Multi-agent Reinforcement Learning on the EDGE through Integrating Spiking Neural Network and Federated Learning

by

Mohammad Tayefe Ramezanlou

Thesis proposal submitted to
The Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering
Ottawa-Carleton Institute of Electrical and Computer Engineering Department of
Systems and Computer Engineering

Carleton University
Ottawa, Ontario
Spring, 2024

Abstract

This proposal outlines an innovative approach to multiagent reinforcement learning by integrating Spiking Neural Networks (SNNs) with Federated Learning (FL), aimed at advancing the capabilities of edge computing. This integration promises notable improvements in learning efficiency and data privacy but presents unique challenges requiring further investigation. The research will focus on four pivotal areas: enhancing learning stability in SNNs, reducing communication overhead in FL, exploring the heterogeneity of spiking neuron models, and strengthening resilience against model poisoning attacks.

This research aims to develop adaptive mechanisms for SNNs to maintain learning stability amidst dynamic environmental rewards, thereby addressing the critical balance between synaptic plasticity and equilibrium. The proposal also aims to innovate in managing communication overhead by introducing adaptive communication strategies that ensure efficiency without compromising responsiveness, leveraging algorithms that reduce redundancy in model updates.

Furthermore, the research will investigate the implications of neuron model diversity within SNNs, examining how heterogeneity affects learning outcomes and system performance. This includes employing methodologies such as Neuroevolution of Augmenting Topologies (NEAT), transfer learning, and evolutionary optimization algorithms to optimize the global model. Lastly, the proposal emphasizes enhancing the system's defense against model poisoning attacks, proposing the use of robust statistical methods to detect and mitigate such risks.

By addressing these areas, this research aims to significantly contribute to the develop-

ment of more resilient, efficient, and adaptive multiagent systems, leveraging the synergies between SNNs and FL in dynamic and potentially adversarial environments. This work aims to improve the efficiency of bio-inspired neural network models in edge computing and ensure the integrity and privacy of data in decentralized networks.

Contents

List of Figures

List of Tables

Acronyms

FL Federated Learning

GRF Gaussian Receptive Fields

LIF Leaky Integrate-and-Fire

R-STDP Reward-modulated Spike-Timing-Dependent Plasticity

RCSE Reward-Modulated Competitive Synaptic Equilibrium

SNN Spiking Neural Network

Chapter 1

Introduction

Federated Learning (FL) is a groundbreaking approach to machine learning that allows models to be trained directly on edge devices without centralizing data. This method addresses significant concerns regarding data privacy and security by keeping sensitive information local to the device it originates from, thus preventing data leakage and enhancing user privacy. FL represents a shift towards decentralized, privacy-preserving machine learning models, making it a critical area of research for advancing machine learning technologies that are both accessible and secure.

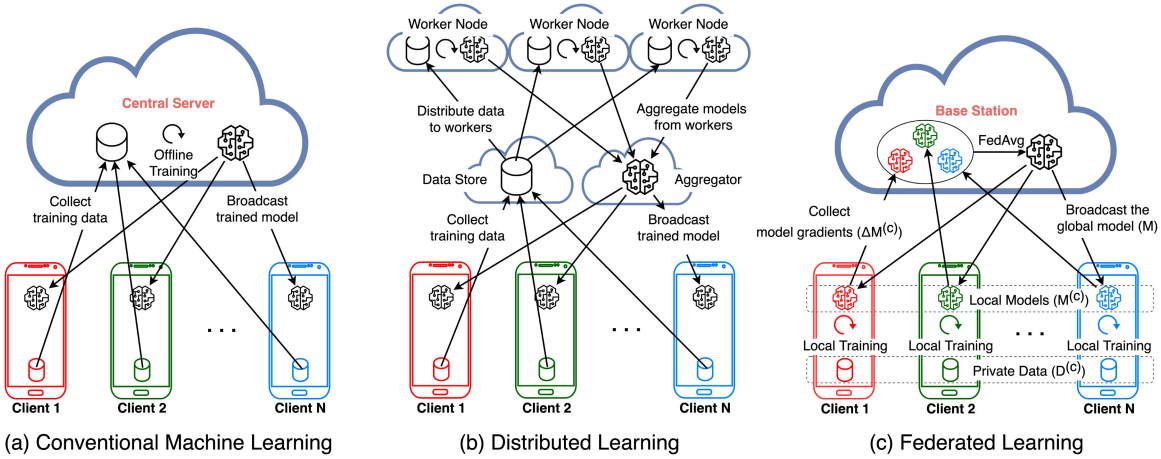

Figure 1.1: Schematic diagram to illustrate distributed learning and FL [1].

Figure 1.1 shows a diagram of distributed learning and FL. In conventional learning, the data is collected from the clients, and the model is trained offline. In distributed learning, the dataset is collected at a single data store, and then it is distributed across multiple worker nodes for training. In FL, the training is performed at the client, and model gradients are communicated to the central server, which aggregates the gradients, updates the global model, and broadcasts it back to all the clients. Each of these processes is iterative, and models for the clients are periodically updated.

Spiking Neural Network (SNN)s bring a new dimension to the capabilities of FL. Inspired by the biological processes of the human brain, SNNs operate on an event-driven basis, processing information only in response to stimuli. This method of operation significantly reduces the power consumption of these networks, making them particularly suited for deployment across distributed, battery-operated devices. The integration of SNNs with FL leverages these unique advantages, combining energy efficiency with the privacy-preserving features of FL to address the challenges faced by traditional artificial neural network models in federated settings [2, 3].

The energy-efficient nature of SNNs aligns perfectly with the objectives of FL, where models are trained across a network of distributed devices without centralized data collection. Unlike conventional artificial neural networks that require continuous data flow and computation, SNNs' event-driven operation allows for significant reductions in energy consumption, which is critical for battery-operated agents participating in FL. This characteristic of SNNs supports the development of low-power, distributed learning solutions, enabling more agents to participate in FL without compromising on power efficiency [4].

Furthermore, the integration of SNNs in FL scenarios can also help overcome limitations associated with bandwidth-constrained environments. The sparse nature of data representation and communication in SNNs means that less information needs to be exchanged between devices and the central server during the training process. This efficiency is crucial in FL environments, where network bandwidth and connectivity can significantly impact the feasibility and performance of distributed learning systems [5].

Another critical advantage of SNNs in the context of FL is their compatibility with neuromorphic hardware. Neuromorphic chips, designed to replicate the neural structures of the human brain, provide an ideal platform for deploying SNNs. This synergy between neuromorphic computing and SNNs paves the way for the development of highly efficient, scalable, and adaptive FL systems capable of leveraging the full potential of edge computing [6].

Despite these advantages, the integration of SNNs with FL presents several challenges. The primary hurdle is the complexity of training SNNs in complex problems. The dynamic and temporal nature of SNNs introduces new challenges in developing effective FL protocols that can accommodate the unique online learning rules of SNNs. Proper scheduling of communication rounds within the local SNN time steps is essential for successful collaborative training [7].

Moreover, managing the performance trade-offs associated with the frequency of communication rounds in FL is another significant challenge. Experiments have shown the impact of the number of time steps between local updates and the frequency of model aggregation on the training performance of SNNs. Finding an optimal balance to maximize model performance while minimizing communication overhead is crucial for the efficient deployment of FL systems powered by SNNs [8].

Additionally, communication constraints and the non-stationarity of data distribution pose significant challenges. The need for larger model update intervals to reduce communication costs and the changing nature of data over time (e.g., reward function in RL) require innovative solutions to maintain the accuracy and reliability of FL systems employing SNNs [9].

Despite these challenges, the potential benefits of integrating SNNs with FL justify continued research and development in this area. The combination of energy efficiency, privacy preservation, and compatibility with neuromorphic hardware, along with the distributed and collaborative nature of FL, represents a compelling approach to deploying machine learning models in real-world applications [10].

1.1 Spiking Neural Networks: Models and Learning Algorithms

1.1.1 Neuron model

In the study of computational neuroscience and the development of neural networks, particularly SNN, various neuron models have been proposed to simulate the electrical activity of neurons. These models range from simple to complex, aiming to capture the essential features of neuronal dynamics. The diagram in Figure 1.2 illustrates a basic network consisting of spiking neurons. In this network, the pre-synaptic neuron (pre-neuron) in the input layer spikes in response to the input (I) and transmits these spikes through synaptic weights to the post-synaptic neuron (post-neuron). Whenever the pre-neuron spikes, the post-neuron receives input current with a value of W.

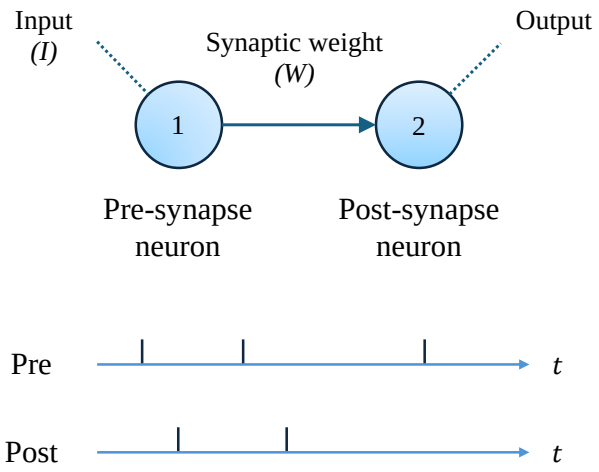

Figure 1.2: Simple model of a SNN. The spike pattern shows that the neurons spike whenever the voltage of the neruon reaches a threshold.

Table 1.1 shows the differential equations used in the network. In SNNs, the activation function, like ReLu and tanh in regular neural networks, is replaced by a differential equation that mimics the biological neuron activity.

	Spike Response Function
Pre-Neuron	$\dot{V}_1 = \frac{1}{\tau_m}\left[E_L - V_1 + R_m I\right]$
Post-Neuron	$\dot{V}_2 = \frac{1}{\tau_m}\left[E_L - V_2 + R_m\left(W\delta(t - t_{\text{pre}})\right)\right]$

Table 1.1: Spike Response Function of the neurons in an SNN.

Several spike response functions have been proposed to emulate neuron activity. In this chapter, the Leaky-Integrate and Fire model is represented because it is simple and accurate enough in emulating real neurons. This section overviews the neuron model and its characteristics.

Leaky-Integrate and Fire model

The Leaky Integrate-and-Fire (LIF) model is a biological model that can be represented as a circuit with a resistor and capacitor and represents a first-order dynamic system [11],

$$R_m C_m \frac{dV_m(t)}{dt} = E_l - V_m(t) + R_m I(t) \tag{1.1}$$

where $V_m(t)$ is the neuron's membrane potential shown as V_1 and V_2 in Table 1.1, R_m is the membrane resistance, C_m is the membrane capacitance, E_l is the resting potential, and $I(t)$ is the input current. The neuron spikes when its potential reaches the threshold potential (V_{th}). The potential of the neuron immediately reaches the reset potential (V_{res}) after it spikes.

The spike rate is a parameter that determines how fast the neuron spikes [12].

$$r_{[Hz]} = \frac{1}{t_{isi}\ \ [s]} \tag{1.2}$$

where t_{isi} is the inter-spike interval that can be calculated using the neuron model, when the potential of a neuron reaches the threshold potential, it fires. Therefore, based on the analytical solution of (1.1), the inter-spike interval time can be written as,

$$t_{isi} = \tau_m \ln\left(\frac{E_l + R_m I - V_{res}}{E_l + R_m I - V_{th}}\right) \tag{1.3}$$

where τ_m is the membrane time constant.

According to (1.3), the following condition should be satisfied to have a finite value for t_{isi},

$$E_l + R_m I - V_{th} > 0 \tag{1.4}$$

or

$$I > \frac{V_{th} - E_l}{R_m} \tag{1.5}$$

which means that the input current higher than the above value generates spikes.

After calculating the minimum input for neurons, we must find the maximum input based on the inter-spike interval. Equation (1.3) can be written as,

$$t_{isi} = \tau_m \ln\left(1 + \frac{V_{th} - V_{res}}{E_l + R_m I - V_{th}}\right) \tag{1.6}$$

Equation (1.6) can be approximated using the Maclaurin series for the natural logarithm function ($\ln(1 + z) \approx z$) as follows,

$$t_{isi} = \frac{\tau_m \left(V_{th} - V_{res}\right)}{E_l + R_m I - V_{th}} \tag{1.7}$$

Solving for I, an input current as a function of the inter-spike interval can be obtained,

$$I = \frac{\tau_m \left(V_{th} - V_{res}\right)}{t_{isi} R_m} + \frac{V_{th} - E_l}{R_m} \tag{1.8}$$

The maximum value for the input current makes the neuron fire at each sample time (Δt). Therefore, the maximum input current is,

$$I^{max} = \frac{\tau_m \left(V_{th} - V_{res}\right)}{\Delta t R_m} + \frac{V_{th} - E_l}{R_m} \tag{1.9}$$

In this section, we obtained the minimum and maximum values for input current using (1.5) and (1.9). These equations are used in the learning and encoding processes of the SNN.

1.1.2 Learning Approaches in SNNs

Hebbian Learning

Hebbian Learning is a fundamental neural learning principle summarized by the axiom "neurons that fire together, wire together," describing how simultaneous activation of neurons leads to strengthened connections between them [13]. Hebbian learning, particularly within the context of SNN, primarily revolves around the modulation of synaptic strengths based on the firing rates of pre- and postsynaptic neurons. The principles of locality and joint activity are fundamental, emphasizing that synaptic changes occur only when both neurons are active simultaneously.

Methods of Synaptic Modification:

- **Local Rules:** Synaptic changes are influenced directly by the activities of the connecting neurons without external influences.
- **Bounded Growth:** To avoid uncontrolled increases in synaptic strength, models typically incorporate mechanisms such as hard and soft bounds. Hard bounds prevent any further increase once a maximum weight is achieved, while soft bounds slow down the rate of increase as the maximum is approached [14].
- **Synaptic Decay:** Realistic models also consider mechanisms for reducing synaptic strengths, typically through a decay term that weakens connections in the absence of activity [15].

Advanced Hebbian Models:

- **Covariance Rule:** This model refines the synaptic modification to depend on the deviation of firing rates from their means, enhancing the dynamic response of synapses to changes in neural activity [16].

- **Oja's Rule:** A self-stabilizing rule that ensures synaptic weights do not grow indefinitely by normalizing the weight vector, thereby maintaining the overall stability of the network [17].

- **BCM Rule:** The Bienenstock-Cooper-Munro rule introduces an adaptive threshold for synaptic modification, which evolves based on the historical activity of the neuron, allowing for more refined potentiation and depression based on relative activity levels. [18].

- **RCHP:** Rarely Correlating Hebbian Plasticity focuses on synaptic changes driven by rare, significant coincidences in neuronal activity, aiming to strengthen connections that are crucial for neural function while avoiding over-strengthening due to common activity patterns [19].

Incorporation into Reinforcement Learning: Modern adaptations in SNNs integrate the concept of rewards, adding a third dimension to synaptic adjustments. This integration uses scaling or gating mechanisms in response to global reward signals, further refining the learning capabilities of neural networks based on external feedback.

Neo-Hebbian Learning and Modulation Mechanisms

In neo-Hebbian reinforcement learning, significant advancements come from a global reward signal modulating synaptic plasticity alongside an eligibility trace that decays over time [20]. This trace increases with recent, successful neurotransmission and decreases as time passes, linking closely with Temporal-Difference (TD) learning mechanisms. Such dynamics allow for Long-Term Potentiation (LTP) or Depression (LTD) at synapses based on the timing of synaptic activity and the nature of the reward signal. Specifically, recent successful activities followed by positive rewards enhance LTP, while activities preceding negative rewards lead to LTD.

Distal rewards and credit assignment

This model introduces a sophisticated approach to synaptic modification through the interaction of Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP) with a modulatory signal reflective of reward, embodying the essence of TD learning within the realm of spiking neurons.

Central to this framework is the eligibility trace mechanism, elegantly adapted from its conventional application in TD learning to facilitate synaptic credit assignment over varying temporal extents. This adaptation allows for the dynamic modulation of synaptic strengths based on the timing and sequence of pre- and post-synaptic spikes, in conjunction with the temporal dynamics of received rewards. The eligibility trace is mathematically represented as follows [21]:

$$\frac{dC_{ji}}{dt} = -\frac{C_{ji}}{\tau_C} + \text{STDP}(t_{\text{post}} - t_{\text{pre}})\delta(t - t^{(f)}) \tag{1.10}$$

Here, $C_{ji}(t)$ denotes the eligibility trace for the synapse between pre-synaptic neuron j and post-synaptic neuron i, evolving with a decay governed by τ_C. The Dirac delta function, $\delta(t - t^{(f)})$, signifies the occurrence of a spike, serving as a pivotal factor in the temporal credit assignment process.

The STDP function can be represented as follows,

$$STDP(\tau) = \mathcal{A} \exp\left(-\frac{\tau}{\tau_s}\right) \text{ for } \tau \geq 0, (\tau = t_{\text{post}} - t_{\text{pre}}) \tag{1.11}$$

where $\mathcal{A}$ stands as the amplitude and τ_s acts as the time constant.

Synaptic weight updates are then guided by the interaction between the eligibility trace and the reward signal, $R(t)$, as captured in the following equation:

$$\frac{dw_{ji}}{dt} = C_{ji}(t) \cdot R(t) \tag{1.12}$$

where dw_{ji}/dt symbolizes the rate of change in synaptic weight, contingent upon the compounded influence of the eligibility trace and the reward signal.

1.2 Literature Review

1.2.1 Spiking Neural Network

SNNs have garnered significant attention for their ability to emulate complex neural dynamics observed in biological systems. This literature review focuses on the learning mechanisms, including STDP, and their implications in modeling biological learning behaviors.

Besides the LIF model, there are different types of SNN models. The Hodgkin-Huxley model is a foundational framework in neuroscience that meticulously describes the ionic processes critical for the initiation and propagation of action potentials in neurons [22]. The model uses differential equations to capture the dynamics of membrane potential (V) influenced by various ion-specific currents and gating variables, which regulate ion channel states. These gating variables, which transition between 0 and 1, reflect the proportion of ion channels in different states, crucial for mimicking the action potentials' temporal complexity. In contrast, the FitzHugh-Nagumo model simplifies the complex Hodgkin-Huxley framework into a two-variable system, focusing on capturing the essential characteristics of neuronal excitability with fewer computational demands.

The Izhikevich model merges the biological realism of the Hodgkin-Huxley model with the computational simplicity of integrate-and-fire models, offering a balanced approach for simulating neuronal behavior [23]. This model is particularly noted for its ability to produce diverse firing patterns with computational efficiency, making it suitable for simulating large networks of neurons while still capturing key aspects of neuronal activity.

The learning in SNNs is based on the timing between neurons firing. The STPD process is one of the biological processes observed in real neurons. The investigation of competitive Hebbian learning through STDP reveals how the timing of spikes modulates synaptic efficacy, underscoring the significance of temporal dynamics in synaptic modification within neural circuits [24]. This highlights the critical role of spike timing in learning processes, which is essential for understanding neural adaptation.

In a significant study, the distal reward problem was tackled by linking STDP with

dopamine signaling, illustrating a novel approach to reward-based learning in the brain [21]. This study proposed a model where dopamine modulates STDP mechanisms, effectively capturing the essence of reward-based learning processes.

Another research posited that STDP emerges from fundamental learning rules governed by intracellular calcium dynamics, suggesting a deeper biophysical basis for synaptic strength regulation [25]. This challenges the traditional understanding of STDP and points towards a more comprehensive framework for neural plasticity.

An extensive review of the development and implications of STDP in neural circuits provided insights into its critical role in precise spike timing for synaptic modification, enriching our understanding of the neural basis for learning and memory [26].

A study on multiagent reinforcement learning within the Iterated Prisoner's Dilemma framework highlighted SNNs' comparable or enhanced performance against traditional models [27]. This emphasizes the critical role of spike timing in complex decision-making processes, showcasing SNNs' potential in simulating interactive and competitive environments.

The introduction of a model for parallel path planning using spiking neural activity, inspired by hippocampal navigation strategies, showcased the efficiency of SNNs in solving spatial navigation problems through biologically plausible mechanisms [28].

In a pioneering study, the implementation of probabilistic inference within networks of LIF neurons demonstrated how Bayesian networks can be transformed into a computable framework for SNNs [29]. This methodology underscores SNNs' capability to perform complex cognitive functions through probabilistic reasoning.

The performance comparison between SNNs and multilayer perceptrons in a computer-based racing game highlighted SNNs' superior adaptability and decision-making capabilities in dynamic environments, attributed to the efficient processing of real-time data through spike-based mechanisms [30].

Hierarchical Bayesian inference within SNNs introduces a learning algorithm based on synaptic plasticity, showcasing the ability of SNNs to execute complex tasks like pattern recognition through hierarchical structures [31]. This emphasizes the adaptability and com-

putational efficiency of SNNs.

BP-STDP, which approximates backpropagation using STDP, bridges the gap between SNN mechanisms and traditional neural network computational efficiency [32]. This algorithm advances learning algorithms for SNNs, making them more accessible for machine learning tasks.

Indirect and direct training methods for SNNs in end-to-end control of a lane-keeping vehicle highlight the effectiveness of both approaches in utilizing SNNs for real-world control tasks, providing insights into training SNNs for complex systems like autonomous vehicles [33].

N3-CPL, a neuroplasticity-based learning method for neuromorphic networks, focuses on cell proliferation in SNNs, enhancing learning capabilities through mechanisms inspired by biological neural network growth and adaptation [34].

Parameter optimization in an SNN model for UAV obstacle avoidance tackles the challenge of tuning SNN parameters for specific tasks using optimization techniques, emphasizing the importance of optimization in SNN application for real-time processing and decision-making [35].

A methodology for multi-task autonomous learning in mobile robots using SNNs was proposed, employing Modified Integrate-and-Fire and LIF neuron models, alongside a task switch mechanism inspired by lateral inhibition and a novel learning rule based on R-STDP. This approach demonstrated SNNs' capabilities in efficiently learning and adapting across various tasks such as obstacle avoidance and target tracking [36].

An autonomous learning framework that combines SNNs with a pre-trained binary Convolutional Neural Network for image-based reinforcement learning was developed. This hybrid model efficiently processes high-dimensional sensory data and learns from sparse rewards, highlighting the synergistic potential of SNNs and CNNs in complex visual environments [37].

A transfer learning algorithm for SNNs aimed at deep learning tasks was introduced, utilizing Centered Kernel Alignment for domain distance measurement and focusing on domain-

invariant representation. This methodology enables effective knowledge transfer, showcasing the scalability and adaptability of SNNs [38].

Mathematical constraints influencing Hebbian learning and STDP in SNNs were explored, revealing relationships between synaptic weight promotion/demotion probabilities and normalized weights, offering insights into optimizing SNN training [39].

"Spikepropamine," incorporating differentiable plasticity within SNNs, was presented. This approach enables adaptive learning and improved task performance, signifying advancements in neuromorphic computing and robotics applications of SNNs [40].

NeuronGPU, a library for simulating large-scale networks of spiking neurons with GPU acceleration, was developed. This tool demonstrated scalability and efficiency in simulating complex neuronal dynamics, marking a significant contribution to computational neuroscience tools [41].

1.2.2 FL

FL aggregation methods form a cornerstone of how distributed models learn collectively while maintaining the privacy of local data. The seminal aggregation method in FL is FedAvg, or Federated Averaging. As introduced by McMahan et al. [42], FedAvg operates by averaging the weights of models updated locally by clients. This simple yet effective approach allows a global model to benefit from diverse local updates without sharing the data itself.

Addressing heterogeneity in client data distributions, the FedProx method [43], introduced by Li et al., modifies the local optimization process. FedProx introduces a proximal term which moderates the deviation of local updates from the global model, aiming to stabilize training across clients with non-IID data.

The Scaffold method [44] deals with statistical heterogeneity and client drift, issues prevalent in scenarios where client data is not Independent and Identically Distributed (IID). Karimireddy et al. propose using control variates that correct the local updates toward the global objective, which helps in reducing variance in updates and aligning clients closer to the global model.

Introduced by Wang et al., the FedNova method [45] employs a novel normalization mechanism in the aggregation process that adjusts based on the number of local updates performed by each client. This design accommodates the diversity in client update contributions, particularly beneficial in non-IID data scenarios.

The MOON method, or Model-contrastive Federated Learning, minimizes the contrastive loss between local and global models to maintain consistency across updates [46]. By focusing on minimizing differences in model parameters using a similarity metric, MOON seeks to preserve the quality of the global model despite the diversity of local datasets.

Each of these methods represents an evolution in FL, introducing unique adjustments to the aggregation process to enhance convergence, reduce communication overhead, and improve robustness against adversarial clients and non-IID data conditions.

Building on the foundational principles of FL methods detailed previously, we now turn our focus towards specialized neural network architectures that leverage these FL strategies under unique operational constraints. A particularly interesting area is the utilization of SNNs, which offer a biologically inspired alternative to traditional Artificial Neural Networks (ANNs). SNNs are well-suited for on-device learning in edge computing scenarios due to their efficient energy consumption and capability for processing temporal information. However, the deployment of SNNs in such environments is not without challenges, primarily due to the limited data available on individual devices, which could significantly constrain the learning capabilities.

The limitations imposed by on-device training of SNNs, due to the constrained data availability on each device, is studied in [2]. Based on the results, the limitation can be mitigated through cooperative training utilizing FL. An online FL-based learning rule, termed FL-SNN, for networked on-device SNNs is introduced. Through this scheme, local feedback signals are utilized within each SNN instead of backpropagation, and global feedback is facilitated through communication via a base station, showcasing the potential to significantly enhance training over isolated approaches by enabling a flexible compromise between communication load and accuracy through selective synaptic weight exchange.

A novel approach of selective model aggregation in FL within the context of Vehicular edge computing (VEC) is explored in [47]. This approach focuses on the unique challenges posed by the diversity in image quality and computational capacity of vehicular clients for image classification. The approach, which selectively aggregates 'fine' local DNN models based on an evaluation of local image quality and computation capability without compromising client privacy, employs two-dimensional contract theory for model selection due to the central server's unawareness of these local parameters. This methodology is shown to outperform conventional federated averaging in terms of accuracy and efficiency on the MNIST and BelgiumTSC datasets, also offering higher utility at the central server.

The paper by [48] discusses a novel approach for fast global model aggregation in FL via cloud computation, aiming to address the primary bottleneck of limited communication bandwidth in the aggregation of locally computed updates across devices with strict latency and privacy requirements, like drones and smart vehicles. This approach, employing a sparse and low-rank optimization problem solution through a Difference-of-Convex-functions algorithm for joint device selection and beamforming design, demonstrates significant algorithmic advantages and performance improvements in numerical results.

According to [49], FL is suggested as a solution to the challenges faced in training machine learning models on IoT data due to constraints in network bandwidth, storage, and privacy. The focus is on how existing work has primarily centered on learning algorithms' convergence time, leaving issues like incentive mechanisms largely unexplored. A deep reinforcement learning-based incentive mechanism is proposed to motivate edge nodes towards model training contribution, with its efficiency validated through numerical experiments.

In [4], the exploration of synergies between wireless communications and artificial intelligence, particularly the challenges of implementing ML models on battery-powered devices connected via bandwidth-constrained channels, is discussed. The paper highlights how FL for distributed training of SNNs and the integration of neuromorphic sensing with SNNs can help overcome these challenges.

The research presented in [1] introduces a FL method designed for training decentral-

ized and privacy-preserving SNNs, focusing on the energy efficiency advantages of SNNs in FL scenarios across energy-constrained devices. The method's effectiveness is experimentally validated through improved accuracy and energy efficiency in comparison to ANNs on CIFAR10 and CIFAR100 benchmarks.

A comprehensive study on deploying distributed learning over wireless edge networks, highlighting the challenges and emerging paradigms like FL and distributed inference, is provided in [50]. It presents an overview of communication techniques for efficient deployment and future research opportunities within wireless networks.

The effect of user selection and resource allocation on the convergence time and accuracy of FL over wireless networks was investigated in [51]. A probabilistic user selection scheme is proposed alongside the use of ANNs to estimate unselected users' local FL models, showing a reduction in convergence time and improvement in model accuracy through simulations.

Federated deep learning approaches for cybersecurity in IoT applications are reviewed and experimentally analyzed in [52]. The article highlights the superiority of FL in preserving IoT device data privacy and enhancing attack detection accuracy across various deep learning models.

The challenge of training FL algorithms over wireless networks is studied in [53], where the impact of wireless factors on training quality and the optimization of learning, wireless resource allocation, and user selection to minimize FL loss function are discussed. Simulation results demonstrate the effectiveness of the proposed framework in improving identification accuracy.

In [54], an investigation is conducted into the allocation of energy-efficient transmission and computation resources for FL over wireless communication networks. The model considered involves users training local FL models with their data and sending these to a base station (BS), which then aggregates these models and broadcasts the aggregated model back to the users. The optimization problem formulated aims to minimize total energy consumption under a latency constraint, with an iterative algorithm proposed that derives closed-form solutions for various parameters at every step. Initial feasible solutions are

generated through a bisection-based algorithm aimed at minimizing completion time. It is shown through numerical results that energy consumption can be reduced by up to 59.5% compared to conventional FL methods.

In [9], a lead federated neuromorphic learning (LFNL) technique is proposed, designed as a decentralized, energy-efficient, brain-inspired computing method utilizing SNNs. This technique allows edge devices to collaboratively train a global model, preserving privacy and significantly reducing both data traffic and computational latency, with only a slight accuracy loss reported. The efficacy of LFNL in reducing energy consumption while maintaining competitive recognition accuracy, despite uneven dataset distribution, is demonstrated through experimental results.

The aggregation strategies in FL through a comprehensive mathematical convergence analysis, leading to the derivation of novel aggregation algorithms that adapt model architecture by differentiating client contributions based on loss values, is presented in [55]. This approach, which extends beyond theoretical assumptions to practical performance evaluations, is compared with the conventional FedAvg method across both IID and non-IID settings without additional hypotheses.

An edge-based backhaul (BH) selection technique aimed at improving traffic delivery by leveraging multiobjective feedback, utilizing advantage-actor-critic deep reinforcement learning methods, is introduced in [56]. To enhance DRL training performance in large-scale IoT system deployments, FL is applied, enabling collaboration among multiple edge devices. The effectiveness of this federated DRL approach in solving the BH selection problem is verified through extensive simulations.

In [57], a hierarchical trajectory planning method named HALOES, which employs deep reinforcement learning within a FL scheme, is proposed for efficient, automatic parking in narrow spaces. This method combines high-level deep reinforcement learning with low-level optimization, improving planning time and model generalization capabilities while protecting data privacy during model parameter aggregation. Simulation results affirm HALOES's efficiency in various narrow-space parking scenarios.

The hierarchical FL paradigm of HiFL is discussed in [58], which aims to mitigate communication overhead in FL systems by integrating synchronous client-edge and asynchronous edge-cloud model aggregations. An enhanced design, HiFlash, incorporates adaptive staleness control and a heterogeneity-aware client-edge association strategy through deep reinforcement learning, demonstrating improved system efficiency, communication reduction, and maintained model accuracy in experiments.

A survey on poisoning attacks and defense strategies in FL from a privacy-preserving perspective, classifying and analyzing poisoning attacks and defense mechanisms, is provided in [59]. The paper highlights the urgency of defending against poisoning attacks and suggests potential research directions for both attack and defense strategies, emphasizing the need for systematic reviews in this area.

Chapter 2

Modular Learning in SNNs for Optimal Multi-Agent Decision-Making

2.1 Introduction

This chapter looks at two main things: how well SNNs handle noise and how they can handle complex scenarios like optimal decision-making. This detailed study shows us the potential of SNNs in managing complex behaviors even when there are outside disturbances and their capability to separate the reward function for the neural network's different parts.

The neural structure in SNNs helps us implement complex learning systems. One good example of a complex situation is a differential game like the Active Target Defense (ATD) problem [60].

In the ATD, a defender tries to protect a target, while a superior invader with higher velocity than the defender and a moving target tries to reach the target and escape the defender at the same time. In the context of defense strategy, aircraft movement analysis is crucial in determining the most effective strategies. In the context of defense strategy, aircraft movement analysis is crucial in determining the most effective strategies. Law enforcement benefits from these games by minimizing escape possibilities. The ATD problems are a fundamental component of game theory, providing insights into strategic decision-making

across disciplines like economics and biology. Additionally, these games find applications in Cybersecurity for modeling attacker-defender interactions [61].

There are two distinct outcomes to the ATD game, characterized by two termination sets [62]. The first outcome happens when the invader reaches the target while the defender is far from it. The second outcome happens when the defender reaches the target while the invader's distance from the target is larger than the defender's distance.

2.2 The ATD problem and SNN-based solution

The ATD problem has various solving methods, such as Apollonius Circle and Cartesian Ovals (CO). This chapter opts for the CO method over the Apollonius Circle because it considers the capture radius and effectiveness against superior invaders [62]. The optimal capture point is considered the minimum distance between the target's position and reachable region if the target is inside the defender's dominant region. The defender's dominant region is a region where the defender can reach the target without letting the invader capture the target.

In this chapter, the problem is solved using reinforcement learning. It is considered that each agent knows the relative velocity of other agents. Figure 2.1 shows LOS angles used as the input for the SNNs [63].

The R-STDP algorithm is used to train two separate SNNs simultaneously that control the invader and defender. The target in this chapter moves in the environment, and the SNNs receive the LOS angles (e.g., the defender receives the LOS angle to both the invader and the target) and calculate the steering angle for the agent (Figure 2.2). In this figure, the ϕ_I^T represents the LOS angle to the target relative to the invader.

2.3 Learning using R-STDP

R-STDP is a biological learning algorithm that is believed to underlie certain learning mechanisms in the brain [64]. The R-STDP algorithm is based on the idea that if a pre-synaptic

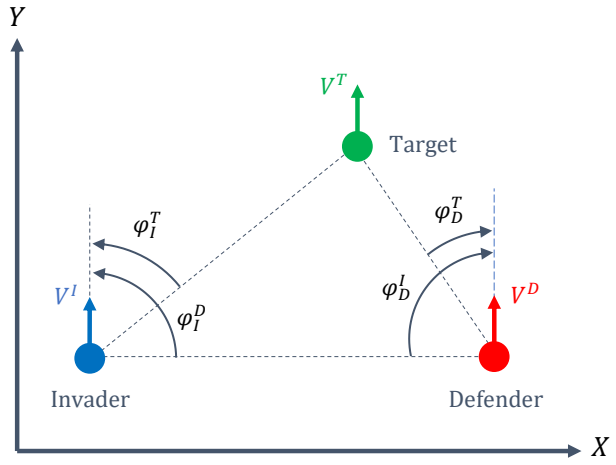

Figure 2.1: Active target defense game with three agents (LOS angles are shown for both agents).

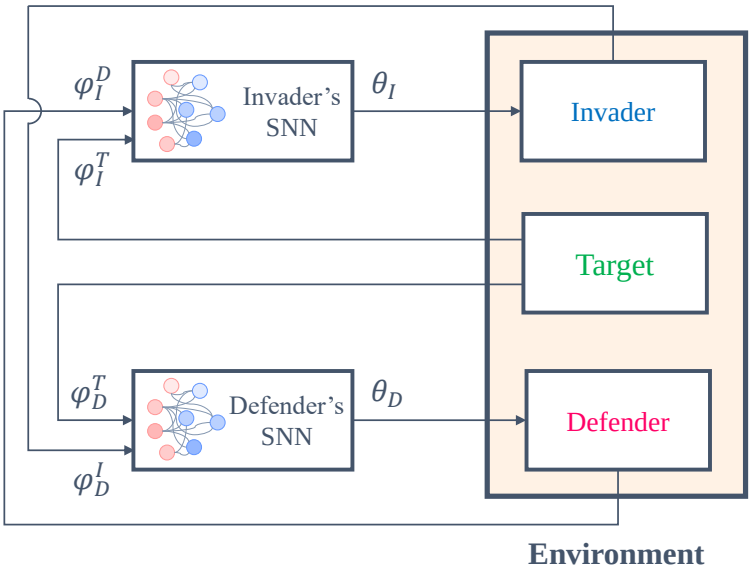

Figure 2.2: Two SNNs simultaneously play and control the invader and defender. Each agent uses the LOS angle and relative velocities for training.

neuron fires just before a post-synaptic neuron and the network receives a reward, the strength of the synapse between the two neurons should be increased.

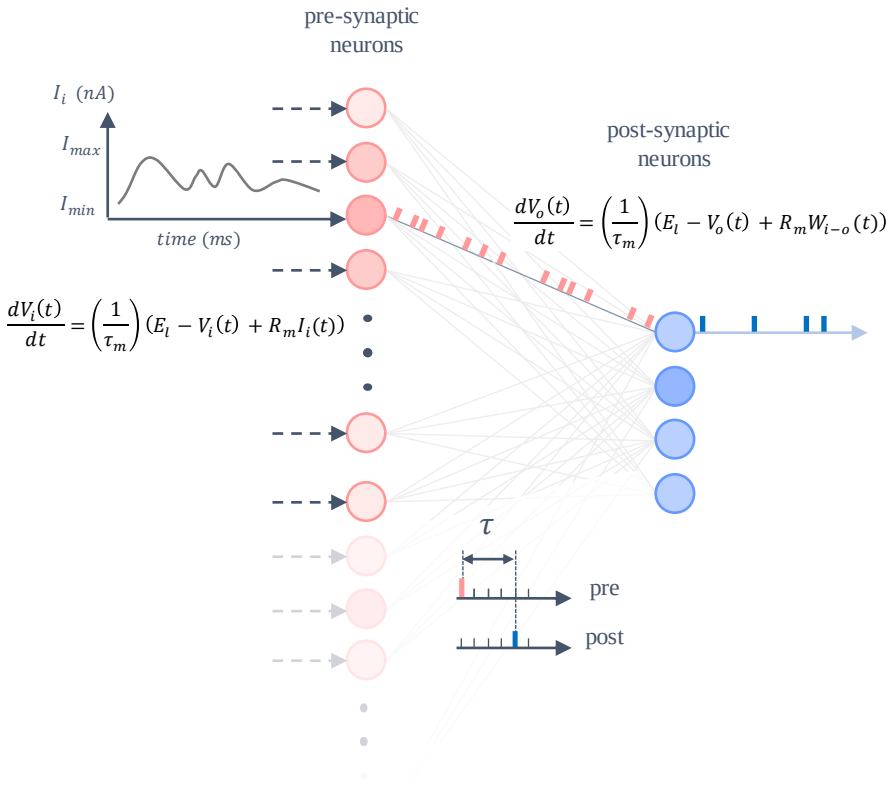

Figure 2.3: An illustration of input current to SNN and synaptic current to the output neurons after training for a single connection.

In the SNN, the pre-synaptic neurons are input neurons, and post-synaptic neurons are output neurons. The synaptic plasticity is referred to as an eligibility trace (C), which is calculated based on the following equation [21],

$$\dot{C}^{i-j} = -C^{i-j}/\tau_C + STDP^{i-j}(\tau)\delta(t - t_{pre/post}) \tag{2.1}$$

where the C^{i-j} is the eligibility trace for the synaptic connection between neurons i and j, τ, is the spike timing difference between the input and output spike times, τ_C is the time constant for the synaptic plasticity, δ is the Dirac function, $t_{pre/post}$ is the firing time of the pre or post-synaptic neuron, and $STDP(\tau)$ is a function of the firing time of the input and output neurons as follow,

$$STDP^{i-j}(\tau) = \begin{cases} A_+ exp\left(\frac{-\tau}{\tau_s}\right) & \text{if } \tau > 0 \\ A_- exp\left(\frac{\tau}{\tau_s}\right) & \text{if } \tau < 0 \end{cases} \tag{2.2}$$

where A_+ and A_- are the amplitude of the exponential function, and τ_s is the time constant that determines the decaying rate of the R-STDP function. If τ_s goes to infinity, the exponential function becomes 1, and the effect of time in R-STDP will be lost.

The synaptic weights are changed according to the following equation,

$$\dot{W}^{i-j}(t) = C^{i-j}(t)R(t) \tag{2.3}$$

where W^{i-j} is the synaptic weight between neurons i and j, which is the amount of input $(I(t))$ that the post-synaptic neuron receives when the pre-synaptic neuron spikes, and $R(t)$ is the reward.

This chapter uses the Multiplicative Synaptic Normalization (MSN) method to keep runaway excitation under control. This method keeps pre-existing memories in the network by conserving the proportional difference between smaller and larger synaptic weights. According to (1.9), we can calculate the maximum input for each post-synaptic neuron (output layer).

The MSN normalizes the synaptic weights based on the cumulative input synaptic weights and the maximum input as follows [64],

$$\overrightarrow{W}^n(t) = \overrightarrow{W}^n(t-1)\left(\frac{I^{max}}{\Sigma_{\iota=1}^{N} W^{\iota}(t)}\right) \tag{2.4}$$

where $\overrightarrow{W}^n(t)$ is the vector consisting of synaptic weights that send input current to the n^{th} output neuron, and N, is the total number of input synapses for each output neuron. Therefore, when R-STDP increases a single synaptic weight, the MSN proportionally decreases the other synaptic weights for the n^{th} neuron.

The reward for the invader and defender is defined based on the projection of the velocities along the LOS direction. Figure 2.4 shows the projected velocities for the invader and

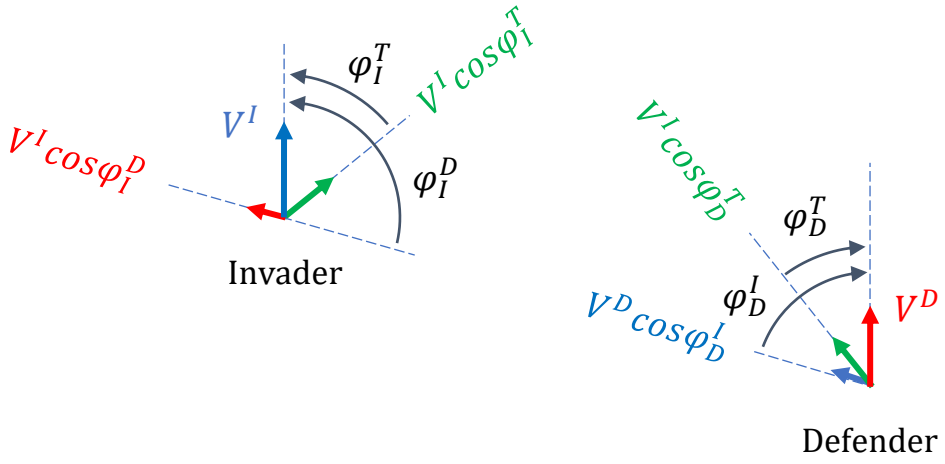

Figure 2.4: Relative velocities and LOS angles used in the reward function

defender, where both agents measure the relative velocities from each other. It means that the invader receives a negative reward when it moves toward the defender and receives a positive reward when it moves toward the target. The defender receives a positive reward when it moves toward the target and invader. Since the velocities are constant, the reward value depends only on the headings that cause the change in relative velocities. Therefore, the LOS toward the other agents determines the reward value.

The reward for the invader considering the target and defender consists of two parts,

$$R_I^T(t) = \eta_I^T \left(V^I + V^T\right) cos\left(\phi_I^T\right) \tag{2.5}$$

and

$$R_I^D(t) = \eta_I^D \left(V^D - V^I\right) cos\left(\phi_I^D\right) \tag{2.6}$$

The reward for the defender also can be calculated as follows,

$$R_D^T(t) = \eta_D^T \left(V^D + V^T\right) cos\left(\phi_D^T\right) \tag{2.7}$$

and

$$R_D^I(t) = \eta_D^I \left(V^D - V^I\right) cos\left(\phi_D^I\right) \tag{2.8}$$

In (2.5)-(2.8), η_I^T, η_I^D, η_D^T, and η_D^I are constant coefficients. Adjusting these values changes the agent's attention to other agents. For example, in the invader case, increasing the η_I^T and decreasing the η_I^D increases the effect of the target on the output and reduces the defender's effect. The invader then places more importance on getting to the target than evading the defender.

The change in synaptic weights for the invader regarding the target and defender consists of two parts as follows (the same process is true for the defender),

$$\mathbf{W}_I^T(t) = \mathbf{W}_I^T(t-1) + \mathbf{C}_I^T(t) R_I^T(t) \tag{2.9}$$

and

$$\mathbf{W}_I^D(t) = \mathbf{W}_I^D(t-1) + \mathbf{C}_I^D(t) R_I^D(t) \tag{2.10}$$

where $\mathbf{W}_I^T$ is a $k \times l$ matrix that represents the synaptic weights corresponding to the target (k is the number of output neurons and l is the number of input neurons for the target). The $\mathbf{W}_I^D$ is a $k \times m$ matrix that represents the synaptic weights regarding the defender (m is the number of input neurons for the defender). The eligibility trace matrices $\mathbf{C}_I^T$ and $\mathbf{C}_I^D$ are of dimension $k \times l$ and $k \times m$, respectively.

2.4 Network structure and encoding method

Figure 2.5 shows the defender's network structure and encoding process. The invader has the same network with different inputs. The network receives the LOS angles and converts the inputs to Fuzzy Membership Values (FMV) using Gaussian Receptive Fields (GRF) [65]. Since the fuzzy membership values are used just for encoding data into the SNN, the type of the membership function does not affect the computation complexity.

There are q input neurons, and a membership function is assigned to each neuron. There-

fore, there are q membership functions. The acquired fuzzy membership values are real numbers between 0 and 1 and should be converted to the input currents. This can be done using a linear function and the minimum and maximum inputs.

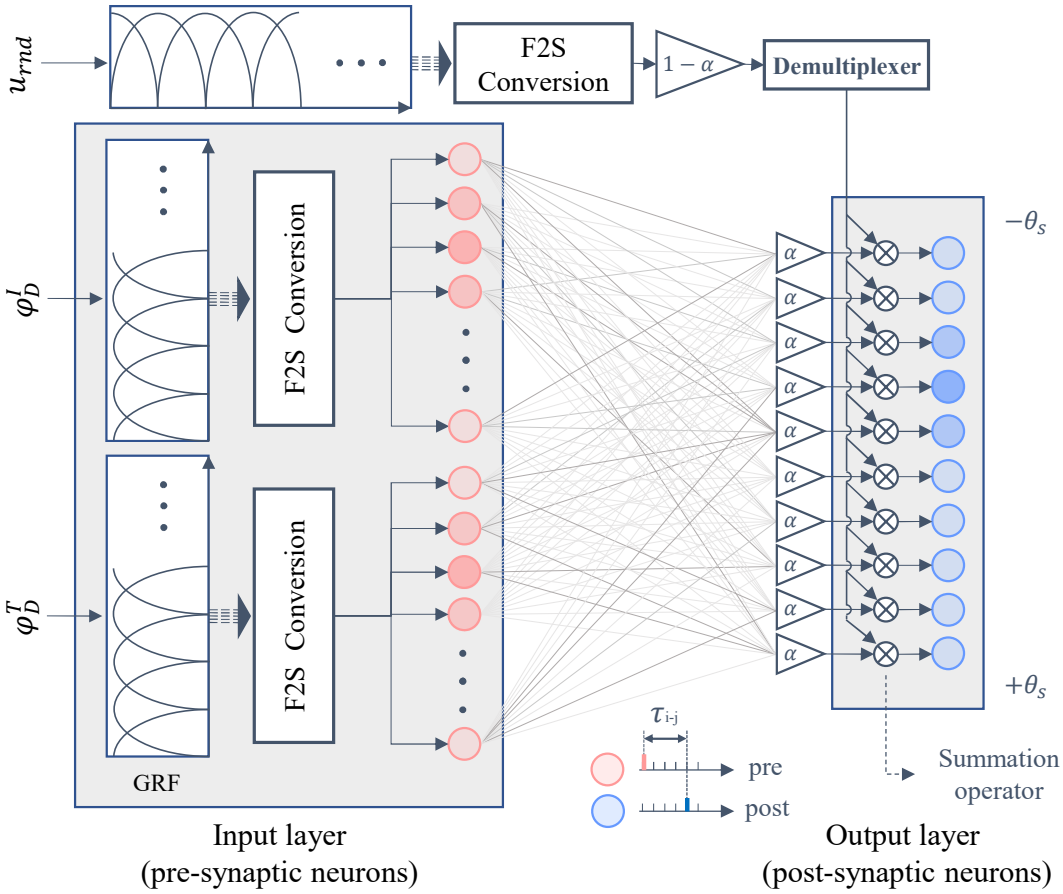

Figure 2.5: Network structure and encoding process for the input layer (Defender). Each neuron is associated with a membership function in GRF. The GRF encodes an input State (S^t) at each time step. There is both a training phase when ($\alpha = 0$) and an operating phase when ($\alpha = 1$).

The acquired fuzzy membership values are converted to the spiking inputs for the neurons using fuzzy-to-spiking (F2S) conversion. If $FMV = 0$ ($t_{isi} = \infty$), then the input to the desired neuron in the input layer is I^{min}, and if $FMV = 1$ ($t_{isi} = \Delta t, \Delta t$ is the sampling time), then the input is I^{max}. Therefore, the input current for the input neurons can be calculated using the following equation,

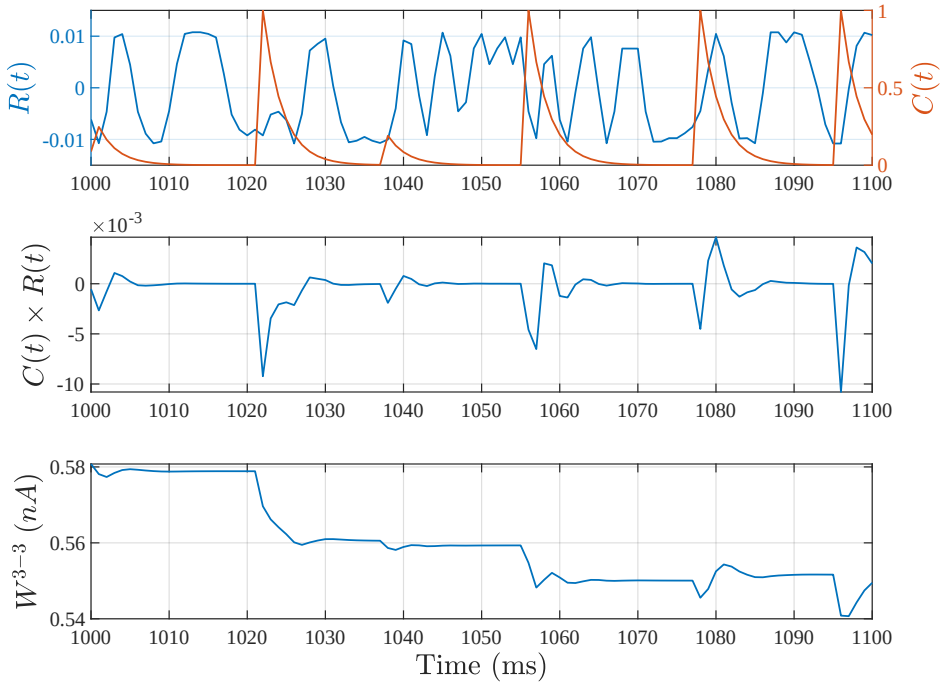

Figure 2.6: Reward, eligibility trace, and weight change during the simulation for W^{3-3}. The $C(t)R(t)$ changes the synaptic weight by considering activation strength and reward value.

$$I_\sigma = \left(I^{max} - I^{min}\right) FMV_\sigma + I^{min}$$

or

$$I_\sigma = \frac{\tau_m \left(V_{th} - V_{res}\right)}{\Delta t R_m} FMV_\sigma + \frac{V_{th} - E_l}{R_m} \tag{2.11}$$

where σ is the index of each neuron in the input layer and its corresponding fuzzy membership value in GRF.

The output of the SNN is the steering angle (θ_s) for the agent. The output is calculated using the weighted average method. Each neuron in the output layer represents a specific steering angle. The number of spikes for each output neuron in τ_s millisecond represents how much it contributes to the output. The contribution level is considered 1 for an output neuron that fires at each step time, while it is considered 0 for the output neuron that has not fired. Levels of contributions are then multiplied by the angle that each output neuron

Algorithm 1 Weight training algorithm

for $t = 0 : \Delta t : t_{final}$ **do**
  input ϕ_I^T and ϕ_I^D
  $\zeta \leftarrow$ number of input neurons
  $\kappa \leftarrow$ number of output neurons
  for i = 0:$2\pi/\zeta$:2π **do**
    $Z_T(i,1) = exp\left(-0.5(\phi_I^T - i)/\sigma\right)$
    $Z_D(i,1) = exp\left(-0.5(\phi_I^D - i)/\sigma\right)$
  end for
  $I^T = (I^{max} - I^{min})\, Z_T + I^{min}$
  $I^D = (I^{max} - I^{min})\, Z_D + I^{min}$
  Generate a uniform random number for exploration $u_{rnd} \in [-\theta_s, \theta_s]$
  $j = 1$
  for i = $-\theta_s$:$2\theta_s/\kappa$:θ_s **do**
    $Z_{rnd}(j,1) = exp\left(-0.5(u_{rnd} - i)/\sigma\right)$
    $j \leftarrow j + 1$
  end for
  $I_{rnd} = (I^{max} - I^{min})\, Z_{rnd} + I^{min}$
  for j = 1 to κ **do**
    $I_{syn}(j,1) = \sum_{i=1}^{\zeta} W^{i-j}\delta(t - t_i)$ ▷ t_i is the firing time of the i^{th} input neuron
  end for
  $I_{in} = \begin{bmatrix} I^T \\ I^D \end{bmatrix}$ Input current
  $I_{out} = \alpha I_{syn} + (1 - \alpha) I_{rnd}$ ($\alpha = 0$ in training phase)
  $\mathbf{V_m}(t + \Delta t) = (1 - \frac{\Delta t}{\tau_m})\mathbf{V_m}(t) + \frac{\Delta t}{\tau_m}\left(\mathbf{E_l} + R_m \begin{bmatrix} I_{in} \\ I_{out} \end{bmatrix}\right)$ {$\mathbf{V_m}$ and $\mathbf{E_l}$ are $[\zeta + \kappa] \times 1$}
  Find fired neurons in input and output layer
  Calculate the $\boldsymbol{\tau}$ matrix that shows the difference in firing time between the fired input neurons and fired output neurons (Figure 2.5)
  Calculate $\mathbf{R - STDP}$ for all connection using (2.2)
  Calculate $\mathbf{C}$ matrix using (2.1) for all the connections
  Calculate $\mathbf{W}$ matrix using (2.3) for all the connections considering reward from (2.5) to (2.8)
  for i = 1 to κ **do**
    if sum of the input weights to ith neuron $\geq I^{max}$ **then**
      Normalize input weights of ith neuron using (2.4)
    end if
  end for
  Set voltage of the fired neurons to reset voltage (V_{res})
end for

represents. Finally, the summation of all the calculated terms is divided by the summation of all levels of contributions.

The output consists of two terms. One term comes from the synaptic weights, and the other term is random noise for exploration. The output of the SNN can be shown as follows,

$$I_{out} = \alpha I_{syn} + (1 - \alpha) I_{rnd} \tag{2.12}$$

where α is a constant that is 0 during training and becomes 1 after training is completed, I_{rnd} is a random steering angle that is selected at each time step, and I_{syn} is the synaptic output (steering angle based on synaptic weights). Therefore, there are two phases: a training phase and an operating phase.

During training, the input State (S^t) is encoded into the network, and a random steering angle (u_{rnd}) is encoded as a random action using the F2S process into the output layer. These two encoding currents for the input and output layer make input and output neurons fire independently. The R-STDP adapts the weights for the fired neurons. Since α is 0 during the training, the I_{syn} does not affect the SNN's output (Equation 2.12).

In the training process, the output and input neurons are excited separately. The input neurons are fired based on the agent's current state, whereas the agent's steering angle is randomly assigned based on the random input to the output neurons, as shown in Figure 2.6. The agent then takes a step based on the random steering angle and a reward is assigned. The training algorithm then evaluates the reward for that given random steering angle. If the reward is positive, then the weight associated with the input neurons to output neurons that fired for that state is strengthened, and if the reward is negative, then the weight for the input to output neurons in (2.2) is weakened. Future research will include an inhibitory effect where the weights can become negative.

After training, the α changes to 1 and eliminates the effect of random output, and the SNN's output is calculated based on the synaptic currents. Algorithm 1 shows the training process.

2.5 Results

A numerical simulation is conducted to evaluate the SNN's performance in solving the ATD problem. The simulation is done in MATLAB 2022a, with a 1 ms sample time. The simulation parameters for neurons are presented in Table 2.1.

Table 2.1: Parameter values for LIF neuron model [10]

Parameter	**Value**	**Description**
R_m	40 MΩ	Membrane Resistance
τ_m	30 ms	Membrane time constant
E_l	-70 mV	Resting potential
V_{res}	-70 mV	Reset potential
V_0	-70 mV	Initial membrane potential
V_{th}	-50 mV	Threshold membrane potential

Table 2.2: Parameter values for R-STDP

Parameter	**Value**	**Description**
τ_s	3 ms	Time constant
$A_{\pm}$	1	Amplitude of the R-STDP function
η_D^T	0.90	Reward coefficient
η_D^I	1.10	Reward coefficient
η_I^T	1.20	Reward coefficient
η_I^D	0.80	Reward coefficient

After several simulations, the number of input neurons for invader and defender networks was set to 20. Both agents have 10 neurons in their output layer. The encoding resolution can be enhanced by increasing the number of neurons, although this results in a higher

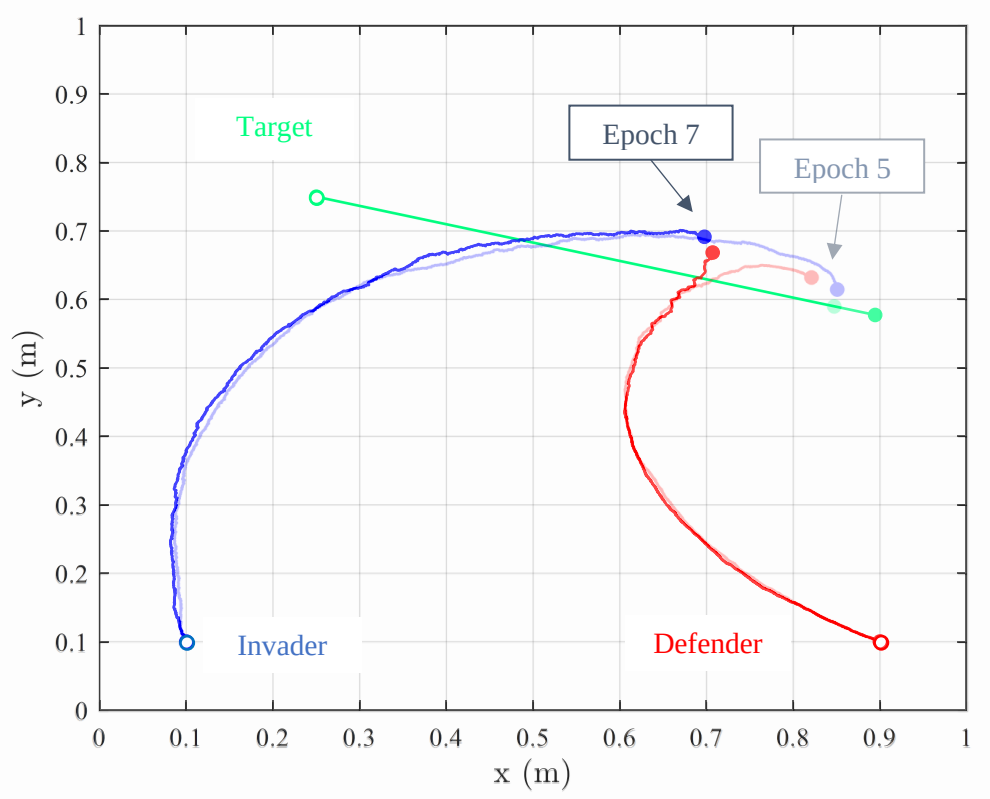

Figure 2.7: SNN's performance during training. Hollow circles show the initial positions.

number of synaptic connections. An optimization algorithm can be employed to determine the optimal neuron number in the SNN. Half of the input neurons for the invader are for the ϕ_I^T, and the other half is for the ϕ_I^D. Half of the defender's input neurons are for the ϕ_D^T, and the other half is for the ϕ_D^I. Since each network contains 20 input neurons, the input layer has 20 Gaussian membership functions.

The output of the activation of each membership function is the input to a neuron associated with that specific membership function. There are 10 input neurons for the 10 membership functions related to each input. Furthermore, no more than 2 membership functions fire for any given input. Therefore, at most, only two neurons are excited and generate an impulse sequence for a given input.

The target's velocity is 0.15 m/s. The invader's velocity is 0.3 m/s. The γ is 0.75, so the defender's velocity is 0.225 m/s. The defender's capture radius (ρ) is set to 0.025 m. The

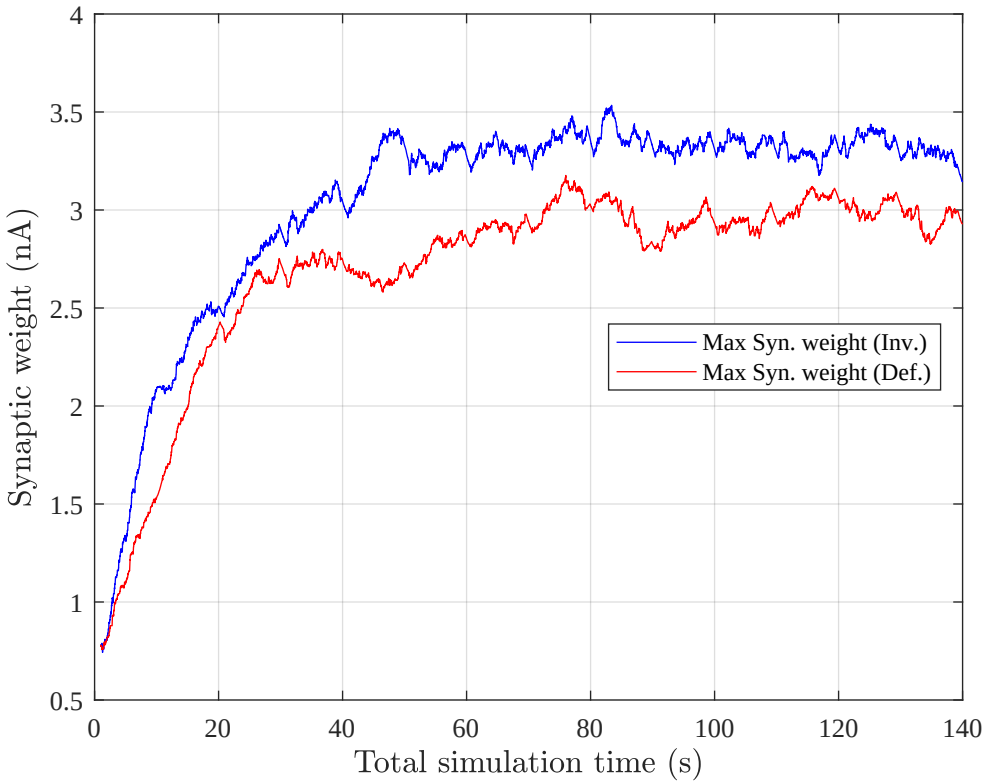

Figure 2.8: Changes in synaptic weights during training. Only Maximum synaptic weights for both agents are shown.

maximum simulation time for each epoch is 10 seconds. The $-\pi/4 < \theta_s < \pi/4$ for invader and defender, and the σ is set to 1.25. The parameter values for R-STDP, shown in Table 2.2, are set through several simulations.

The reward coefficients are set manually. The τ_s in Table 2.2 defines the decaying rate of C in (2.1) that determines the R-STDP sensitivity to prior firings. According to (2.2), higher τ_s means that the R-STDP takes into account the activity of the two neurons that have fired in the relatively larger time window. Different studies have considered different values for this parameter.

As mentioned in Section 3.1, the optimal capture point is the closest point from the reachable region to the target position. Figure 2.7 shows the agents during the training process. In epoch 5, the defender is not able to capture the invader, and the invader reaches the target. In epoch 7, the defender learns how to block and capture the invader. The

defender has won the game. However, the invader should learn to reach the minimum distance from the target.

2.5.1 Simulation without noise

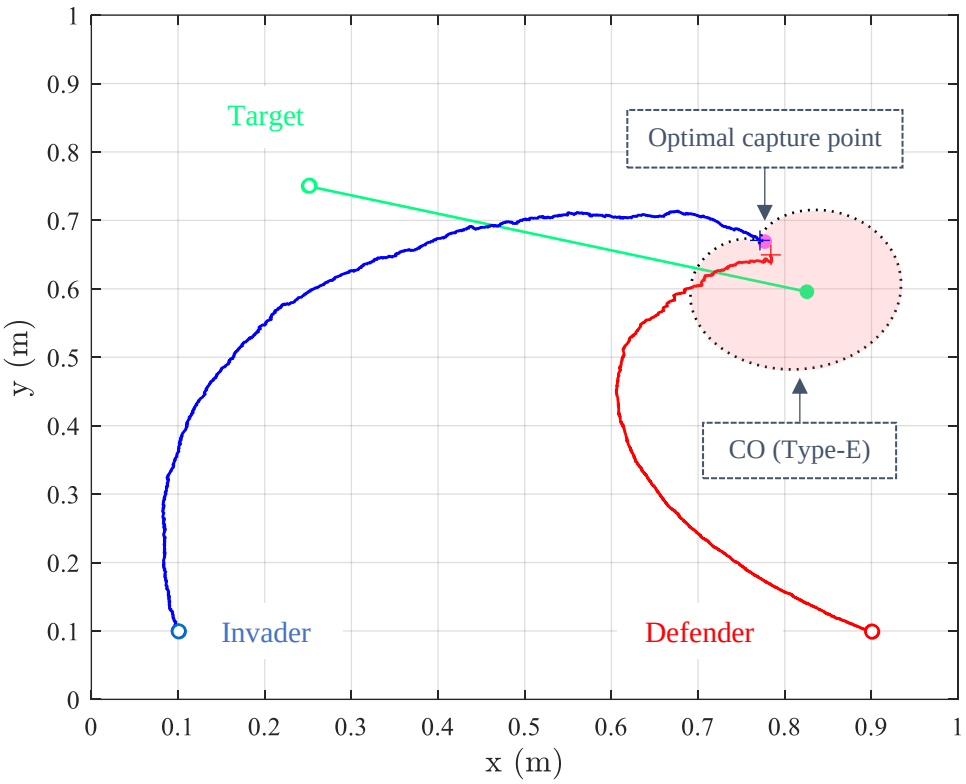

Figure 2.9: SNN's performance after training. The highlighted region shows the defender's dominant region. The purple dot is the optimal capture point. The CO Type-E shows the reachable regions of the Invader and the Defender.

Figure 2.9 shows the performance after training. Each epoch has a maximum time of 10 seconds. After 14 epochs, the defender learned to capture the target, while the invader learned to reduce its distance from the target. According to the CO, the target is inside the defender's dominant region. Therefore, although the invader's velocity is higher than the defender's velocity, it cannot reach the target. In this situation, the optimal policy for the invader is to minimize its distance from the target.

According to Figure 2.9, the invader's SNN can find the optimal [62] capture point for the

invader, while the defender's SNN can protect the moving target against a superior invader. It should be noted that this solution is obtained without a global reference frame. This is important because, in real-world swarm applications, defining a global reference frame is difficult while the learning process is highly dependent on the precise definition of the coordinate system.

Figure 2.8 shows the changes in synaptic weights. Only the synaptic weights with maximum values are shown in this figure because the network has 200 synaptic connections. The minimum value is limited to zero because negative synaptic weights inhibit the post-synaptic neurons. This chapter does not consider the inhibition process. According to figure 2.8, after almost 100 seconds of simulation time, the MSN process causes the synaptic weights to converge.

2.5.2 Simulation with noise

In Figure 2.10, we observe the performance of two methods, namely the SNN and the Cartesian Oval (CO) method, in the presence of noise. The noise in this experiment is introduced as white Gaussian noise, characterized by a mean of zero and a variance of 0.01. Both SNN and CO receive position data that has been corrupted by this noise.

The results of the simulation reveal that the CO method is highly affected by the presence of noise, making it unable to calculate the optimal capture point accurately. Due to its sensitivity to measurement noise, the CO method exhibits a significant deviation from the desired capture point. On the other hand, the SNN method demonstrates a higher level of robustness against noise. Despite the presence of measurement noise, the SNN method manages to achieve the optimal capture point with an error of only 0.036 m. This outcome highlights the superior performance of the SNN method in noisy conditions compared to the CO method.

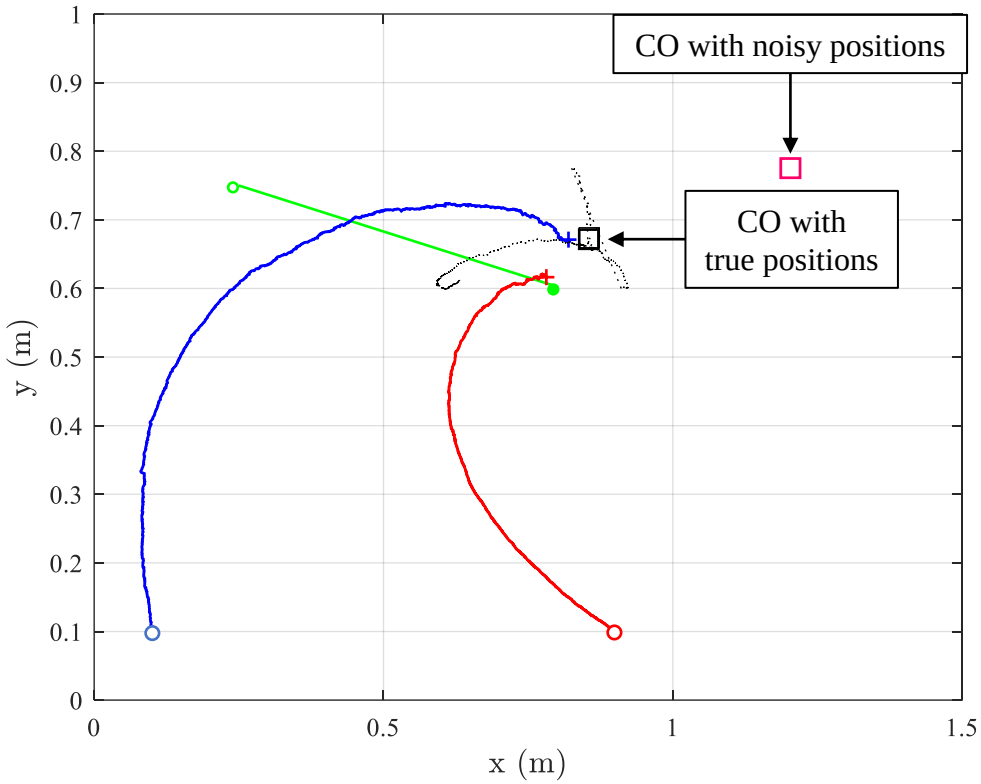

Figure 2.10: SNN's performance in noisy conditions. A white Gaussian noise with a variance of 0.01 is added to the measured inputs.

2.6 Conclusion

This chapter focused on addressing the ATD problem within a dynamic environment involving two agents, where the target is in motion. The approach involved training two SNN simultaneously to engage in a competitive game. During the game, the target transitions from the invader's dominant region to the defender's dominant region. This shift in the target's location within the defender's dominant region satisfied the necessary conditions for determining the reachable regions for both the invader and defender.

To evaluate the effectiveness of the SNN's solution, a comparison was made with Cartesian Oval designed for centralized problems. The results demonstrated that the SNN method was capable of identifying the optimal solution for decentralized problems, even under the presence of noise. This result holds significant practical implications, particularly in scenarios

where establishing a global coordinate system for all agents proves to be challenging. The obtained solution provided by the SNN approach offers a valuable alternative in such cases, showcasing its potential in real-world applications.

Chapter 3

Integration of R-STDP and Federated Learning

3.1 Consensus Flying Problem

This chapter studies the cooperation between follower drones to follow the leader drone by integrating R-STDP and FL. The cooperation problem is formation flying or "Consensus Flying". The consensus flying problem deals with ensuring drones can work together in real-time to agree on their flight paths and positions. When many drones are close together, like in swarms, avoiding crashes is vital. Advanced algorithms and communication methods are needed so drones can exchange information and handle changing situations and unexpected obstacles.

As shown in Figure 3.1, a swarm of agents (follower drones) flies around a leader. The leader is controlled from a remote base station, and the swarm agents should learn to fly safely with the leader. The leader sends its position to all agents, and each agent only sees two neighboring agents. The swarm aims to learn how to keep a commanded distance from each other and the leader. The commanded distance is provided from the leader. Each agent uses the onboard sensors to find the distance and line of sight to neighboring agents.

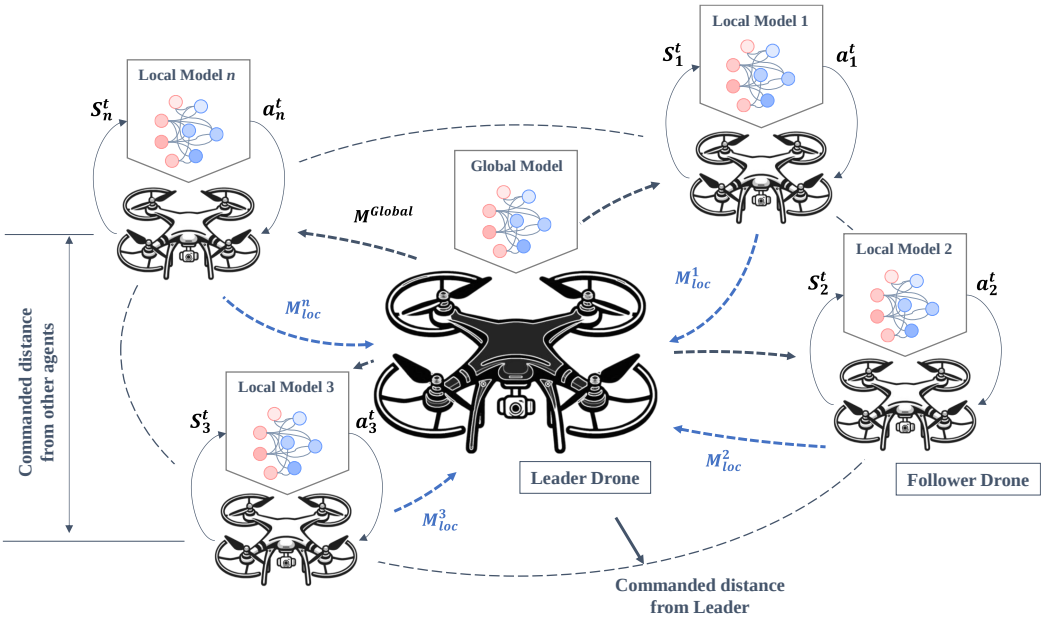

Figure 3.1: The central server (the leader) and the surrounding follower agents (white drones). The follower agents learn to fly in a formation to maintain the commanded distance. The Local models trained individually by follower agents are sent to the leader. The leader aggregates the models and sends back the global model for another round of training of the follower agents.

The follower agents are equipped with an SNN, and their learning algorithm incorporates R-STDP and FL. Each follower agent trains a local network (M^n_{loc}) using R-STDP and sends its model to the leader as the central server. The leader aggregates models and sends back the global model (M^{Global}).

This chapter employs the SNN model to train a group of swarm agents that follow a leader. Each agent has its own SNN, which is trained independently using the R-STDP algorithm. Each agent receives position data from the agents nearby. The goal is for each agent to keep a commanded distance from the leader agent and the other agents in the group. The encoding and decoding processes for the input and output layers of the SNN are fuzzy encoding, and a novel method is introduced to stabilize the network dynamics considering the reward function. This work presents several key contributions:

- The chapter presents a comprehensive method for stabilizing and enhancing the learning process in SNN. This method focuses on controlling the unbounded growth of synaptic weights in SNNs, utilizing a strategy that dynamically adapts to changes in reward conditions and coefficients. It introduces a decay rate and learning rate adjustment based on the status of synaptic weights and enhances the responsiveness of the SNN weights to reward change.

- In terms of advancements in FL with R-STDP, the chapter addresses the FL challenges in the R-STDP framework. It introduces an event-triggered mechanism for model publishing and receiving within the network, improving network traffic. Additionally, the chapter implements a novel weighted aggregation method on the server. This method calculates weights based on the time of arrival of the models, effectively tackling the asynchronous issues in FL.

3.2 Proposed Method

3.2.1 Network Structure

This chapter assumes that each agent detects only two neighboring agents in addition to the leader. The information obtained from other agents includes the Line-of-Sight (LOS) angle and the distance. Each agent's neural network consists of three sub-layers in the input layer, as shown in Figure 3.2. Two sub-layers correspond to the two neighboring follower agents (F_1 and F_2), and the third is dedicated to the leader (L). Inputs for these sub-layers are encoded using the Gaussian Receptive Fields (GRF) that use fuzzy membership functions. The network uses the difference between current and commanded distances within the swarm (r_{cmd}) and between followers and the leader (R_{cmd}) to stimulate input neurons.

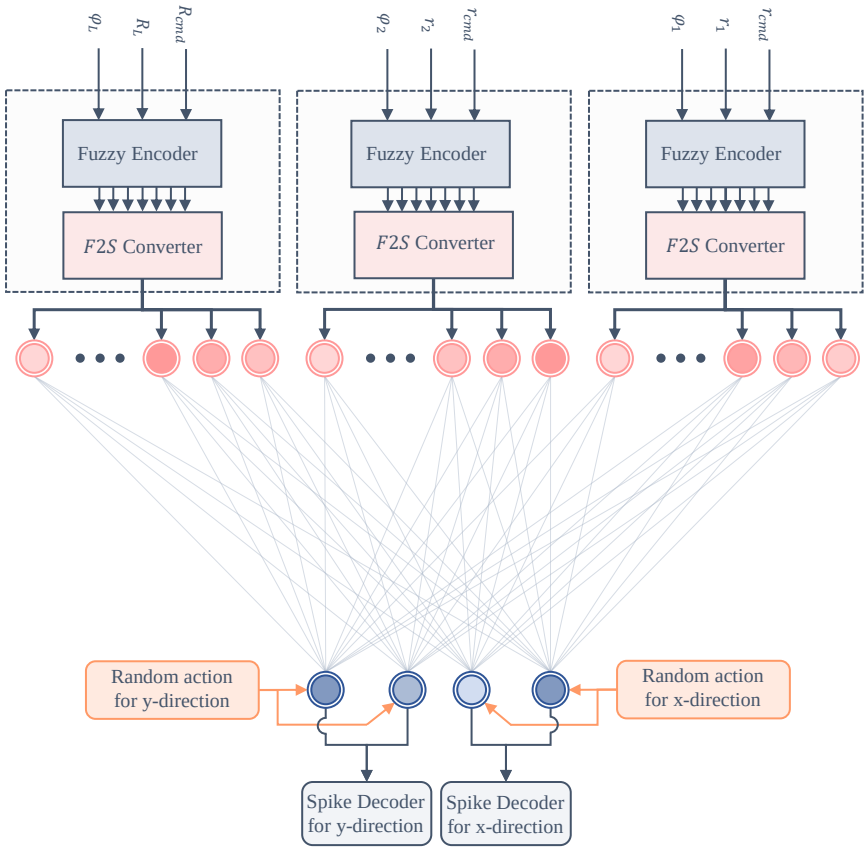

Figure 3.2: SNN structure with encoding and decoding layers. Each sub-layer consists of a fuzzy encoder and the F2S Converter, with the output layer receiving inputs from synaptic weights and a random action selector. During the training phase, the output layer receives input only from the random action selector, which then shifts to synaptic weight inputs after the training.

Every input sub-layer is split into two parts. The first part deals with distances greater than the commanded distance, while the second focuses on the space between the agent and the commanded distance. Within each part, the LOS angle is encoded with fuzzy membership functions. The difference between the current and commanded distance is represented as the error. We transform this difference into an amplitude value using the *tanh* function so that it is bounded between 0 and 1. An error of zero leads to an amplitude of zero, and as the error increases towards infinity, the amplitude approaches one. Consequently, the

encoding function for the input layer is expressed as follows,

$$\mu_I(\phi_i, r_i) = |tanh(r - r_i)| \cdot \exp\left(-\frac{(\phi_i - \zeta)^2}{2\sigma^2}\right) \tag{3.1}$$

where ζ and σ are the Gaussian membership functions' center and standard deviation. The r_i is the distance from the corresponding agent, ϕ_i is the LOS angle, and μ_I is the vector of the membership degrees. Here, r is a placeholder that can either represent r_{cmd} or R_{cmd}, depending on the context. The firing strengths from fuzzy encoders are then converted to the spiking input based on the neuron model as follows [66],

$$I_{sub-layer} = \left(I^{max} - I^{min}\right) \mu_I(\phi_i, r_i) + I^{min} \tag{3.2}$$

or

$$I_{sub-layer} = \frac{\tau_m \left(V_{th} - V_{res}\right)}{\Delta t R_m} \mu_I(\phi_i, r_i) + \frac{V_{th} - E_l}{R_m} \tag{3.3}$$

The encoding process is shown in Figure 3.3. The Fuzzy-to-Spiking (F2S) block uses (3.3) to calculate the inputs for the associated sub-layer.

The output layer has two sub-layers, and each sub-layer has two neurons. The first sub-layer determines the Δx, and the second one determines Δy. The first neuron of the sub-layers is for negative values, and the second one is for positive values. Each neuron is associated with the output sign, and the magnitude of the Δx and Δy is encoded into the output sub-layers based on the minimum and maximum synaptic weights. Equation (3.3) is used to encode the magnitude of the random action into the output sub-layers. The only difference is that a function called μ_O is used to normalize the maximum step between 0 and 1 as follows,

$$\mu_{Ox} = \frac{\Delta x}{\Delta X_{max}} \tag{3.4}$$

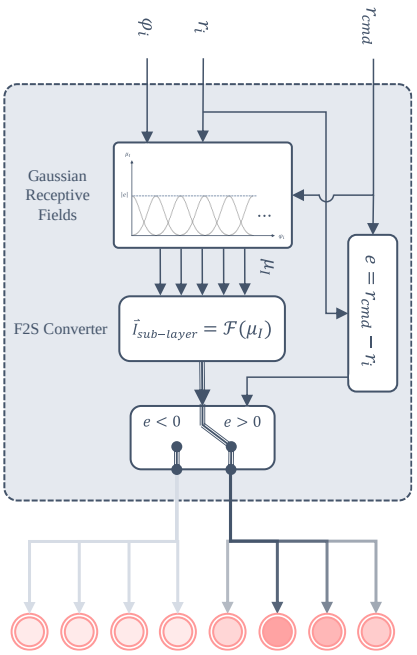

Figure 3.3: The fuzzy encoding principle for the input sub-layer.

$$\mu_{Oy} = \frac{\Delta y}{\Delta Y_{max}} \tag{3.5}$$

where Δx and Δy are selected actions, and ΔX_{max} and ΔY_{max} are maximum steps (displacements) in X and Y directions. Two random actions, one for Δx and one for Δy, are generated for the training process.

The decoding of the spiking output is determined by the difference in the firing rates of the output neurons within each sub-layer. Let us denote $f(t)$ as the activity of the output neurons that control the movement in the x and y-directions:

$$f(i) = \begin{cases} 1 & \text{if the neuron spikes at time } i, \\ 0 & \text{otherwise,} \end{cases}$$

The equation for decoding this activity can be expressed as:

$$\Delta x_{decoded} = \left[\sum_{i=t-\Delta T}^{t} \left(f^{x+}(i) - f^{x-}(i) \right) \right] \Delta X_{max} \tag{3.6}$$

where $f^{x+}(i)$ and $f^{x-}(i)$ are the activities of the two output neurons associated with the x-direction. A similar process is applied for decoding in the y-direction:

$$\Delta y_{decoded} = \left[\sum_{i=t-\Delta T}^{t} \left(f^{y+}(i) - f^{y-}(i) \right) \right] \Delta Y_{max} \tag{3.7}$$

where ΔT is the time window the network updates weights.

The synaptic current entering the output layer is composed of two distinct components:

$$I_{out} = (1 - \alpha)\, I_{syn} + \alpha I_{rnd} \tag{3.8}$$

where α is the learning rate that is set to 1 during the training phase and gradually tends to 0 after training is completed, enabling the system to transition from exploratory to operational modes. I_{rnd} denotes the random steering angle, and I_{syn} represents the steering angle output determined by the synaptic weights.

During the training phase, the current state (S^t) is encoded into the network, and a steering angle (u_{rnd}) is generated randomly via the Feature to Spike (F2S) process into the output layer. This dual encoding process leads to the independent firing of neurons in both the input and output layers. The synaptic weights are adapted based on the activity of these firing neurons through Spike-Timing Dependent Plasticity (STDP). As α is set to 1 during this phase, I_{syn} has no influence on the SNN's output, allowing for purely explorative behavior based on the dynamics of I_{rnd}.

To enhance the smoothness of these transitions and generate the random component I_{rnd}, a recursive random number generation method is utilized:

$$\amalg_t = \gamma \cdot \amalg_{t-1} + (1 - \gamma) \cdot \Upsilon_t \tag{3.9}$$

In this equation, $\amalg_t$ represents the random action at time t, γ is a correlation coefficient

that controls the influence of the previous action, and Υ_t is a random number drawn from a standard distribution (e.g., Gaussian) at time t. This recursive formulation ensures that the action at any given time t is a weighted blend of the previous action and a new random input, functioning as a first-order filter to produce colored noise.

3.2.2 Training algorithm

The R-STDP algorithm without considering the eligibility trace (C) is used for training. The reward $\mathcal{R}(t)$ at time t is defined as,

$$\mathcal{R}_{Fi}^{Fj}(t) = \mathcal{C}_{Fi}^{Fj}\left[r_{Fi}^{Fj}(t-1) - r_{Fi}^{Fj}(t)\right] tanh(r_{Fi}^{Fj}(t) - r_{cmd}) \tag{3.10}$$

$$\mathcal{R}_{Fi}^{L}(t) = \mathcal{C}_{Fi}^{L}\left[r_{Fi}^{L}(t-1) - r_{Fi}^{L}(t)\right] tanh(r_{Fi}^{L}(t) - R_{cmd}) \tag{3.11}$$

where, $\mathcal{R}_{Fi}^{Fj}$, r_{Fi}^{Fj}, and r_{cmd} denote the reward, distance, and commanded distance between two i and j follower agents, respectively. Similarly, $\mathcal{R}_{Fi}^{L}$, r_{Fi}^{L}, and R_{cmd} represent the reward, distance, and the commanded distance between the follower agent i and the Leader (L), respectively. The term $\mathcal{C}_{Fi}^{Fj}$ and $\mathcal{C}_{Fi}^{L}$ are reward coefficients and the $tanh(r_{Fi}^{Fj}(t) - r_{cmd})$ and $tanh(r_{Fi}^{L}(t) - R_{cmd})$ functions determine the reward's sign according to the agents' relative distance and the commanded distance. The expressions $r_{Fi}^{Fj}(t-1) - r_{Fi}^{Fj}(t)$ and $r_{Fi}^{L}(t-1) - r_{Fi}^{L}(t)$ specify the magnitude of the instantaneous reward.

If an agent finds itself farther away from the commanded distance than a neighboring agent or the leader, it will be rewarded positively for decreasing its distance. Conversely, moving closer results in a negative reward if the agent is within the commanded distance from a neighboring agent or the leader. This system is designed to encourage the maintenance of a commanded distance: being too far away from the commanded distance invites a penalty. At the same time, positive reinforcement is given for closing the gap between the current distance and the commanded distance.

One of the challenges in R-STDP is the unbounded growth or decay of synaptic weights,

which can impede stable and effective learning in neural networks. The following section introduces a novel method focused on learning rate and weight stabilization to address this challenge and enhance the algorithm's applicability. This proposed method, designed to regulate synaptic weight changes, ensures a balanced and controlled learning process. It innovatively incorporates an adaptive decay rate technique designed to maintain stability in synaptic weight adjustments, thereby significantly improving the performance and reliability of R-STDP in SNNs.

3.2.3 Weight Stabilization using Reward-Modulated Competitive Synaptic Equilibrium (R-CSE)

Controlling the excessive increase of synaptic weights in SNNs is important to maintain network resilience and function. If not controlled, this growth can lead to saturation, affecting the network's ability to learn and adapt. When the network receives fuzzy sets of firing strengths as input, the synaptic weights grow in a pattern influenced by the Gaussian function's shape used for fuzzy encoding. Imposing a limit on synaptic weights disrupts this growth pattern over time, and eventually, all the synaptic weights reach the maximum. Weight normalization, while preventing excessive growth in one part of the network, can inhibit overall growth; when a synaptic connection reaches its maximum, its activation subsequently diminishes other weights.

Traditional methods like L1 regularization and weight decay employ a constant decay rate, which can slow the network's responsiveness to changes in rewards. Alternatively, a more advanced approach, the Bienenstock, Cooper, and Munro (BCM) method dynamically adjusts both a threshold and a decay rate in response to input variations. However, this method does not provide a control mechanism for the fuzzy inputs. In this chapter, we introduce a method called Reward-Modulated Competitive Synaptic Equilibrium (RCSE) to manage the unbounded growth of synaptic weights while maintaining the gradual change in the synaptic weights formed due to differences in firing strength from fuzzy membership functions. Our method also dynamically adjusts the network when the reward changes by

adjusting the maximum synaptic weight based on the reward.

The enhanced version of the R-STDP method considering the control mechanism from RCSE algorithm is expressed as follows,

$$\dot{\boldsymbol{W}}(t) = \boldsymbol{\alpha} \odot \mathbf{STDP}(\tau) \odot \mathcal{R}(t) - \boldsymbol{\Theta} \odot \operatorname{sgn}(\boldsymbol{W}) \tag{3.12}$$

where $\odot$ is the Hadamard product, $\boldsymbol{\alpha}$ is the learning rate matrix, and $\boldsymbol{\Theta}$ is the decay rate matrix. The primary distinction between the RCSE method and the approach detailed in Section 2.4 lies in the decay rate, which allows the learning system to remain adaptable after the learning phase, and in the learning rate, which is represented as a matrix rather than a scalar value affecting all synaptic weights uniformly. These modifications enhance the learning algorithm's flexibility in responding to reward changes and provide greater control over synaptic weight adjustments.

Let us define $\mathcal{S}$ as the set of input and output neurons that fired at time t in one of the network sections. If we consider $W^{\mathcal{S}}_{max}(t)$ as the maximum weight among the firing neurons in set $\mathcal{S}$, then we can characterize the learning rate using a Sigmoid function. The learning rate value ($\boldsymbol{\alpha}^{\mathcal{S}}(t)$) gradually transitions from 1 to 0 as the learning process advances, as explained below:

$$\boldsymbol{\alpha}^{\mathcal{S}}(t) = \frac{1}{1 + \exp\left[\frac{1}{\epsilon}\left(|W^{\mathcal{S}}_{max}(t)| - \Psi^{\mathcal{S}}\right)\right]}, \quad \left(\Psi^{\mathcal{S}} = \frac{\mathcal{R}^{\mathcal{S}}_{max}}{\mathcal{R}^{G}_{max}} I^{max}\right) \tag{3.13}$$

where $\mathcal{R}^{\mathcal{S}}_{max}$ is the maximum reward in the network section (e.g., $max(\mathcal{R}^{Fj}_{Fi})$), $\mathcal{R}^{G}_{max} = max(\mathcal{R}^{Fj}_{Fi}, \mathcal{R}^{L}_{Fi})$, and ϵ is a small positive number that controls the curvature of the function around $W^{\mathcal{S}}_{max}(t) = \Psi^{\mathcal{S}}$. This model determines the learning rate by the highest synaptic weight among the active input and output neurons. This mechanism is similar to the "winner-takes-all" approach. When a synaptic connection reaches its weight limit, it prevents further changes in the adjacent synaptic weights.

The network contains a variety of reward functions, each with its own maximum and minimum values. The highest reward value in a specific area of the network sets the limit for

the synaptic weight in that area. The synaptic weight limit is linked to the ratio of the local maximum reward ($\mathcal{R}^{\mathcal{S}}_{max}$) to the global maximum reward ($\mathcal{R}^{G}_{max}$). As a result, the network section with the highest local maximum reward ($\mathcal{R}^{\mathcal{S}}_{max} = \mathcal{R}^{G}_{max}$) attains the maximum allowable synaptic weights because $\frac{\mathcal{R}^{\mathcal{S}}_{max}}{\mathcal{R}^{G}_{max}} = 1$, while sections with lower local maximum rewards reach only a proportional fraction of the maximum weight. The adjustment of the learning rate transforms into a competitive algorithm that modifies the growth rate of individual synaptic weights by considering network parameters, like reward and maximum synaptic weight.

A significant challenge in learning algorithms is their capacity to adapt to changes in rewards. Commonly, once the learning rate reduces to zero, weight adjustments stop. To address this, a variable decay rate is introduced to prevent weights in each network section from indefinitely remaining at their peak values. In our method, the decay rate is represented as a matrix, and it is calculated using the SoftPlus function, enabling it to adjust according to the current stage of learning. This method ensures that weight modifications continue to respond effectively to changes in the learning environment.

This chapter defines the decay rate as a function of the maximum synaptic weight among neurons in the set $\mathcal{S}$. This approach is designed to address a critical aspect: when the maximum synaptic weight in $\mathcal{S}$ reaches its peak ($|W^{\mathcal{S}}_{max}(t)| = \Psi^{\mathcal{S}}$), it is essential that the learning rate remains above zero. This condition is necessary to allow weight change and prevent the learning rate from stagnating at zero ((3.12)). Simultaneously, the learning rate must not exceed the maximum acceptable rate of weight change, which is $\mathcal{A} \times \mathcal{R}^{G}_{max}$. When the reward coefficients change after training, it can cause $|W^{\mathcal{S}}_{max}(t)|$ to exceed $\Psi^{\mathcal{S}}$ for set $\mathcal{S}$. With these considerations, we propose that the decay rate should be set to $\mathcal{A}/\lambda \times \mathcal{R}^{G}_{max}$ when $|W^{\mathcal{S}}_{max}(t)| = \Psi^{\mathcal{S}}$ and increase to $\lambda\mathcal{A} \times \mathcal{R}^{G}_{max}$ when $|W^{\mathcal{S}}_{max}(t)| = 2\Psi^{\mathcal{S}}$, where λ is a coefficient that controls the rate of decay when $|W^{\mathcal{S}}_{max}(t)| > \Psi^{\mathcal{S}}$.

By applying the mentioned condition and solving for the SoftPlus function, the decay rate function can be obtained as follows,

$$\Theta^{\mathcal{S}} = \left(\frac{\eta}{\beta}\right) \log\left(1 + \exp\left[\beta\left(|W^{\mathcal{S}}_{max}(t)| - \Psi^{\mathcal{S}}\right)\right]\right) \tag{3.14}$$

where $\eta = \frac{\mathcal{A}\mathcal{R}^{G}_{max} \ln\left(2^{\lambda^2}-1\right)}{\lambda \Psi^{\mathcal{S}} \log(2)}$ is a scaling parameter that can adjust the output scale of the function, and $\beta = \frac{\ln\left(2^{\lambda^2}-1\right)}{\Psi^{\mathcal{S}}}$ controls the curvature of the function. A higher β makes the SoftPlus function approach a step function, making it closer to the binary behavior. Conversely, a smaller β makes the function smoother and more gradual. Equation (3.14) can be represented as,

$$\Theta^{\mathcal{S}} = \left(\frac{\mathcal{A}\mathcal{R}^{G}_{max}}{\lambda \log(2)}\right) \log\left(1 + \exp\left[\left(\frac{\ln\left(2^{\lambda^2}-1\right)}{\Psi^{\mathcal{S}}}\right)\left(|W^{\mathcal{S}}_{max}(t)| - \Psi^{\mathcal{S}}\right)\right]\right) \tag{3.15}$$

The choice of setting the decay rate to $\lambda\mathcal{A} \times \mathcal{R}^{G}_{max}$ when $|W^{\mathcal{S}}_{max}(t)| = 2\Psi^{\mathcal{S}}$ is based on the feature of reward coefficients. Specifically, when the reward coefficients in (3.10) and (3.11) increase, leading to new condition where $\mathcal{R}^{\mathcal{S}}_{max}$ or $\mathcal{R}^{G}_{max}$ change, the $|W^{\mathcal{S}}_{max}(t)|$ is allowed to increase. Conversely, a decrease in the reward coefficient, resulting in $|W^{\mathcal{S}}_{max}(t)| > \Psi^{\mathcal{S}}$, necessitates a higher decay rate to reduce the $|W^{\mathcal{S}}_{max}(t)|$ back to $\Psi^{\mathcal{S}}$.

When $|W^{\mathcal{S}}_{max}(t)| < \Psi^{\mathcal{S}}$, the reward adjusts the synaptic weights, and there is no weight decay to disturb the learning process. When $|W^{\mathcal{S}}_{max}(t)| > \Psi^{\mathcal{S}}$, the decay rate changes the synaptic weights and brings the maximum weight to the reward zone, where $|W^{\mathcal{S}}_{max}(t)| < \Psi^{\mathcal{S}}$ and the networks responds to reward change.

Numerical example for the RCSE algorithm

Consider a set of input neurons firing based on their fuzzy membership degrees. For an input received, a set of adjacent input neurons fires, along with an output neuron. Let the set $\mathcal{S}$ be defined based on the fired neurons as follows:

$$\mathcal{S} = \{15, 16, 17, 18, 73\}$$

In this set, neurons 15, 16, 17, and 18 are fired input neurons, and neuron 73 is the fired output neuron. First, the algorithm finds the maximum synaptic weight connecting these input neurons to the output neuron, denoted as $W^{\mathcal{S}}_{max}(t)$. Let's assume that $I^{max} = 15.5$

and ϵ is a small positive number (e.g., 0.0001).

The decay rate is formulated using a SoftPlus function, which is parameterized by λ and $\mathcal{A}$, where $\lambda = 5$ and $\mathcal{A} = 1$ in this numerical example. The η and β influence the decay rate adjustments under various synaptic conditions and reward structures.

Case 1: $W^{\mathcal{S}}_{max}(t) < \Psi^{\mathcal{S}}$

Let's assume that $\mathcal{R}^{\mathcal{S}}_{max} = 0.5$ and $\mathcal{R}^{G}_{max} = 1$ ($\Psi^{\mathcal{S}} = 0.5/1 \times 15.5 = 7.75$), so the maximum change rate for this set $\mathcal{S}$ is $\mathcal{A}\mathcal{R}^{\mathcal{S}}_{max} = 0.5$ while the maximum change rate in the network is $\mathcal{A}\mathcal{R}^{G}_{max} = 1$. For $W^{\mathcal{S}}_{max}(t) = 7$, the decay rate $\Theta^{\mathcal{S}}$ can be calculated as follows,

$$\beta = \frac{\ln\left(2^{\lambda^2} - 1\right)}{\Psi^{\mathcal{S}}} = \frac{\ln(2^{5^2} - 1)}{7.75} = 2.236$$

$$\eta = \frac{\mathcal{A}\mathcal{R}^{G}_{max} \ln\left(2^{\lambda^2} - 1\right)}{\lambda \Psi^{\mathcal{S}} \log\left(2\right)} = \frac{1 \cdot 1 \cdot \ln(2^{5^2} - 1)}{5 \cdot 7.75 \cdot \log(2)} = 1.485$$

$$\Theta^{\mathcal{S}} = \left(\frac{1.485}{2.236}\right) \log\left(1 + \exp\left[2.236 \cdot (7 - 7.75)\right]\right) = 0.05$$

$$\boldsymbol{\alpha}^{\mathcal{S}}(t) = \frac{1}{1 + \exp\left[\frac{1}{0.0001}\left(7 - 7.75\right)\right]} = 1$$

According to (3.12), the decay rate is low enough in comparison with the learning rate to allow for synaptic growth ($\mathcal{A}\mathcal{R}^{\mathcal{S}}_{max}$), promoting an increase in synaptic strength.

Case 2: $W^{\mathcal{S}}_{max}(t) > \Psi^{\mathcal{S}}$

Let's assume that $W^{\mathcal{S}}_{max}(t) = 10$, the decay and learning rates can be calculated as follows,

$$\Theta^{\mathcal{S}} = \left(\frac{1.485}{2.236}\right) \log\left(1 + \exp\left[2.236 \cdot (10 - 7.75)\right]\right) = 1.453$$

$$\boldsymbol{\alpha}^{\mathcal{S}}(t) = \frac{1}{1 + \exp\left[\frac{1}{0.0001}\left(10 - 7.75\right)\right]} \approx 0$$

This results in a higher decay rate for the set $\mathcal{S}$ (while $\alpha^{\mathcal{S}} = 0$), actively working to reduce synaptic strength towards the threshold $\Psi^{\mathcal{S}}$, due to $W^{\mathcal{S}}_{max}(t)$ exceeding $\Psi^{\mathcal{S}}$.

Case 3: When $\Psi^{\mathcal{S}}$ increases from 7.75 to 15.5 (Reward Change)

Assuming a reward change causes $\Psi^{\mathcal{S}}$ to increase to 15.5 ($\mathcal{R}^{\mathcal{S}}_{max} = \mathcal{R}^{G}_{max} = 1$) and considering $W^{\mathcal{S}}_{max}(t) = 7.8$, the decay rate $\Theta^{\mathcal{S}}$ is recalculated as follows,

$$\beta = \frac{\ln\left(2^{\lambda^2} - 1\right)}{\Psi^{\mathcal{S}}} = \frac{\ln(2^{5^2} - 1)}{15.5} = 1.118$$

$$\eta = \frac{\mathcal{A}\mathcal{R}^{G}_{max} \ln\left(2^{\lambda^2} - 1\right)}{\lambda \Psi^{\mathcal{S}} \log\left(2\right)} = \frac{1 \cdot 1 \cdot \ln(2^{5^2} - 1)}{5 \cdot 15.5 \cdot \log(2)} = 0.743$$

$$\Theta^{\mathcal{S}} = \left(\frac{0.743}{1.118}\right) \log\left(1 + \exp\left[1.118 \cdot (7.8 - 15.5)\right]\right) \approx 0$$

$$\boldsymbol{\alpha}^{\mathcal{S}}(t) = \frac{1}{1 + \exp\left[\frac{1}{0.01}\left(7.8 - 15.5\right)\right]} = 1$$

The decay rate allows for synaptic growth of set $\mathcal{S}$ as the maximum weight is below the new threshold.

Case 4: When $\Psi^{\mathcal{S}}$ decreases from 15.5 to 7.75 (Reward Change)

If $\Psi^{\mathcal{S}}$ decreases to 7.75 due to reward changes and assuming $W^{\mathcal{S}}_{max}(t) = 15.6$, the decay rate $\Theta^{\mathcal{S}}$ increases as follows,

$$\beta = \frac{\ln\left(2^{\lambda^2} - 1\right)}{\Psi^{\mathcal{S}}} = \frac{\ln(2^{5^2} - 1)}{7.75} = 2.236$$

$$\eta = \frac{\mathcal{A}\mathcal{R}^{G}_{max} \ln\left(2^{\lambda^2} - 1\right)}{\lambda \Psi^{\mathcal{S}} \log\left(2\right)} = \frac{1 \cdot 1 \cdot \ln(2^{5^2} - 1)}{5 \cdot 7.75 \cdot \log(2)} = 1.485$$

$$\Theta^{\mathcal{S}} = \left(\frac{1.485}{2.236}\right) \log\left(1 + \exp\left[2.236 \cdot (15.6 - 7.75)\right]\right) = 5.063$$

$$\boldsymbol{\alpha}^{\mathcal{S}}(t) = \frac{1}{1 + \exp\left[\frac{1}{0.01}\left(15.6 - 7.75\right)\right]} = 0$$

In this scenario, the decay rate is very high, and the learning rate is 0. Therefore, the

algorithm aggressively pulls the synaptic weight back toward the lowered threshold.

Lemma 3.2.1. *The RCSE method is asymptotically stable in its equilibrium point* $W^{\mathcal{S}}_{max}(t) = \Psi^{\mathcal{S}}$.

Proof. We consider the dynamical system given by the equation of synaptic weights for $W^{\mathcal{S}}_{max}(t) \geq 0$ that receives a positive reward ($\mathcal{R}(t) > 0$) as,

$$\begin{aligned}\dot{W}^{\mathcal{S}}_{max}(t) =& \frac{1}{1+\exp\left[\frac{1}{\epsilon}\left(W^{\mathcal{S}}_{max}(t)-\Psi^{\mathcal{S}}\right)\right]} STDP(\tau)\mathcal{R}(t) \\ &- \left(\frac{\mathcal{A}\mathcal{R}^{G}_{max}}{\lambda \log(2)}\right) \log\left(1+\exp\left[\left(\frac{\ln\left(2^{\lambda^2}-1\right)}{\Psi^{\mathcal{S}}}\right)\left(W^{\mathcal{S}}_{max}(t)-\Psi^{\mathcal{S}}\right)\right]\right),\end{aligned} \tag{3.16}$$

To assess the stability of this system around the equilibrium point, we introduce a Lyapunov function candidate $V(z)$, where $z = W^{\mathcal{S}}_{max}(t) - \Psi^{\mathcal{S}}$. A common choice for such analyses is a quadratic function of the deviation from the equilibrium:

$$V(z) = \frac{1}{2}z^2. \tag{3.17}$$

This positive definite function has a minimum at the equilibrium point, satisfying the essential criteria for a Lyapunov function. The derivative of $V(z)$ with respect to time, $\dot{V}(z)$, is then calculated to determine the rate of change of the Lyapunov function along the trajectories of the system:

$$\dot{V}(z) = z\dot{z} \tag{3.18}$$

Substituting $\dot{W}^{\mathcal{S}}_{max}(t)$ from (3.16) into the above expression, we have,

$$\dot{V}(z) = z \times \left[\frac{1}{1+\exp\left[\frac{1}{\epsilon}(z)\right]} STDP(\tau)\mathcal{R}(t) - \left(\frac{\mathcal{A}\mathcal{R}^{G}_{max}}{\lambda \log(2)}\right) \log\left(1+\exp\left[\left(\frac{\ln\left(2^{\lambda^2}-1\right)}{\Psi^{\mathcal{S}}}\right)(z)\right]\right)\right] \tag{3.19}$$

A negative $\dot{V}(z)$ indicates that the system's energy decreases over time, concluding that the equilibrium point is asymptotically stable. Conversely, a positive $\dot{V}(z)$ in any region would suggest the presence of instability or regions of attraction that do not encompass the entire state space.

In analyzing the system's stability, we focus on the behavior of the derivative of the Lyapunov function, $\dot{V}(z)$, across different regions of z. We decompose the dynamics of $\dot{z}$ into its constituent components to systematically analyze the stability conditions. We assess the relative magnitudes of the two main components influencing $\dot{V}(z)$:

1. The first term, represented as $\frac{1}{1+\exp\left[\frac{1}{\epsilon}z\right]}STDP(\tau)\mathcal{R}(t)$, denotes the effect of learning rate and is inherently positive when the synaptic connection receives positive reward consistently.

2. The second term, $\left(\frac{\mathcal{A}\mathcal{R}^G_{max}}{\lambda \log(2)}\right) \log\left(1+\exp\left[\left(\frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}}\right)z\right]\right)$, captures the dynamic decay rate of synaptic weights, which is governed by the SoftPlus function.

Analysis for $z < 0$:

In this region, the term $\exp\left[\left(\frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}}\right)z\right]$ tends to zero for $z \to -\infty$ and makes the term inside the log function go to 1. This implies that the contribution of this term to $\dot{V}(z)$ is negligible in this region. Moreover, the first term remains positive throughout, and given that it is multiplied by z (which is negative in this region), the overall contribution to $\dot{V}(z)$ is negative. Consequently, $\dot{V}(z)$ is negative for $z < 0$, indicating that any perturbations from the equilibrium in this region will decrease over time, thereby contributing to the system's stability.

Analysis for $z > 0$:

For this region, the magnitude of the second term significantly exceeds that of the first term. This predominance is critical as it is associated with a negative sign in the $\dot{V}(z)$ equation. Therefore, the negative contribution of this component ensures that $\dot{V}(z)$ remains negative throughout this region. It indicates that any deviation from the equilibrium state results in the system's energy decreasing over time, leading to the conclusion that the equilibrium point $W^{\mathcal{S}}_{max}(t) = \Psi^{\mathcal{S}}$ is asymptotically stable for $z > 0$.

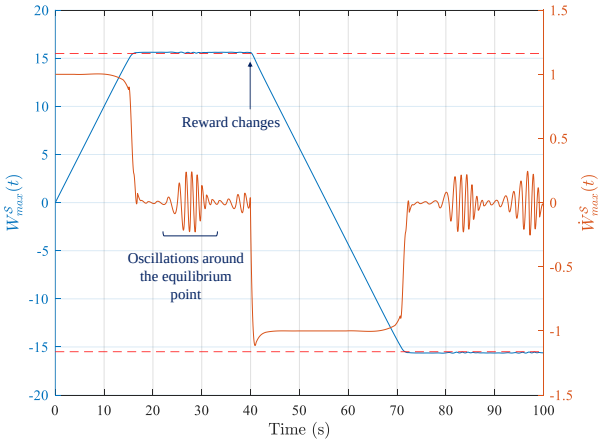

Figure 3.4: Synaptic weight change for $\lambda = 5$, $\Psi^{\mathcal{S}} = 15.5$, and $\mathcal{AR}^{G} = 1$.

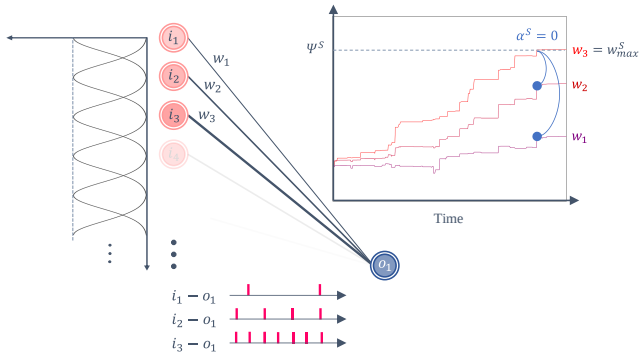

Figure 3.5: RCSE working principle in inhibiting the adjacent synaptic connections.

Figure 3.4 demonstrates the performance of the RCSE when it regulates the synaptic weights to prevent unbounded growth. The red dotted line represents the value of $\Psi^{\mathcal{S}}$, which is derived from the maximum reward value of the corresponding network section. As shown in Figure 3.4, the synaptic weight oscillates around $\Psi^{\mathcal{S}}$, and when it drops below $\Psi^{\mathcal{S}}$, the learning rate is set to 1 by (3.13). This allows any changes in the reward function to be applied to the synapse. Figure 3.5 shows how the maximum synaptic weight of the active synapses in set $\mathcal{S}$ stops the adjacent synaptic connections' growth by setting the learning rate of the set to 0.

3.2.4 Federated Learning for Consensus Flying

In FL, a key challenge is centralizing various models on one server. This process must effectively combine these models to create a unified global model without compromising the specific adjustments made to each model. A critical strategy involves choosing models that contain substantial information. Another significant aspect is determining the frequency of model aggregation. Shorter intervals between aggregations can enhance learning efficiency

but may strain network resources, particularly as the number of participating agents and devices grows. Conversely, longer intervals might slow down the learning process due to delayed updates of the global model. This section proposes an aggregation method for SNN. Our focus is on reducing network usage and energy consumption.

This approach allows clients to upload their local model updates at different times rather than synchronously. Such a method is particularly beneficial in reducing the negative impacts of device heterogeneity, which can include varying computational capacities and network connectivity among devices [67]. In traditional FL setups, delays caused by poor network signals or unexpected client crashes can significantly prolong the time the server takes to receive updates from all clients. By adopting asynchronous aggregation, the server processes and aggregates models as they are received without synchronizing with all clients. This strategy accelerates the training process, making FL more efficient and adaptable to diverse client conditions.

Our proposed FL model aggregation algorithm aims to establish an efficient and event-triggered system for global and local model publishing. This system relies on the similarity between consecutive global and local models and publishes updates only when significant changes are detected, thus avoiding redundant updates and improving overall efficiency. Unlike the uniform model updates in FedAvg [68, 69], our approach allows individual agents to evaluate and send their local models based on a similarity threshold with the global model, thereby enabling a potentially more effective update process. Our aggregation strategy emphasizes similarity metrics for model updates, which is not commonly emphasized in methods like FedNova [70], adding a layer of context sensitivity to our approach.

In our approach, considering the difference in agents' neural network parameters and maximum and minimum synaptic weights, the weights are normalized to align them on a uniform scale ranging from -1 to 1. This normalization process makes the neural model values comparable across the network. Based on the maximum and minimum synaptic weights outlined in Equations (1.5) and (1.9), and taking into account the highest excitation (I^{max}) and inhibition ($-I^{max}$), the normalization of synaptic weights is performed as follows:

$$\overline{\boldsymbol{W}}_k(t) = \frac{1}{I_k^{max}}[\boldsymbol{W}_k(t)] \tag{3.20}$$

where $\boldsymbol{W}_k(t)$ represents the matrix of synaptic weights, $\overline{\boldsymbol{W}}_k(t)$ denotes the normalized synaptic weight matrix for agent $k \in \{1, 2, 3, ..\}$, and I_k^{max} is the maximum synaptic weight for agent k.

The global model on the server (Leader) is then computed using a weighted average,

$$\overline{\boldsymbol{W}}_G(t) = \frac{\sum_{k=1}^{N} \omega_k . \overline{\boldsymbol{W}}_k(t)}{\sum_{k=1}^{N} \omega_k} \tag{3.21}$$

where $\overline{\boldsymbol{W}}_G(t)$ is the global normalized model on the central server, N is the number of agents, and ω_k is the aggregation weight for each SNN model, defined as,

$$\omega_k = \frac{1}{\sqrt{mn}} \|\overline{\boldsymbol{W}}_k(t)\|_F \exp\left(-\frac{t - T_k}{\tau_{cs}}\right) \quad (t \geq T_k) \tag{3.22}$$

where the term $\|.\|_F$ is the Frobenius norm, and m and n are the dimensions of the matrix $\overline{\boldsymbol{W}}_k(t)$, used for normalizing the Frobenius norm. T_k indicates the time at which agent k last transmitted its local model to the central server, and τ_{cs} is a time constant that reduces the weight to zero if there is no recent update from the agent.

Both agents and the central server employ an event-triggered mechanism for transmitting local and global models. Throughout the training phase, each agent calculates the Euclidean distance between the most recent global model from the central server and its current synaptic weights matrix, as follows,

$$\mathcal{D}_a(\overline{\boldsymbol{W}}_k(t), \overline{\boldsymbol{W}}_G(T_{cs})) = \frac{1}{2\sqrt{mn}} \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} (a_{ij} - b_{ij})^2} \tag{3.23}$$

where $\mathcal{D}_a$ is the Euclidean distance on the agent side, T_{cs} is the time when the central server published the global model, and a_{ij} and b_{ij} are elements of the latest global model and the current local model, respectively. If this distance exceeds a certain threshold, set between 0 and 1, the agent transmits its model to the central server.

If the $\mathcal{D}_a$ on the agent k reaches the threshold and it does not receive any update from the server, the agent sends its model to the server, and then it calculates the $\mathcal{D}_a$ between current synaptic weights $\overline{\boldsymbol{W}}_k(t)$ and the last model that it sent to the server at time T_k ($\overline{\boldsymbol{W}}_k(T_k)$) until it receives a new model update from the central server.

The central server follows a similar procedure as the agents, evaluating the distance $\mathcal{D}_G$ between the current and recently published model at time T_{cs},

$$\mathcal{D}_G(\overline{\boldsymbol{W}}_G(t), \overline{\boldsymbol{W}}_G(T_{cs})) = \frac{1}{2\sqrt{mn}} \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ij} - b_{ij})^2} \quad (t \geq T_{cs}) \tag{3.24}$$

Incorporating the proposed FL method with the RCSE algorithm, the modified R-STDP equation can be represented as follows,

$$\begin{aligned}\dot{\boldsymbol{W}}_k(t) =& (1 - \delta(t - T_{cs}))\left[\boldsymbol{\alpha} \odot \mathbf{STDP}(\tau) \odot \mathcal{R}(t) - \boldsymbol{\Theta} \odot \operatorname{sgn}(\boldsymbol{W}_k(t))\right] \\ &+ \delta(t - T_{cs}) I_k^{max} \left(\overline{\boldsymbol{W}}_G(t) - \overline{\boldsymbol{W}}_k(t)\right)\end{aligned} \tag{3.25}$$

where δ is the Dirac delta function. Algorithm 2 shows the step-by-step implementation process of the proposed method. The $\overline{\boldsymbol{W}}_G(t) - \overline{\boldsymbol{W}}_k(t)$ part is the difference between the agent's current model and the global model at aggregation time. Therefore, the change in synaptic weights $\dot{\boldsymbol{W}}_k(t)$ is the difference between models when $t = T_{cs}$.

Algorithm 2 High-Level Algorithm for the Proposed FL Algorithm

Require: Initialization of Central Server and Agents

Ensure: Updated Global Model on the Central Server and Local Models on Agents

Initialize the agents and Central Server with default parameters for model publication threshold, Euclidean distance, and model publish status
Initialize the Global Model on the Central Server
if t is greater than 0 **then**
    Normalize synaptic weights of local models using (3.20)
    Aggregate models from all agents at the Central Server using (3.21)
    Calculate the $\mathcal{D}_G$ between the current and previous global models on the Central Server using (3.24)
    if $\mathcal{D}_G >$ the Central Server's threshold **then**
        Publish the global model
        set $T_{cs} = t$
    end if
    for each Agent in the network **do**
        if Central Server publishes a new global model **then**
            Update the local model of the Agent with the global model using (3.25)
        else
            Agents evaluate their local models against the latest global model ($\mathcal{D}_a$) using (3.23)
            if $\mathcal{D}_a >$ the Agent's threshold **then**
                Send the model to the Central Server
                set $T_k = t$
            end if
        end if
    end for
end if

The proposed algorithm allows agents to communicate less often and save energy. It only sends essential updates to the Central Server, which helps when many agents have different SNN models and communication interfaces. This method reduces unnecessary data transmission, making the whole system more efficient.

3.3 Results and Discussion

In this section, we conducted a numerical simulation to validate the performance of the proposed method. The simulation involves a group of five agents flying around a leader who is moving in a circular path. Initially, a scenario without implementing FL was conducted to evaluate the performance of the SNN in achieving coordinated flight. During this phase, the effect of the change in reward was simulated to examine the RCSE method. In the second part of the simulation, the proposed FL aggregation algorithm is used, where the leader agent acts as a central server. Finally, the algorithm was tested both before and after changing the rewards.

3.3.1 Simulation without FL

In this simulation, we modeled five agents, each equipped with its own SNN model, capable of reaching a maximum speed of 1 m/s. The architecture of each agent's neural network included 72 input neurons. The number of input neurons was derived from a series of numerical simulations. Since each agent was designed to detect three distinct objects within its environment, the input layer was organized into sub-layers, with 24 neurons dedicated to each object (24 membership functions and neurons for each object). The network's output layer comprised 4 neurons, divided equally to represent negative and positive Δx and Δy movements. The SNN model in the simulation is a fully connected network, and the parameters of the LIF neuron are also presented in Table 3.1.

Table 3.1: Parameter values for LIF neuron model [10]

Parameter	**Value**	**Description**
R_m	40 MΩ	Membrane Resistance
τ_m	30 ms	Membrane time constant
E_l	-70 mV	Resting potential
V_{res}	-70 mV	Reset potential
V_0	-70 mV	Initial membrane potential
V_{th}	-50 mV	Threshold membrane potential

The R-STDP mechanism updated synaptic weights at 10 ms intervals. During these intervals, the learning algorithm adjusted the agent's states based on received data from other agents and the leader while simultaneously generating random outputs as part of an exploration strategy (section 3.2.3).

Table 3.2: Simulation Parameters

Parameter	**Value**	**Description**
ΔT	10 ms	Weight and state update sample time
τ_s	2 ms	Time constant for R-STDP
$\mathcal{A}$	1	Amplitude in R-STDP function
λ	5	Decay rate coefficient
Δx and Δy	0.01 m	Max step per ΔT
σ	0.5	Gaussian function's std. deviation
Δt	1 ms	Minimum inter-spike interval
I^{min}	0.5	Lower bound of synaptic weight
I^{max}	15.5	Upper bound of synaptic weight
γ	0.95	Correlation Coefficient

Table 3.2 shows the simulation parameters. The simulation was done in a 10 m by 10 m area, and the leader followed a circular path centered at (5,5) with a 2.5 m radius and a 0.1 m/s speed.

In order to monitor the swarm performance, the minimum and maximum distances of each agent from other agents and the minimum and maximum distances of the swarm from the leader were measured. Figure 3.6 shows the definition of the distances.

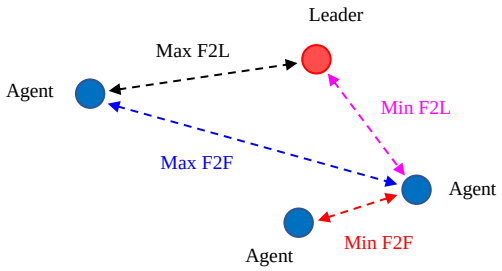

Figure 3.6: Measured distances used for evaluating swarm flight performance and collision detection.

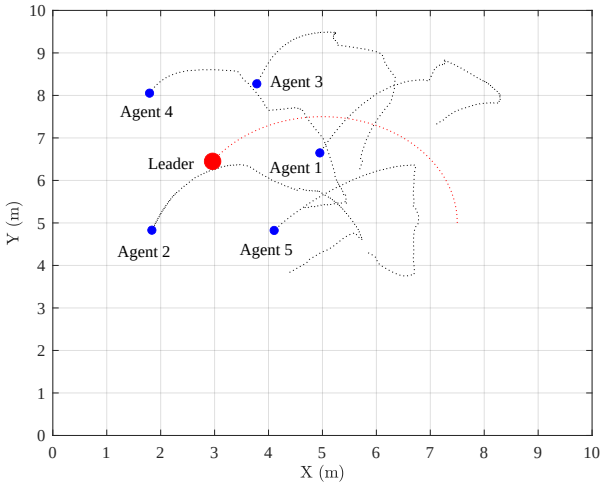

Figure 3.7: Agents' trajectory during the test phase

The simulation included two phases. During the initial phase, the objective was for the agents to learn to maintain the commanded distance from each other and the leader. This phase took 600 seconds for training, and the reward coefficient among followers ($\mathcal{C}_{Fi}^{Fj}$) was set at 0.02, while the coefficient between followers and the leader ($\mathcal{C}_{Fi}^{L}$) was set at 0.07. These parameters were derived from a series of numerical simulations. A higher value of $\mathcal{C}_{Fi}^{L}$ signifies an increased emphasis on the leader in the learning process, which means that the distance to the leader is more important than the commanded distance between agents.

Figure 3.8 shows the simulation results for the RCSE method. According to the results, the agents rapidly aligned around the leader within 6.89 seconds, and the maximum distance was reduced from 7.632 meters to the target distance of 2 meters. The swarm completed the formation around the leader in approximately 8.94 seconds, avoiding collisions.

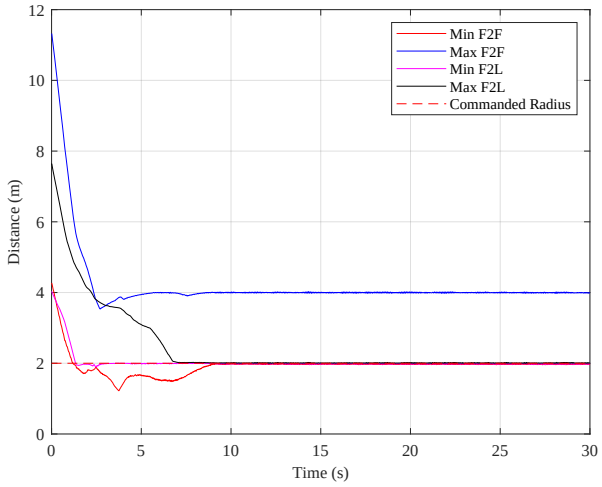

Figure 3.8: Variation of distances within the swarm during the test phase

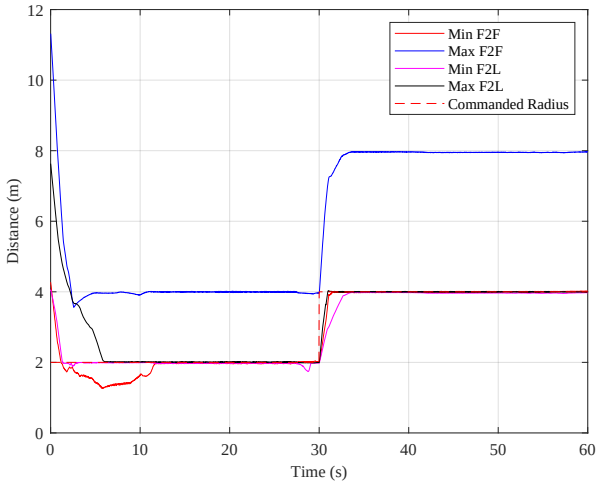

Figure 3.9: Adaptive response to commanded distance adjustments - reconfiguration during the test phase

As mentioned in section 3, the input encoding uses the error between current and commanded distance. Therefore, one of the advantages of the encoding and learning method in this chapter is that the learned policies are independent of the commanded distance. The commanded distance can be changed after training since the SNN uses the distance error. Figure 3.9 shows the agents' response to changes in commanded distance after training. According to this figure, when the commanded distance is changed at 30 seconds, the swarm immediately responds to this change in 2.98 seconds without disrupting the formation or any collision.

After 600 seconds, the leader is changed into an obstacle, and its reward coefficient $\mathcal{C}_F^L$ is changed to 0.0175. The reward sign function, tanh in (3.11), is also changed to -1, so the reward function for the leader is changed as follows,

$$\mathcal{R}_{Fi}^L(t) = -\mathcal{C}_{Fi}^L \left[r_{Fi}^L(t-1) - r_{Fi}^L(t) \right] \tag{3.26}$$

When the leader is transformed into an obstacle, the encoding equation for the input layer

must be changed. This is because the obstacle has no commanded distance, and the agents must maintain a commanded distance only from each other. Therefore, the commanded distance from the obstacle encoder in the input layer must be removed. Therefore, (3.1) can then be rewritten as follows:

$$\mu_I(\phi_i) = \exp\left(-\frac{(\phi_i - \zeta)^2}{2\sigma^2}\right) \tag{3.27}$$

The simulation proceeded for an additional 1200 seconds, during which the synaptic weights were adjusted in accordance with the new reward function given by (3.26). In this case, because the leader is now an obstacle, so then $\alpha^{\mathcal{S}}(t)$ in Figure 3.5 switches to 1 and training starts again with random signals to the output layer.

The results of the reward change are shown in Figures 3.10 and 3.11, which indicates that the agents quickly reduced their initial distance to the commanded distance of 2 m. Simultaneously, the minimum distance from the obstacle, which was the leader, increased over time, indicating that the agents adapted their behavior to maintain a greater distance from the obstacle. Figure 3.10 shows the trajectory of each agent after the reward change.

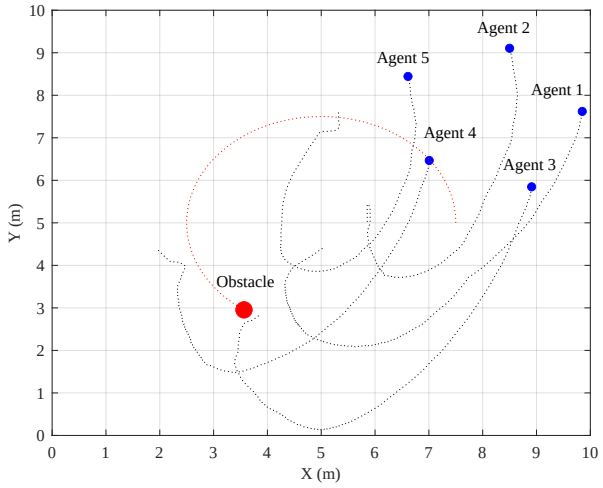

Figure 3.10: Trajectory adaptations of following agents in response to reward change for the leader during the test phase.

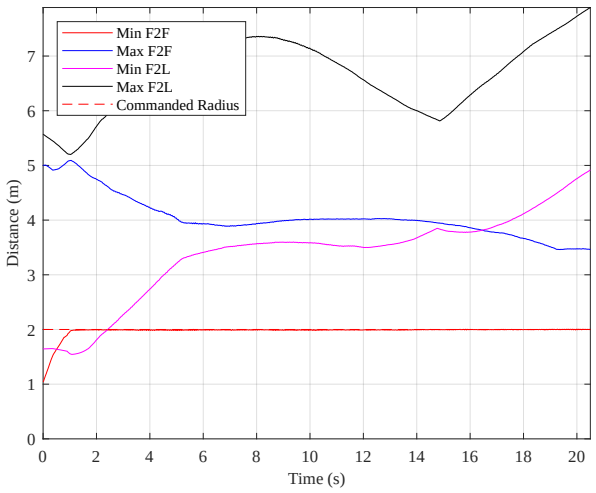

Figure 3.11: Variations in distances after reward changes and Leader becomes Obstacle - test phase.

Each agent in the swarm detects two neighboring agents and the leader, acquiring the Line-of-Sight angle and distance. Their neural network consists of three sub-layers in the input layer: two for neighboring followers (F_1 and F_2) and one for the leader (L), with inputs encoded using Gaussian Receptive Fields and fuzzy membership functions. The network uses differences between current and commanded distances within the swarm (r_{cmd}) and between followers and the leader (R_{cmd}) to stimulate input neurons, which are split into parts for distances greater than and within the commanded distance, encoding the LOS angle with fuzzy membership functions. Errors in distances are transformed into amplitude values using the *tanh* function, bounded between 0 and 1. Each agent's neural network includes 72 input neurons, with 24 neurons per detected object, derived from numerical simulations. The output layer has 4 neurons representing positive and negative Δx and Δy movements.

In order to better understand the effect of reward change on the SNN, the synaptic weights matrix before and after reward change has been illustrated in Figures 3.12 and 3.13, the vertical axis shows the output neurons. The first output neuron is for negative displacement in the x-direction, while the second output neuron is dedicated to positive displacement in the x-direction. Similarly, the third output neuron corresponds to negative displacement in the y-direction and the fourth output neuron to positive displacement in the y-direction. The horizontal axis shows the input neurons. The neuron IDs from 1 to 24 are for the first sub-layer dedicated to the neighboring follower. The network has two sub-layers for the neighboring follower agents, but only one is shown since they are similar in the case of synaptic weight values. The neuron numbers from 25 to 48 are for the sub-layer dedicated to the leader. The RCSE method aims to maintain the synaptic weight matrix gradient while adapting to changes in the reward signal.

Considering the numerical values presented in Table 3.2 along with the reward coefficients $\mathcal{C}_{Fi}^{Fj} = 0.02$ and $\mathcal{C}_{L}^{Fj} = 0.07$, and $r_{Fi}^{Fj} = 1\ m/s$ and $r_{Fi}^{L} = 0.1\ m/s$, the maximum rewards at each weight update interval (ΔT) for $\mathcal{R}_{Fi}^{Fj}$ and $\mathcal{R}_{Fi}^{L}$ are calculated using (3.10) and (3.11) as 4×10^{-4} and 7.7×10^{-4}, respectively. Consequently, $\mathcal{R}_{max}^{G} = \max(\mathcal{R}_{Fi}^{Fj}, \mathcal{R}_{Fi}^{L}) = 7.7 \times 10^{-4}$. The Ψ^S for the follower section in the network is $\left[\frac{4\times10^{-4}}{7\times10^{-4}}\right] 15.5 = 8.0519$, and for the leader

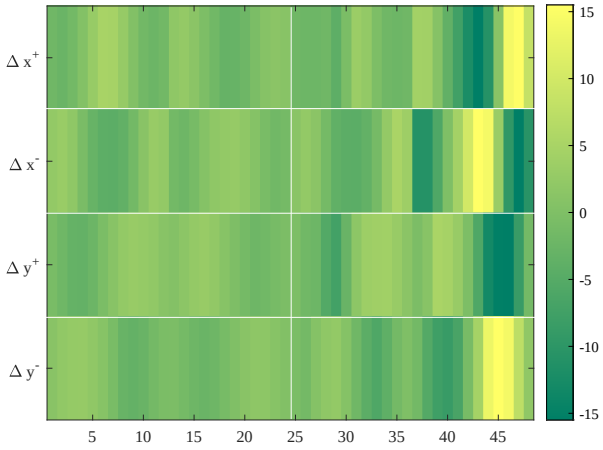

Figure 3.12: Synaptic Weights before Reward change.

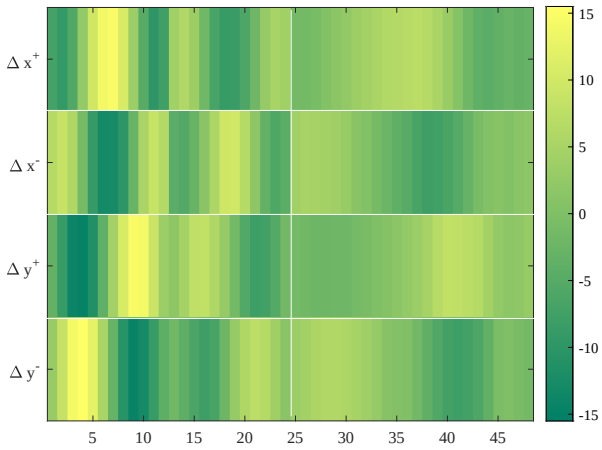

Figure 3.13: Synaptic Weights after Reward change.

section, it is $\left[\frac{7\times10^{-4}}{7\times10^{-4}}\right] 15.5 = 15.5$. The η and β for the follower section within the network are $\frac{\mathcal{AR}^{G}_{max}\ln(2^{\lambda^2}-1)}{\lambda\Psi^{\mathcal{S}}\log(2)} = 0.0011$ and $\beta = \frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}} = 2.152$, respectively. For the leader section, these values are $\frac{\mathcal{AR}^{G}_{max}\ln(2^{\lambda^2}-1)}{\lambda\Psi^{\mathcal{S}}\log(2)} = 5.719\times10^{-4}$ and $\beta = \frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}} = 1.118$, respectively.

The visual patterns observed in the synaptic weights matrix in Figure 3.12 and 3.13, specifically, the gradual increases and decreases in values across weights, directly result from applying Gaussian membership functions for encoding. As illustrated in the heatmap visualization, regions of higher values denote areas closer to the function's center, where the degree of membership peaks. Conversely, areas of lower values reflect points moving away from the center, where the membership degree decreases according to the Gaussian distribution's tails. Figure 3.5 further demonstrates this behavior. The differences in synaptic strengths arise from variations in spike rates, reflecting differences in membership degree.

Since the reward coefficients for followers and leaders are different, their maximum allowed synaptic weights are also different. The proposed method given by (3.12) for controlling the unbounded growth of synaptic weights has successfully stabilized the network.

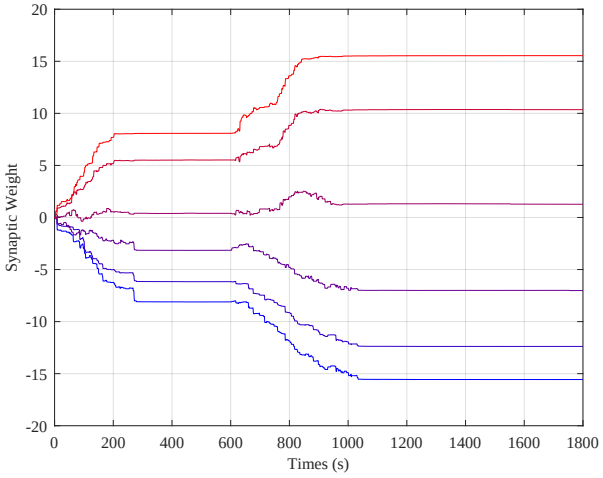

Figure 3.14: Synaptic weights increase after reward change.

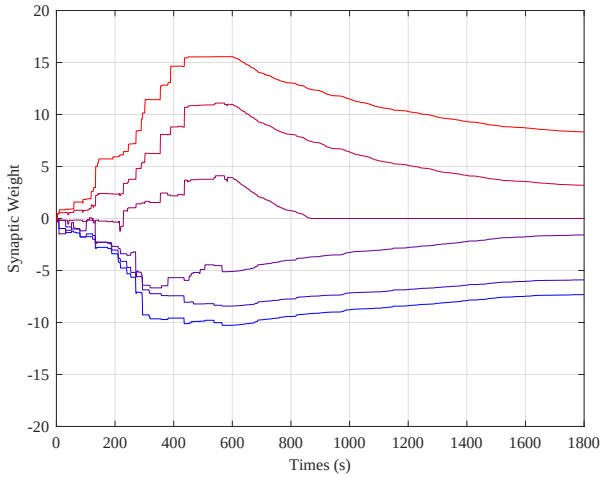

Figure 3.15: Synaptic weights decrease after reward change.

Figures 3.14 and 3.15 show the synaptic weights after the reward change. In this case, since the reward coefficients are changed, the η and β values in (3.14) are changed for the represented sub-layers, and the proposed method has helped the R-STDP algorithm to adjust the weights based on the new situation in the environment.

3.3.2 Simulation with FL and RCSE

In this section, the proposed aggregation algorithm is tested. In this case, the agents only send their models when the Euclidean distance between the current and previously published model or the latest global model reaches a threshold. In the first phase, the simulation was done in 600 seconds, and the agents learned to follow the leader. The threshold for publishing the agents' and server models was 0.0005 and 0.00051, respectively. The reason for choosing the server's threshold higher than the agents' is that as soon as the first agent sends its model to the server, the Euclidean distance between the current and previously published model by the server reaches 0.0005, and the server distributes the model immediately. Therefore, the serve's threshold is set higher than the agents' threshold, so it waits for the other agents

to send their models.

Figure 3.16 shows the distances between agents and the leader before the reward change, indicating that the agents converge to the solution faster than in the scenario where federated learning has not been used, without any error. Figure 3.17 shows the simulation results for the reward change scenario, indicating that the proposed event-triggered FL method has improved the learning performance, enabling the swarm to converge to the solution in 6 seconds. In this case, the neural network learned the policies in less time while preserving the performance.

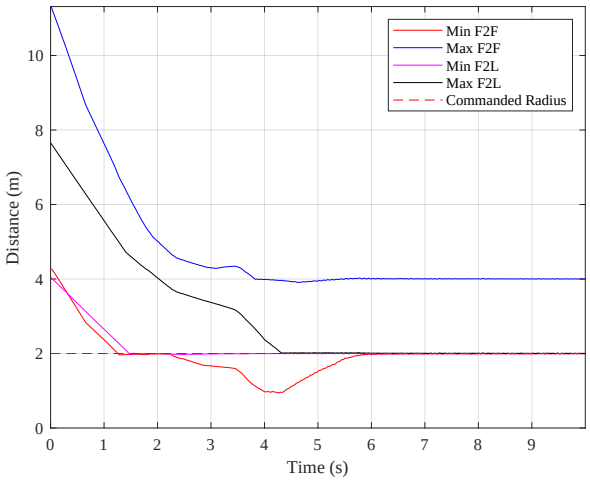

Figure 3.16: Distances during the test phase before reward change in the proposed event-triggered FL method.

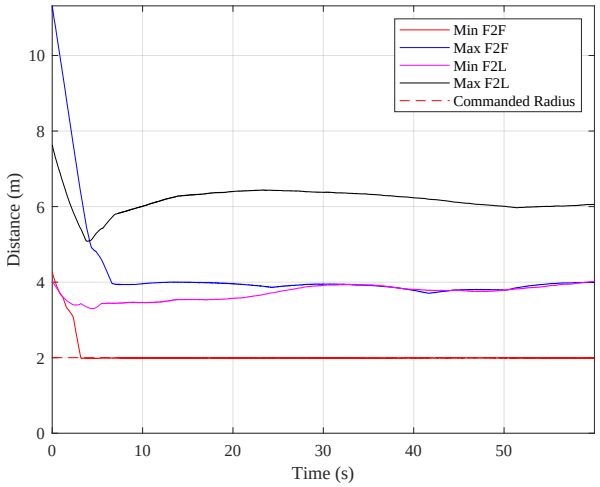

Figure 3.17: Distances during the test phase after reward change in the proposed event-triggered FL method.

The norm of the synaptic weights can represent the changes in the synaptic weights due to the change in the environment during the training process, which can be used to adjust the learning process in SNNs. In the proposed event-triggered FL method, agents communicate with the Central Server (leader) during training. According to Figure 3.19, the aggregation step time is small at the beginning of the training and increases as the SNN models converge to the final solution. The rate of change of the norm of the synaptic weights determines the

communication sample time of the aggregation process, which results in small aggregation intervals when the change rate is high and larger intervals when it reduces.

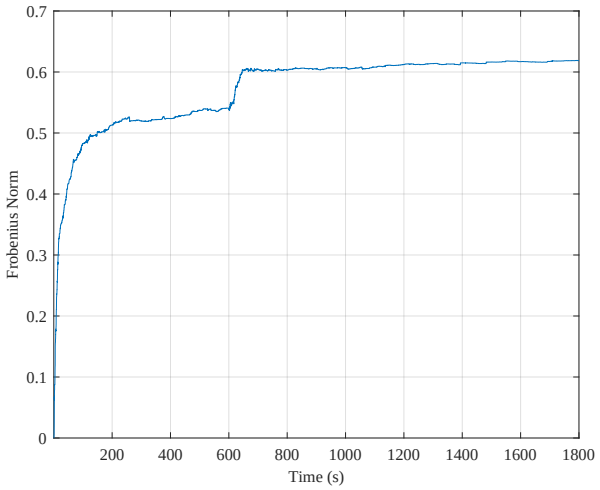

Figure 3.18: Frobenius norm of the Agent 1's weighs during the learning phase. The reward changes for the Leader after 600 s.

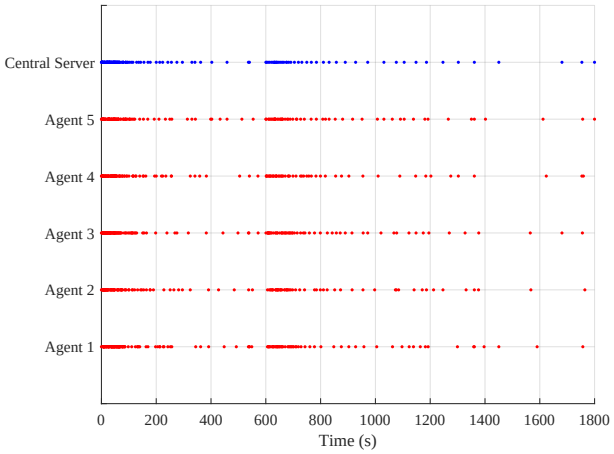

Figure 3.19: Communication times for agents and the Central Server (Leader). Red and blue dots show the times that agents and the Central Server have sent their model, respectively.

Therefore, the aggregation frequency is very high initially, and it reduces after the change in synaptic weights goes to zero because the learning rate matrix in (3.13) goes to zero. Also, after the reward changes and the leader becomes an obstacle after 600 seconds, the Euclidean distance between the converged and current models increases and reaches the threshold. As soon as the first agent sends its model to the server, the aggregation process starts again, and the agents adjust the associated synaptic weight.

Table 3.3: Comparative Performance Analysis of the Proposed Aggregation Algorithm+RCSE and RCSE

	FL+RCSE	**RCSE**
Learning (Convergence) Time - Training Phase (s)	261.92	554.71
Max Distance Convergence Error - Test Phase (%)	0.71	1.95
Convergence Time for Distance - Test Phase (s)	5.81	8.94

Table 3.3 compares the results and focuses on three critical metrics: learning time, maximum error after convergence, and convergence time. The proposed FL algorithm demonstrates significant improvements in terms of efficiency and accuracy, as evidenced by its considerably shorter learning and convergence times and a notable reduction in error after convergence.

3.4 Conclusion

In this chapter, we presented a comprehensive approach addressing the challenges of uncontrolled growth in synaptic weights and the limited responsiveness of R-STDP to real-time changes within SNNs. Our proposed solution integrates the RCSE method with a dynamic aggregation interval in FL and significantly reduces learning time while improving performance.

The R-CSE method introduces a novel mechanism to manage the unbounded growth of synaptic weights by dynamically adjusting the decay rate through the SoftPlus function. This adjustment is sensitive to the learning stages and rewards changes, ensuring that synaptic weight adjustments remain responsive over time. By addressing the challenge of synaptic weight saturation, the RCSE method facilitates a balanced approach to weight adjustment, preventing network saturation and promoting continuous learning adaptability.

We introduce a novel approach that uses FL in SNN and employs the Frobenius norm

to adjust weighted aggregation in FL. Additionally, we include weight decay proportional to the time elapsed since an agent's last model publication. This improves the efficiency and responsiveness of the learning process.

Our proposed method adjusts its aggregation time based on the Euclidean norm. This metric measures the distance between the weight matrices of the agents and the server, determining reduced intervals for model publication. Our results show that the proposed aggregation method significantly accelerates agents' learning while reducing the distance convergence error (Table 3.3).

Moreover, the dynamic aggregation interval effectively reduces communication overhead between the agents and the central server, particularly after model converges and the algorithm switches from the training mode to the operational mode. This reduction is critical when communication bandwidth is limited or costly. This approach is particularly advantageous in 5G networks, where efficient bandwidth use can enhance the overall throughput and reduce latency in real-time applications. Moreover, the adaptive use of communication resources aligns with the scalable and flexible infrastructure of 5G, optimizing network performance even during peak demand periods.

Chapter 4

Proposed Investigations

The integration of SNN with FL presents a novel approach to multiagent reinforcement learning on the EDGE, promising to enhance learning efficiency and data privacy simultaneously. This approach, however, introduces several challenges and opportunities for further investigation. Future work will focus on four primary areas:

- **Improving Stable Learning in SNNs:** Addressing the challenge of learning stability and synaptic weight control in SNNs during reinforcement learning, particularly in response to dynamic environmental rewards.
- **Managing Communication Overhead in FL:** Reducing communication overhead among agents while ensuring responsiveness to changes through adaptive communication strategies.
- **Heterogeneity of Spiking Neuron Models:** Exploring the implications of neural network heterogeneity in spiking neuron models on collective learning outcomes and system performance.
- **Resilience to Model Poisoning Attacks:** Investigating the system's vulnerability to model poisoning attacks during FL and developing strategies to mitigate such risks.

- **Utilization of Advanced Robotic and 5G Laboratory Setup:** Developing the comprehensive laboratory environment to conduct experiments that test and optimize algorithms for autonomous navigation and machine learning, utilizing 5G connectivity to enhance real-time data processing and control in complex robotic networks.

4.1 Improving Stable Learning in SNNs

The stability of learning in SNN is a foundational aspect for their deployment in dynamic environments, necessitating precise adjustments of synaptic weights in response to probable changes in the environment and the reward signals. In Chapter 3, the adaptation mechanisms essential for maintaining the learning processes' stability, emphasizing the equilibrium between synaptic plasticity and homeostasis, were investigated to ensure that the neurons do not become overly excitable or unresponsive over time. This involved developing algorithms that dynamically adjust learning rates and decay rates based on reward functions. Furthermore, the complexities of incorporating these adaptive mechanisms within the FL framework were addressed, ensuring that distributed SNNs can learn from decentralized data sources while adapting to environmental changes.

While this research marks substantial progress, it also highlights the emergence of challenges, particularly in understanding and enhancing swarm behavior in contexts where a leader's presence and dynamic obstacles coexist. Future work will propose methods and algorithms focusing on the effects of these dual factors on swarm dynamics and how a swarm can adeptly react to dynamic obstacles while following a leader and maintaining its formation.

4.2 Managing Communication Overhead in FL

Efficient communication among distributed agents is a cornerstone of FL, particularly in EDGE computing environments where bandwidth and latency can be significant constraints. Reducing communication overhead without compromising the system's responsiveness to

changes presents a unique challenge. Initially, strategies for improving information distribution during the training in FL should be explored. This involves investigating adaptive communication frameworks that dynamically adjust the frequency of model exchange based on current learning conditions. Secondly, the impact of reduced communication on the learning process should be studied, ensuring that critical information necessary for model updates is not lost. This includes developing algorithms to identify the most impactful updates to transmit, reducing redundancy without sacrificing learning efficacy. These methods will enhance local computations and inferences to reduce communication burden while maintaining system coherence and performance.

In Chapter 3, an algorithm was developed for event-triggered aggregation of FL, which addresses managing communication overhead by utilizing a threshold mechanism for model updates. For future work, we propose extending this model with an additional index or threshold to further improve the communication overhead, especially in scenarios with larger swarms of agents. For instance, if we have 100 agents and the current threshold criteria are met by 50 agents, this still results in a substantial number of models being sent to the central server. Introducing an extra index (or threshold) like information entropy for measuring the amount of data in each neural network model or peer-to-peer communication between agents could reduce the number of models sent while ensuring the global model's integrity and efficacy.

4.3 Heterogeneity of Spiking Neuron Models on Server

Incorporating diversity in the spiking neuron models within multiagent systems presents challenges and opportunities for synchronizing learning processes and achieving consensus in shared tasks. The initial focus will delve into how this diversity impacts individual and collective learning performance. This includes examining variations in learning efficiency, behavioral responses, and adaptability to diverse tasks. Studies prove the benefits of this heterogeneity, revealing that various neuron models significantly improve task performance, particularly in tasks with complex temporal structures. Examples of such tasks include

the precise coordination of robotic movements in dynamic environments and the complex decision-making required for autonomous driving, where the diversity in spiking neuron models enables the systems to process temporal sequences more effectively and respond to intricate, time-dependent challenges (e.g., processing LIDAR data).

Moreover, this diversity has been linked to enhanced generalization capabilities, especially when the learning parameters are tuned inaccurately [71]. Insights from previous studies indicate that heterogeneity in neuronal and synaptic dynamics can diminish spiking activity while simultaneously enhancing system performance, suggesting a pathway to achieving spike-efficient learning that conserves energy without compromising on learning efficacy [72].

This research aims to assess network model diversity's influence on the multiagent framework's robustness and resilience, particularly within FL. To address this challenge, the future investigation includes exploring three potential methodologies: Neuroevolution of Augmenting Topologies (NEAT) algorithm, Transfer Learning, and Evolutionary Algorithms such as Particle Swarm Optimization (PSO) and Genetic Algorithms (GA). Each of these methods offers unique advantages for optimizing the global model. NEAT, for instance, can evolve and optimize network topologies and parameters to adapt to various tasks dynamically. Transfer Learning, on the other hand, allows for leveraging pre-learned behaviors and patterns across different but related tasks, facilitating quicker adaptation. Evolutionary algorithms can optimize the global model parameters during the model aggregation process to achieve a cohesive and high-performing system.

4.4 Resilience to Model Poisoning Attacks

Model poisoning attacks during FL pose significant threats to the integrity and reliability of the learning process. The first step in addressing this challenge involves identifying vulnerabilities within the SNN-FL integration that could be exploited by such attacks. This will be followed by developing detection mechanisms for identifying anomalous behavior or compromised nodes within the federated network. Techniques such as Differential Privacy

(DP) and robust statistical methods, including multi-Krum, can be good candidates. For example, the DP algorithm introduces randomness into the neural network model during the learning process, ensuring that the contribution of individual updates does not significantly affect the aggregated model. This makes it difficult for attackers to have their poisoned updates significantly influence the model's behavior [73]. On the other hand, multi-Krum selects the most reliable update from among the participants based on a distance metric that identifies updates likely to be benign and representative of the true model.

4.5 Utilization of Advanced Robotic and 5G Laboratory Setup

In the next phase of research, the advanced laboratory that has been established by our team will be utilized. This laboratory is equipped with small Wi-Fi drones, specifically DJI Tello EDU models, and an X500 Holybro drone serving as an aerial server with dual Wi-Fi and 5G capabilities. Additionally, two Clearpath Chameleon R100 UGVs, which are integrated with cameras and 2D LIDAR sensors, are included in the setup. All these devices are interconnected through a comprehensive ROS2 framework, ensuring seamless communication and efficient data transfer are maintained across the system via a robust 5G network. Once the construction of the laboratory is finalized, experiments will be carried out to evaluate and enhance the algorithms that have been developed during the PhD program. These algorithms leverage 5G connectivity to facilitate real-time data transmission and processing, which is crucial for machine learning applications and for coordinating the operations of drone swarms and UGVs in complex scenarios. The integration of 5G technology is pivotal for enabling higher data throughput and reduced latency, essential for real-time control and decision-making processes in autonomous robotic systems.

These investigations will pave the way for more resilient, efficient, and adaptive multi-agent systems capable of leveraging the strengths of both SNNs and FL in dynamic and potentially adversarial environments.

Bibliography

[1] Y. Venkatesha, Y. Kim, L. Tassiulas, and P. Panda, "Federated learning with spiking neural networks," *IEEE Transactions on Signal Processing*, vol. 69, pp. 6183–6194, 2021.

[2] N. Skatchkovsky, H. Jang, and O. Simeone, "Federated neuromorphic learning of spiking neural networks for low-power edge intelligence," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8524–8528, IEEE, 2020.

[3] S. A. Tumpa, S. Singh, M. F. F. Khan, M. T. Kandemir, V. Narayanan, and C. R. Das, "Federated learning with spiking neural networks in heterogeneous systems," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 1–6, IEEE, 2023.

[4] N. Skatchkovsky, H. Jang, and O. Simeone, "Spiking neural networks—part iii: Neuromorphic communications," *IEEE Communications Letters*, vol. 25, no. 6, pp. 1746–1750, 2021.

[5] J. L. Lobo, J. Del Ser, A. Bifet, and N. Kasabov, "Spiking neural networks and online learning: An overview and perspectives," *Neural Networks*, vol. 121, pp. 88–100, 2020.

[6] S. Dora and N. Kasabov, "Spiking neural networks for computational intelligence: an overview," *Big Data and Cognitive Computing*, vol. 5, no. 4, 2021.

[7] K. Xie, Z. Zhang, B. Li, J. Kang, D. Niyato, S. Xie, and Y. Wu, "Efficient federated learning with spike neural networks for traffic sign recognition," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 9, pp. 9980–9992, 2022.

[8] W. J. Yun, Y. Kwak, H. Baek, S. Jung, M. Ji, M. Bennis, J. Park, and J. Kim, "Slimfl: Federated learning with superposition coding over slimmable neural networks," *IEEE/ACM Transactions on Networking*, 2023.

[9] H. Yang, K.-Y. Lam, L. Xiao, Z. Xiong, H. Hu, D. Niyato, and H. Vincent Poor, "Lead federated neuromorphic learning for wireless edge artificial intelligence," *Nature communications*, vol. 13, no. 1, pp. 42–69, 2022.

[10] A. M. AbdelAty, M. E. Fouda, and A. M. Eltawil, "On numerical approximations of fractional-order spiking neuron models," *Communications in Nonlinear Science and Numerical Simulation*, vol. 105, 2022.

[11] Y.-H. Liu and X.-J. Wang, "Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron," *Journal of computational neuroscience*, vol. 10, pp. 25–45, 2001.

[12] L. Long and G. Fang, "A review of biologically plausible neuron models for spiking neural networks," *AIAA Infotech@ Aerospace 2010*, 2010.

[13] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

[14] A. Triche, A. S. Maida, and A. Kumar, "Exploration in neo-hebbian reinforcement learning: Computational approaches to the exploration–exploitation balance with bio-inspired neural networks," *Neural Networks*, vol. 151, pp. 16–33, 2022.

[15] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[16] D. L. Young and C.-S. Poon, "Hebbian covariance learning: a nexus for respiratory variability, memory, and optimization?," *Advances in Modeling and Control of Ventilation*, pp. 73–83, 1998.

[17] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of mathematical biology*, vol. 15, pp. 267–273, 1982.

[18] T. Toyoizumi, J.-P. Pfister, K. Aihara, and W. Gerstner, "Generalized bienenstock–cooper–munro rule for spiking neurons that maximizes information transmission," *Proceedings of the National Academy of Sciences*, vol. 102, no. 14, pp. 5239–5244, 2005.

[19] A. Soltoggio and J. J. Steil, "Solving the distal reward problem with rare correlations," *Neural computation*, vol. 25, no. 4, pp. 940–978, 2013.

[20] H. S. Seung, "Learning in spiking neural networks by reinforcement of stochastic synaptic transmission," *Neuron*, vol. 40, no. 6, pp. 1063–1073, 2003.

[21] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral cortex*, vol. 17, pp. 2443–2452, 2007.

[22] A. L. Hodgkin and A. F. Huxley, "Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo," *The Journal of physiology*, vol. 116, no. 4, 1952.

[23] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[24] S. Song, K. D. Miller, and L. F. Abbott, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neuroscience*, vol. 3, no. 9, pp. 919–926, 2000.

[25] H. Z. Shouval, S. S.-H. Wang, and G. M. Wittenberg, "Spike timing dependent plasticity: a consequence of more fundamental learning rules," *Frontiers in computational neuroscience*, vol. 4, 2010.

[26] H. Markram, W. Gerstner, and P. J. Sjöström, "A history of spike-timing-dependent plasticity," *Frontiers in synaptic neuroscience*, vol. 3, 2011.

[27] V. Vassiliades, A. Cleanthous, and C. Christodoulou, "Multiagent reinforcement learning: spiking and nonspiking agents in the iterated prisoner's dilemma," *IEEE transactions on neural networks*, vol. 22, no. 4, pp. 639–653, 2011.

[28] F. Ponulak and J. J. Hopfield, "Rapid, parallel path planning by propagating wavefronts of spiking neural activity," *Frontiers in computational neuroscience*, vol. 7, 2013.

[29] D. Probst, M. A. Petrovici, I. Bytschok, J. Bill, D. Pecevski, J. Schemmel, and K. Meier, "Probabilistic inference in discrete spaces can be implemented into networks of lif neurons," *Frontiers in computational neuroscience*, vol. 9, 2015.

[30] U. Markowska-Kaczmar and M. Koldowski, "Spiking neural network vs multilayer perceptron: who is the winner in the racing car computer game," *Soft Computing*, vol. 19, pp. 3465–3478, 2015.

[31] S. Guo, Z. Yu, F. Deng, X. Hu, and F. Chen, "Hierarchical bayesian inference and learning in spiking neural networks," *IEEE transactions on cybernetics*, vol. 49, no. 1, pp. 133–145, 2017.

[32] A. Tavanaei and A. Maida, "Bp-stdp: Approximating backpropagation using spike timing dependent plasticity," *Neurocomputing*, vol. 330, pp. 39–47, 2019.

[33] Z. Bing, C. Meschede, G. Chen, A. Knoll, and K. Huang, "Indirect and direct training of spiking neural networks for end-to-end control of a lane-keeping vehicle," *Neural Networks*, vol. 121, pp. 21–36, 2020.

[34] C. Hur, B. Ibrokhimov, and S. Kang, "N3-cpl: Neuroplasticity-based neuromorphic network cell proliferation learning," *Neurocomputing*, vol. 411, pp. 193–205, 2020.

[35] L. Salt, D. Howard, G. Indiveri, and Y. Sandamirskaya, "Parameter optimization and learning in a spiking neural network for uav obstacle avoidance targeting neuromorphic processors," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3305–3318, 2019.

[36] J. Liu, H. Lu, Y. Luo, and S. Yang, "Spiking neural network-based multi-task autonomous learning for mobile robots," *Engineering Applications of Artificial Intelligence*, vol. 104, 2021.

[37] H. Lu, J. Liu, Y. Luo, Y. Hua, S. Qiu, and Y. Huang, "An autonomous learning mobile robot using biological reward modulate stdp," *Neurocomputing*, vol. 458, pp. 308–318, 2021.

[38] Q. Zhan, G. Liu, X. Xie, G. Sun, and H. Tang, "Effective transfer learning algorithm in spiking neural networks," *IEEE Transactions on Cybernetics*, vol. 52, no. 12, pp. 13323–13335, 2021.

[39] D. Chu and H. Le Nguyen, "Constraints on hebbian and stdp learned weights of a spiking neuron," *Neural Networks*, vol. 135, pp. 192–200, 2021.

[40] S. Schmidgall, J. Ashkanazy, W. Lawson, and J. Hays, "Spikepropamine: Differentiable plasticity in spiking neural networks," *Frontiers in neurorobotics*, vol. 15, 2021.

[41] B. Golosio, G. Tiddia, C. De Luca, E. Pastorelli, F. Simula, and P. S. Paolucci, "Fast simulations of highly-connected spiking cortical models using gpus," *Frontiers in Computational Neuroscience*, vol. 15, 2021.

[42] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," *arXiv preprint arXiv:1907.02189*, 2019.

[43] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

[44] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International conference on machine learning*, pp. 5132–5143, PMLR, 2020.

[45] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "A novel framework for the analysis and design of heterogeneous federated learning," *IEEE Transactions on Signal Processing*, vol. 69, pp. 5234–5249, 2021.

[46] Q. Li, B. He, and D. Song, "Model-contrastive federated learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10713–10722, 2021.

[47] D. Ye, R. Yu, M. Pan, and Z. Han, "Federated learning in vehicular edge computing: A selective model aggregation approach," *IEEE Access*, vol. 8, pp. 23920–23935, 2020.

[48] K. Yang, T. Jiang, Y. Shi, and Z. Ding, "Federated learning via over-the-air computation," *IEEE transactions on wireless communications*, vol. 19, no. 3, pp. 2022–2035, 2020.

[49] Y. Zhan, P. Li, Z. Qu, D. Zeng, and S. Guo, "A learning-based incentive mechanism for federated learning," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6360–6368, 2020.

[50] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, 2021.

[51] M. Chen, H. V. Poor, W. Saad, and S. Cui, "Convergence time optimization for federated learning over wireless networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 4, pp. 2457–2471, 2020.

[52] M. A. Ferrag, O. Friha, L. Maglaras, H. Janicke, and L. Shu, "Federated deep learning for cyber security in the internet of things: Concepts, applications, and experimental analysis," *IEEE Access*, vol. 9, pp. 138509–138542, 2021.

[53] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 269–283, 2020.

[54] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1935–1949, 2020.

[55] A. B. Mansour, G. Carenini, A. Duplessis, and D. Naccache, "Federated learning aggregation: New robust algorithms with guarantees," in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 721–726, IEEE, 2022.

[56] A. Jarwan and M. Ibnkahla, "Edge-based federated deep reinforcement learning for iot traffic management," *IEEE Internet of Things Journal*, vol. 10, no. 5, pp. 3799–3813, 2022.

[57] Z. Yuan, Z. Wang, X. Li, L. Li, and L. Zhang, "Hierarchical trajectory planning for narrow-space automated parking with deep reinforcement learning: A federated learning scheme," *Sensors*, vol. 23, no. 8, 2023.

[58] Q. Wu, X. Chen, T. Ouyang, Z. Zhou, X. Zhang, S. Yang, and J. Zhang, "Hiflash: Communication-efficient hierarchical federated learning with adaptive staleness control and heterogeneity-aware client-edge association," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 5, pp. 1560–1579, 2023.

[59] G. Xia, J. Chen, C. Yu, and J. Ma, "Poisoning attacks in federated learning: A survey," *IEEE Access*, vol. 11, pp. 10708–10722, 2023.

[60] M. Pachter, E. Garcia, and D. W. Casbeer, "Differential game of guarding a target," *Journal of Guidance, Control, and Dynamics*, vol. 40, pp. 2991–2998, 2017.

[61] I. E. Weintraub, M. Pachter, and E. Garcia, "An introduction to pursuit-evasion differential games," in *2020 American Control Conference (ACC)*, pp. 1049–1066, 2020.

[62] E. Garcia, "Cooperative target protection from a superior attacker," *Automatica*, vol. 131, 2021.

[63] M. D. Awheda and H. M. Schwartz, "A decentralized fuzzy learning algorithm for pursuit-evasion differential games with superior evaders," *Journal of Intelligent & Robotic Systems*, vol. 83, pp. 35–53, 2016.

[64] A. D. Bird, P. Jedlicka, and H. Cuntz, "Dendritic normalisation improves learning in sparsely connected artificial neural networks," *PLOS Computational Biology*, vol. 17, 2021.

[65] J. L. Lobo, I. Laña, J. Del Ser, M. N. Bilbao, and N. Kasabov, "Evolving spiking neural networks for online learning over drifting data streams," *Neural Networks*, vol. 108, pp. 1–19, 2018.

[66] M. Tayefe Ramezanlou, H. Schwartz, I. Lambadaris, M. Barbeau, and S. H. R. Naqvi, "Learning a policy for pursuit-evasion games using spiking neural networks and the stdp algorithm," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1918–1925, 2023.

[67] P. Qi, D. Chiaro, A. Guzzo, M. Ianni, G. Fortino, and F. Piccialli, "Model aggregation techniques in federated learning: A comprehensive survey," *Future Generation Computer Systems*, vol. 150, pp. 272–293, 2024.

[68] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[69] Z. Wang, Z. Peng, X. Fan, Z. Wang, S. Wu, R. Yu, P. Yang, C. Zheng, and C. Wang, "Fedave: Adaptive data value evaluation framework for collaborative fairness in federated learning," *Neurocomputing*, vol. 574, 2024.

[70] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *Advances in neural information processing systems*, vol. 33, pp. 7611–7623, 2020.

[71] N. Perez-Nieves, V. C. Leung, P. L. Dragotti, and D. F. Goodman, "Neural heterogeneity promotes robust learning," *Nature communications*, vol. 12, no. 1, 2021.

[72] B. Chakraborty and S. Mukhopadhyay, "Heterogeneous neuronal and synaptic dynamics for spike-efficient unsupervised learning: Theory and design principles," *arXiv preprint arXiv:2302.11618*, 2023.

[73] E. Bagdasaryan, O. Poursaeed, and V. Shmatikov, "Differential privacy has disparate impact on model accuracy," *Advances in neural information processing systems*, vol. 32, 2019.