1007/s00454-013-9528-7.

[159] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron D. Ames, Eric Feron, and Magnus Egerstedt. “The Robotarium: A remotely accessible swarm robotics research testbed”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1699–1706. DOI: 10.1109/ICRA.2017.7989200.

[160] J. Plata-Chaves, N. Bogdanović, and K. Berberidis. “Distributed Diffusion-Based LMS for Node-Specific Adaptive Parameter Estimation”. In: *IEEE Transactions on Signal Processing* 63 (2015), pp. 3448–3460.

[161] Amin Lotfzad Pak, Azam Khalili, Md. Kafiul Islam, and Amir Rastegarnia. “A Distributed Target Localization Algorithm for Mobile Adaptive Networks”. In: *ECTI Transactions on Electrical Engineering, Electronics, and Communications* 14 (Aug. 2016), pp. 47–56.

[162] Sheng-Yuan Tu and Ali H.Sayed. “Mobile Adaptive Networks”. In: *IEEE J. Sel. Topics Signal Process* 5.4 (2011), pp. 649–664.

[163] J. Chen and A. H. Sayed. “Diffusion Adaptation Strategies for Distributed Optimization and Learning Over Networks”. In: *IEEE Transactions on Signal Processing* 60.8 (Aug. 2012), pp. 4289–4305. ISSN: 1053-587X. DOI: 10.1109/TSP.2012.2198470.

[164] R. Nassif, C. Richard, A. Ferrari, and A. H. Sayed. “Proximal Multitask Learning Over Networks With Sparsity-Inducing Coregularization”. In: *IEEE Transactions on Signal Processing* 64.23 (2016), pp. 6329–6344.

[165] A. Rastegarnia, W. M. Bazzi, A. Khalili, and J. A. Chambers. “Diffusion adaptive networks with imperfect communications: link failure and channel noise”. In: *IET Signal Processing* 8.1 (2014), pp. 59–66.

[166] Aritra Mitra and Shreyas Sundaram. “Secure distributed observers for a class of linear time invariant systems in the presence of byzantine adversaries”. In: *55th IEEE Conference on Decision and Control (CDC)*. IEEE. 2016, pp. 2709–2714.

[167] Yudong Chen, Lili Su, and Jiaming Xu. “Distributed Statistical Machine Learning in Adversarial Settings: Byzantine Gradient Descent”. In: *POMACS* 1.2 (2017), 44:1–44:25.

[168] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. “Data Poisoning Attacks on Factorization-Based Collaborative Filtering”. In: *Annual Conference on Neural Information Processing Systems*. 2016, pp. 1885–1893.

[169] Lin Xiao, Stephen Boyd, and Seung-Jean Kim. “Distributed average consensus with least-mean-square deviation”. In: *Journal of Parallel and Distributed computing* 67.1 (2007), pp. 33–46.

[170] Angelia Nedic and Asuman Ozdaglar. “Distributed subgradient methods for multi-agent optimization”. In: *IEEE Transactions on Automatic Control* 54.1 (2009), pp. 48–61.

[171] Usman A Khan, Soummya Kar, and José MF Moura. “Higher dimensional consensus: Learning in large-scale networks”. In: *IEEE Transactions on Signal Processing* 58.5 (2010), pp. 2836–2849.

[172] Ion Matei and John S Baras. “Consensus-based linear distributed filtering”. In: *Automatica* 48.8 (2012), pp. 1776–1782.

[173] Jianshu Chen and Ali H Sayed. “Diffusion adaptation strategies for distributed optimization and learning over networks”. In: *IEEE Transactions on Signal Processing* 60.8 (2012), pp. 4289–4305.

[174] Fabio Pasqualetti, Antonio Bicchi, and Francesco Bullo. “Consensus Computation in Unreliable Networks: A System Theoretic Approach”. In: *IEEE Transactions on Automatic Control* 57.1 (2012), pp. 90–104.

[175] Chengcheng Zhao, Jianping He, and Jiming Chen. “Resilient Consensus with Mobile Detectors Against Malicious Attacks”. In: *IEEE Transactions on Signal and Information Processing over Networks* 4.1 (2018), pp. 60–69.

[176] Heath J. LeBlanc and Firas Hassan. “Resilient distributed parameter estimation in heterogeneous time-varying networks”. In: *3rd International Conference on High Confidence Networked Systems (part of CPS Week), HiCoNS ’14, Berlin, Germany, April 15-17, 2014*. 2014, pp. 19–28.

[177] Yuan Chen, Soummya Kar, and José M. F. Moura. “Resilient Distributed Estimation: Sensor Attacks”. In: *arXiv e-prints*, arXiv:1709.06156 (Sept. 2017), arXiv:1709.06156. arXiv: 1709.06156 [math.OC].

[178] Y. Chen, S. Kar, and J. M. F. Moura. “Attack Resilient Distributed Estimation: A Consensus+Innovations Approach”. In: *American Control Conference (ACC)*. 2018.

[179] Ali H. Sayed. “Diffusion Adaptation Over Networks”. In: *Academic Press Library in Signal Processing*. Ed. by Abdelhak M. Zoubir, Mats Viberg, Rama Chellappa, and Sergios Theodoridis. Vol. 3. Elsevier, 2014. Chap. 9, pp. 323–453.

[180] Stephen T. Hedetniemi, Renu C. Laskar, and John Pfaff. “A linear algorithm for finding a minimum dominating set in a cactus”. In: *Discrete Applied Mathematics* 13.2-3 (1986), pp. 287–292.

[181] Reza Olfati-Saber, J. Alexander Fax, and Richard M. Murray. “Consensus and Cooperation in Networked Multi-Agent Systems”. In: *Proc. IEEE* 95.1 (2007), pp. 215–233. DOI: 10.1109/JPROC.2006.887293. URL: https://doi.org/10.1109/JPROC.2006.887293.

[182] Roula Nassif, Stefan Vlaski, Cédric Richard, Jie Chen, and Ali H. Sayed. “Multitask Learning Over Graphs: An Approach for Distributed, Streaming Machine Learning”. In: *IEEE Signal Process. Mag.* 37.3 (2020), pp. 14–25. DOI: 10.1109/MSP.2020.2966273. URL: https://doi.org/10.1109/MSP.2020.2966273.

[183] Danqi Jin, Jie Chen, Cédric Richard, Jingdong Chen, and Ali H. Sayed. “Affine Combination of Diffusion Strategies Over Networks”. In: *IEEE Transactions on Signal Processing* 68 (2020), pp. 2087–2104.

[184] Jie Chen, Cédric Richard, Shang Kee Ting, and Ali H. Sayed. “Chapter 3 - Multitask Learning Over Adaptive Networks With Grouping Strategies”. In: *Cooperative and Graph Signal Processing*. Ed. by Petar M. Djurić and Cédric Richard. Academic Press, 2018, pp. 107 –129. ISBN: 978-0-12-813677-5. DOI: https://doi.org/10.1016/B978-0-12-813677-5.00003-1. URL: http://www.sciencedirect.com/science/article/pii/B9780128136775000031.

[185] J. Chen, C. Richard, and A. H. Sayed. “Diffusion LMS for clustered multitask networks”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing*. 2014, pp. 5487–5491.

[186] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[187] Prasun Roy, Subhankar Ghosh, Saumik Bhattacharya, and Umapada Pal. “Effects of Degradations on Deep Neural Network Architectures”. In: *arXiv preprint arXiv:1807.10108* (2018).

[188] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. “Federated Learning of Deep Networks using Model Averaging”. In: *CoRR* abs/1602.05629 (2016). arXiv: 1602.05629. URL: http://arxiv.org/abs/1602.05629.

[189] Sheng-Yuan Tu and Ali H. Sayed. “Distributed Decision-Making Over Adaptive Networks”. In: *IEEE Trans. Signal Process.* 62.5 (2014), pp. 1054–1069. DOI: 10.1109/TSP.2013.2296271. URL: https://doi.org/10.1109/TSP.2013.2296271.

[190] Abraham Berman and Robert J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Vol. 9. Classics in Applied Mathematics. SIAM, 1994. ISBN: 978-0-89871-321-3. DOI: 10.1137/1.9781611971262. URL: https://doi.org/10.1137/1.9781611971262.

[191] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. “Asynchronous Methods for Deep Reinforcement Learning”. In: *ICML 2016, New York City, NY, USA, June 19-24, 2016*. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 1928–1937. URL: http://proceedings.mlr.press/v48/mniha16.html.

[192] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures”. In: *ICML 2018, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1406–1415. URL: http://proceedings.mlr.press/v80/espeholt18a.html.

[193] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning*. 1992, pp. 279–292.

[194] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[195] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Advances in Neural Information Processing Systems 12, Denver, Colorado, USA*. 1999, pp. 1057–1063. URL: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.

[196] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1506.02438.

[197] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. “OpenAI Gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[198] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. “Continuous control with deep reinforcement learning.” In: *ICLR*. 2016. URL: http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15.

[199] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *ICLR 2015, San Diego, CA, USA, May 7-9*. 2015. URL: http://arxiv.org/abs/1412.6980.

[200] Lucian Busoniu, Robert Babuska, and Bart De Schutter. “Multi-agent Reinforcement Learning: An Overview”. In: vol. 310. July 2010, pp. 183–221. ISBN: 978-3-642-14434-9. DOI: 10.1007/978-3-642-14435-6_7.

[201] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. “Weight Uncertainty in Neural Networks”. In: *CoRR* abs/1505.05424 (2015). arXiv: 1505.05424. URL: http://arxiv.org/abs/1505.05424.

[202] J. Chen, C. Richard, and A. H. Sayed. “Adaptive clustering for multitask diffusion networks”. In: *2015 23rd European Signal Processing Conference (EUSIPCO)*. Aug. 2015, pp. 200–204. DOI: 10.1109/EUSIPCO.2015.7362373.

[203] Shalabh Bhatnagar, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee. “Natural actor-critic algorithms”. In: *Autom.* 45.11 (2009), pp. 2471–2482. DOI: 10.1016/j.automatica.2009.07.008. URL: https://doi.org/10.1016/j.automatica.2009.07.008.

[204] Yan Zhang and Michael M. Zavlanos. “Distributed off-Policy Actor-Critic Reinforcement Learning with Policy Consensus”. In: *58th IEEE Conference on Decision and Control, CDC 2019, Nice, France, December 11-13, 2019*. IEEE, 2019, pp. 4674–4679. DOI: 10.1109/CDC40024.2019.9029969. URL: https://doi.org/10.1109/CDC40024.2019.9029969.

[205] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. “Continuous control with deep reinforcement learning”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1509.02971.

[206] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *ICML*. 2018, pp. 1582–1591. URL: http://proceedings.mlr.press/v80/fujimoto18a.html.

[207] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1856–1865. URL: http://proceedings.mlr.press/v80/haarnoja18b.html.

[208] E. Todorov, T. Erez, and Y. Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[209] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6572.