

Università degli Studi di Messina

Dipartimento di scienze biomediche, odontoiatriche e delle immagini morfologiche e funzionali

dottorato in bioingegneria applicata alle scienze mediche XXXVIII Cycle

---

Neuromorphic and Bio-Inspired Learning Strategies for Sustainable and Low-Power AI

Student:
Rawan Nawaiseh

Advisor: **Prof. Dario Bruneo**
S.S.D: **IINF-05/A**
Coordinator: **Prof. Michele Gaeta**

Academic Year 2024 - 2025

Contents

Abstract **5**

Introduction **6**

1 Background **10**

1.1 Spiking Neural Networks 11

1.1.1 Definition and Biological Inspiration 12

1.1.2 Neuron Models 13

1.1.3 Spike Encoding Schemes 17

1.2 Astrocytes 18

1.3 Advantages and Challenges of SNNs 20

2 Biological Models **22**

2.1 Biologically Inspired Learning Rules 22

2.1.1 Spike-Timing-Dependent Plasticity (STDP) 22

2.1.2 Anti-Spike-Timing-Dependent Plasticity (Anti-STDP) . . . 24

2.1.3 Reward-Modulated STDP (R-STDP) 25

2.1.4 Triplet STDP / Other Variants. 25

3 Astrocytes-SNN **27**

3.1 Intoduction and Motivation 27

3.2 Proposed Methodology 28

3.2.1 Tripartite Synapse 29

3.2.2 Astrocyte Model 30
3.2.3 Astro-Reward-SNN Architecture 31
3.2.4 Learning Rule Mechanism: Astro-Reward-SNN 32
3.3 Experiments and Results 34

4 Spiking Anomaly Detection **37**
4.1 Deep learning Models for Anomaly Detection 37
4.2 Motivation for using SNNs in Anomaly Detection 39
4.3 Proposed SNN Model for Anomaly Detection 40
4.4 Methodology and Data Processing 42
4.5 Experiments and Results 45
4.5.1 BSNN Architecture and Parameters 45
4.5.2 Training Phase 46
4.5.3 Inference Phase 47
4.5.4 Real-Time simulation 48
4.5.5 Performance Evaluation 51
4.6 Stop Learning Mechanism 51
4.6.1 Proposed Modulated BSNN (MBSNN) 51
4.6.2 Results and Discussion 54
4.7 Summary 58

5 Spiking Autoencoders **60**
5.1 Introduction of Spiking Autoencoders 60
5.2 Proposed methodology 61
5.2.1 SAE Achitecture 62
5.2.2 Learning Rule 64
5.2.3 Application Scenarios: Reconstruction and Anomaly Detection 66
5.3 Results and Discussion 67
5.3.1 Reconstruction Results (MNIST) 68

5.3.2 Anomaly Detection Results (MFPT) 70
5.3.3 Comparative Analysis 73
5.4 Summary 74

6 Edge Intelligence Background **75**
6.1 From Cloud-Centric AI to Edge Intelligence 75
6.2 On-Device Training: Importance and Challenges 76
6.3 Toward Bio-Inspired Alternatives for Edge Learning 78
6.4 Forward–Forward Algorithm (Hinton, 2022) 78

7 FF-MCUs **83**
7.1 Introduction 83
7.2 Problem Statement 84
7.3 Use Case 84
7.4 Proposed μ-FF 85
7.5 Experiment and Results 88
7.5.1 Dataset 88
7.5.2 Hardware Settings 89
7.5.3 Results 91
7.6 Summary 94

8 Extended FF Framework **95**
8.1 Motivation for Extending FF 95
8.1.1 Vision Transformers (ViTs) 96
8.1.2 Echo State Networks (ESNs) 97
8.2 Proposed Algorithm 99
8.2.1 Model Architecture(ESN-ViT) 100
8.2.2 The μ-FF Algorithm 101
8.3 Experimental Results 103
8.3.1 Dataset and Experimental Setup 103
8.3.2 Hardware Considerations 104

8.3.3 Discussion 104
8.3.4 Classification Performance 105
8.3.5 Continuous Learning Scenario 106
8.3.6 On-Device Metrics 107
8.4 Comparative Analysis 108
8.5 Summary 109

9 Sparse-FF **111**
9.1 Introduction and Motivation 111
9.2 Proposed Sparse-FF 112
9.2.1 Model Architecture 112
9.3 Computational Complexity Analysis 116
9.3.1 Baseline Complexity in FF 117
9.3.2 Complexity Reduction in Sparse-FF 118
9.3.3 Total Complexity of Sparse-FF 119
9.4 Experiment and Results 120
9.5 Summary 126

10 Conclusions **127**

Bibliography **128**

Abstract

In the contemporary digital era, the proliferation of intelligent systems is increasing exponentially, challenging the capabilities of conventional computing paradigms. As an increasing number of applications migrate to the edge, the necessity for sustainable and low-power solutions has become increasingly urgent. This demand has contributed to the advancement of neuromorphic computing, a field inspired by the principles of biological neural processing. Neuromorphic computing offers an energy-efficient, online learning and adaptation alternative to traditional systems, especially under strict resource constraints.

Concurrently, biologically plausible and backpropagation-free approaches are emerging as complementary strategies that draw inspiration from the brain learning mechanisms, avoiding implausible mechanisms of traditional deep models. Neuromorphic and biological plausible approaches are shifting Artificial Intelligence (AI) towards more natural, adaptive, and sustainable systems.

This thesis explores neuromorphic and biologically plausible learning strategies, with a focus on their application in the design of low-power, sustainable intelligent systems. The investigation encompasses Spiking Neural Networks (SNNs), event-driven processing, and local plasticity-based rules. By comparing these paradigms across different application domains, the work highlights trade-offs in accuracy, latency, power consumption, and scalability. Experimental results demonstrate how carefully designed algorithms can achieve significant gains in efficiency without compromising performance, paving the way toward the next generation of intelligent, autonomous systems.

Introduction

Over the past decade, the rapid evolution of Artificial Intelligence (AI) has fundamentally reshaped the way we design, deploy, and interact with intelligent systems. The expansion of advanced technologies, connected devices, and increasingly sophisticated algorithms has enabled unprecedented capabilities in various domains, from industrial automation [1] to personalized healthcare [2]. However, this progress comes with growing challenges; the most important among them is the need to develop AI systems that are both computationally efficient and energy-saving, especially as intelligence moves from centralized servers to distributed edge and embedded platforms.

Machine learning and deep learning techniques are highly effective not only in extracting patterns from large datasets but also in solving a wide range of problems across diverse application domains. However, their success comes at the cost of high energy consumption, large data requirements, and considerable computational costs. At the same time, there is an increasing demand for autonomous machines capable of operating in real-world environments. Examples of such machines include self-driving vehicles, drones, and robotic systems, where energy efficiency, low latency, and real-time decision making are critical. These challenges make the deployment of conventional AI models in edge devices complicated. Therefore, during the last decade the push towards sustainable energy-efficient systems has stimulated the search for alternative learning strategies that can maintain performance while drastically reducing computational overhead.

One promising direction is neuromorphic computing, which takes inspiration

from the structure and dynamics of the biological brain with the concept of designing hardware and algorithms that process information through parallel, event-driven, and low-power neural circuits unlike traditional computing systems which rely on sequential, clock-driven operations. At the core of many neuromorphic systems are Spiking Neural Networks (SNNs) models that encode and process information using discrete events, or electrical pulses (spikes), rather than continuous-valued activations. This event-driven nature means that neurons only consume energy when spikes occur, enabling inherently low-power computation. Furthermore, what makes SNNs particularly advantageous over traditional models is that they incorporate temporal dynamics that capture time-dependent patterns, making them more suitable for sensory processing [3], robotics [4], and other applications involving continuous streams of data. These advantages enable highly energy-efficient processing, particularly when combined with biological learning rules such as Spike Timing-Dependent Plasticity (STDP) [5], Reward-Modulated STDP (R-STDP) [6], Triplet STDP [7], or homeostatic plasticity mechanisms [8]. These rules allow systems to learn and adapt in real time without the need for costly gradient backpropagation, making them suitable for always-on, low-power applications.

Recent findings in neuroscience show that, in addition to the plasticity of neurons, glial cells, in particular *astrocytes* play a key role in regulating synaptic activity through calcium signaling and gliotransmitter release [9], [10]. Introducing mechanisms inspired by astrocytes into SNNs provides a further biologically grounded pathway to enhance adaptability, stability, and energy-efficient learning, which will be a central focus of this thesis.

In parallel, there has been an increased interest in alternative learning strategies that, while not strictly neuromorphic, seek to improve learning efficiency by leveraging biological principles. One notable example is the Forward–Forward (FF) algorithm [11], which emulates the brain's unidirectional processing of information aligning more closely with the way neural system processes signals.

In particular, it replaces the traditional error backpropagation process with a two-phase comparison of "goodness" between positive and negative data representations. This eliminates the need for backward weight transport and large intermediate storage, reducing memory usage, and potentially enabling more hardware-friendly, energy-efficient training.

This thesis investigates the design and application of biologically inspired neuromorphic strategies for different tasks and bio-Inspired strategies beyond Back-Propagation (BP) for On-Device Training, with a focus on building sustainable, low-power AI systems. The study covers the theoretical foundations of neuromorphic learning rules, the mechanics of spiking neural networks, and the implementation of plasticity-based adaptation, alongside the analysis and deployment of the Forward–Forward algorithm and its variants. Both paradigms are evaluated across a range of application scenarios, including classification, anomaly detection, and embedded edge intelligence, with experimental results that highlight trade-offs in accuracy, latency, scalability, and energy efficiency.

The thesis is organized as follows. Chapter 1 introduces the background on neuromorphic computing and spiking neural networks. Chapter 2 presents biological learning models and plasticity mechanisms. Chapter 3 explores the integration of astrocytes in SNNs and evaluates their performance in a classification scenario. Chapter 4 presents the design and evaluation of neuromorphic learning strategies applied to anomaly detection. Chapter 5 describes the implementation of the spiking autoencoder for reconstruction and anomaly detection problems. Chapter 6 provides an overview of alternative biological learning strategies, with a particular focus on methods for on-device training aimed at creating more sustainable systems. Chapter 7 provides an in-depth examination of the Forward–Forward (FF) algorithm, focusing on its feasibility for deployment on embedded devices, with particular attention to resource-constrained platforms such as microcontrollers (MCUs). Chapter 8 extends the deployment of the FF algorithm on MCUs by integrating it with various deep architectures, with the aim of

further improving memory representation and efficiency. Chapter 9 investigates a variation of the FF algorithm that applies sparsity to achieve higher energy efficiency in both neuromorphic and non-neuromorphic settings. Finally, Chapter 10 concludes the thesis and outlines potential future research directions toward the next generation of sustainable, autonomous AI systems.

Chapter 1

Background on Neuromorphic Computing

Neuromorphic computing is an approach to designing computing systems that is inspired by the structure and functionality of biological neural networks. Contrary to conventional computing architectures, which operate through continuous signals and clock-driven instructions, neuromorphic systems emulate the brain's distributed and event-driven characteristics. These systems are built based on the concept of spiking neurons and dynamic synapses, in which information is transmitted through discrete electrical pulses, or spikes, like how neurons communicate in the brain. Learning in neuromorphic systems is typically governed by local and biological rules. These rules often allow the network to dynamically adapt based on the timing and correlation of neural activity. Neuromorphic computing represents a significant shift towards a more natural and efficient paradigm, offering advantages such as superior energy efficiency, real-time processing capabilities, and the ability to handle complex, dynamic real world-data. Its inherently parallel processing ability makes it ideal for edge AI, robotics, and other applications that require on-device intelligence without constant cloud reliance.

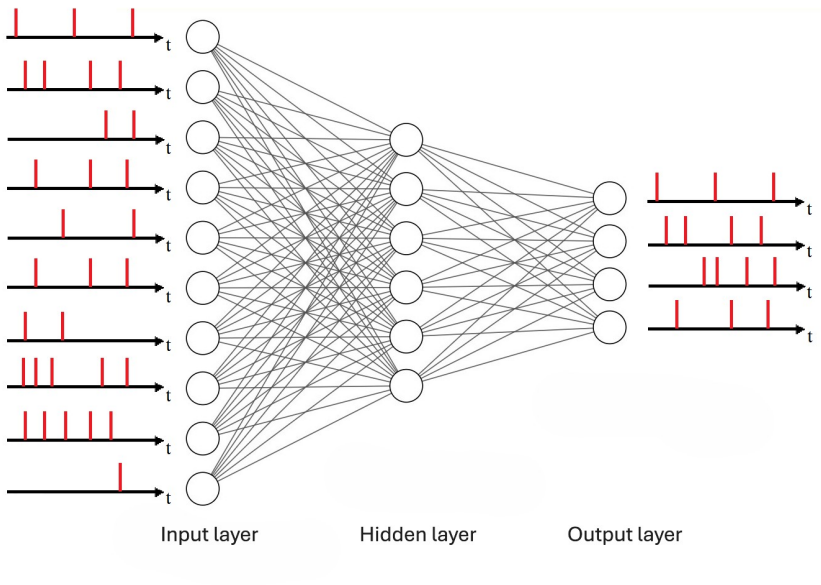


Figure 1.1: SNN Architecture. Source: [12].

1.1 Spiking Neural Networks

Spiking Neural Networks (SNNs) are considered as the third generation of neural network models, representing a transformative step in the realm of artificial intelligence [12]. they emerged from the need to develop more compact networks in terms of energy efficiency and memory footprint, while achieving performance comparable to traditional deep learning models. By deploying systems with less computational complexity and a smaller number of parameters, SNNs offer a promising pathway toward efficient, brain-inspired computing.

Fig. 1.1 depicts the architecture of a standard SNN, comprising an input layer, one or more hidden layers, and an output layer. Unlike traditional artificial neural networks, the input in an SNN is encoded as a series of discrete spikes rather than continuous values. These spike-based signals are propagated through the network using biologically inspired mechanisms. The final layer also generates spike trains or spike counts that can be interpreted for classification, decision-making, or further processing.

1.1.1 Definition and Biological Inspiration

SNNs are computational architectures inspired directly by the real functioning of biological nervous systems. Unlike traditional artificial neural networks, which process continuous values at fixed time steps, SNNs operate using discrete events known as spikes that occur asynchronously in time [13]. In biological neurons, a spike corresponds to an action potential, which is defined as a rapid and transient change in the neuron's membrane potential caused by the movement of ions across the cell membrane through voltage-gated channels. This brief electrical pulse is the fundamental unit of information transmission in both biological and artificial spiking networks and is central to energy-efficient computation.

Fig. 1.2 illustrates a typical neuronal action potential, showing how the membrane potential changes over time. At rest, the neuron has a negative membrane potential of approximately (-70 mV), where the intracellular space is more negative with respect to the extracellular. The membrane's selective permeability and the active transport of ions, mainly facilitated by the sodium-potassium pump, ensure the maintenance of this resting potential [14], [15].

During the depolarization phase (highlighted in red), when a stimulus arrives, it causes the opening of voltage-gated sodium Na^+ channels, allowing a rapid influx of sodium Na^+ ions into the neuron. Because sodium is more concentrated outside the cell, this movement inward the cell will cause the intracellular space to become more positive, consequently the membrane potential will rise upward, often reaching values above (0 mV) [16], [17].

Following this peak, the voltage-gated K^+ channels will open and Na^+ channels will close, starting the repolarization phase, when the K^+ ions, which are more concentrated inside the cell, begin to flow outside, restoring the negative charge inside the neuron and returning the membrane potential back to its resting state [14].

In some cases, the membrane might become more negative than resting values briefly, this phase is known as hyperpolarization, and it happens due to the con-

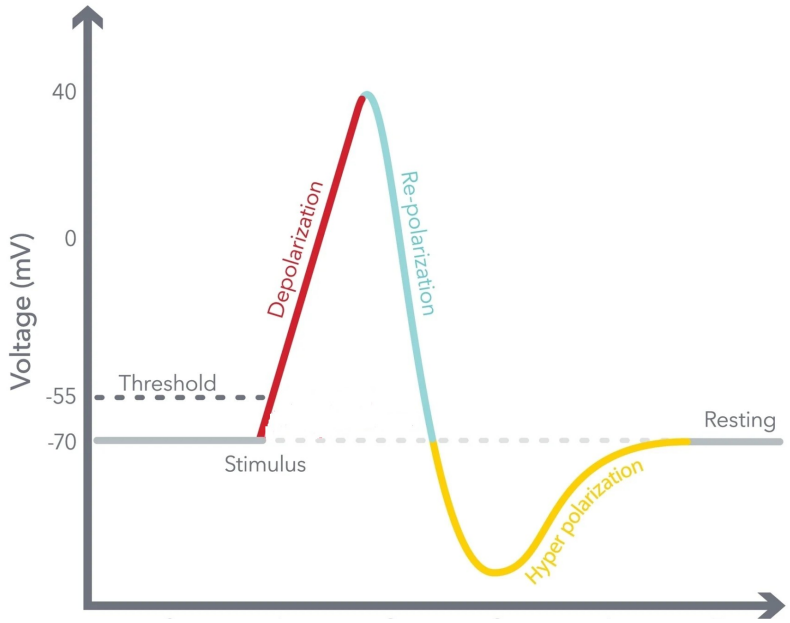


Figure 1.2: Illustration of an action potential. Source: *Simply Psychology* [18].

tinued movement of K^+ ions outside the cell before the channels close completely. The neuron then gradually returns to its resting potential as ion gradients are restored, ready for the next action potential cycle [15].

In SNNs, the membrane potential of each spiking neuron integrates incoming spikes, which represent the neuron's electrical state. In the absence of input, the membrane decays exponentially toward its resting value, reflecting the concept of natural leakage observed in biological neurons. When the membrane potential crosses a predetermined threshold, the neuron generates a spike that subsequently resetting its potential. This sequence of integration, threshold detection, spike generation, and reset mirrors the dynamic of real neurons, which the SNN models aim to capture and exploit for computation.

1.1.2 Neuron Models

Neurons are the fundamental building blocks of the nervous system and are responsible for receiving, processing, and transmitting information through electrical and chemical signals. A typical biological neuron basically consists of three

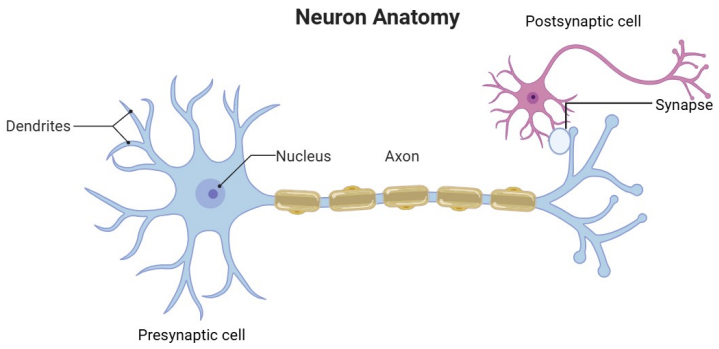


Figure 1.3: Neuron Anatomy.

main parts: dendrites, which receive incoming chemical or electrical signals from other neurons; the cell body (soma), which integrates these inputs and contains the nucleus; and the axon, which conducts electrical impulses to other neurons via specialized junctions called synapses. Communication between neurons occurs through discrete electrical pulses (action potentials), generated when the neuron's membrane potential crosses a certain threshold.

Fig. 1.3 illustrates the anatomy of a biological neuron, highlighting its key structural components and the direction of the flow of information between connected neurons. In this schematic, the presynaptic neuron is considered ad the source of the signal, the signal will travel through the axon terminals to the postsynaptic neuron, which receives the signal via its dendrites, forming the fundamental communication pathway in neural circuits.

In the following subsections, we describe three widely used mathematical models that capture different aspects of neuronal behavior: the Hodgkin–Huxley model, the Izhikevich model, and the Leaky Integrate-and-Fire model. Each model differs in its level of biological realism, computational complexity, and suitability for large-scale simulations.

Hodgkin–Huxley Model

The Hodgkin–Huxley (HH) model [19] is a detailed biophysical model that describes the ionic mechanisms underlying the initiation and propagation of action potentials in neurons. It represents the membrane as an electrical circuit, where ion-specific conductance changes dynamically according to voltage-gated channel kinetics. The membrane potential V evolves according to:

$$C_m \frac{dV}{dt} = -g_{\text{Na}} m^3 h (V - E_{\text{Na}}) - g_{\text{K}} n^4 (V - E_{\text{K}}) - g_L (V - E_L) + I_{\text{ext}} \tag{1.1}$$

where C_m is the membrane capacitance, $g_{\text{Na}}, g_{\text{K}}, g_L$ are maximal conductances for sodium, potassium, and leak channels, $E_{\text{Na}}, E_{\text{K}}, E_L$ are their respective reversal potentials, m, h, n are gating variables, and I_{ext} is the external input current. The HH model consists of coupled nonlinear ODEs that require small time steps for accurate simulation, particularly during spike initiation, making large-scale network simulations computationally demanding. Efficiency can be improved with implicit methods or adaptive solvers, but HH neurons remain costly compared to simplified models.

Izhikevich Model

The Izhikevich model [20] offers a balance between computational efficiency and biological plausibility. It can reproduce a wide range of spiking and bursting patterns with only two coupled differential equations:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{1.2}$$

$$\frac{du}{dt} = a(bv - u) \tag{1.3}$$

with the after-spike reset:

$$\text{if } v \geq v_{th} \text{ mV, then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{1.4}$$

Here, v represents the membrane potential, v_{th}) represents the threshold voltage u the recovery variable, and a, b, c, d are dimensionless parameters controlling spiking dynamics. The Izhikevich model is widely used when biologically realistic spiking patterns are needed at a low computational cost, providing a simplified, yet powerful, two-variable system capable of reproducing a wide range of cortical neuron firing behaviors, including regular spiking, bursting, fast spiking, and chattering, using just a few parameters. Its low-dimensional formulation makes it suitable for large-scale network simulations.

Leaky Integrate-and-Fire Model

The Leaky Integrate-and-Fire (LIF) model [13] is one of the simplest and most computationally efficient spiking neuron models. It describes the membrane potential V as a leaky integrator of the incoming current, governed by the following differential equation:

$$\tau_m \frac{dV_m}{dt} = -(V_m - V_{\text{rest}}) + R_m I(t) \tag{1.5}$$

where τ_m is the membrane time constant, V_{rest} the resting potential, R_m the membrane resistance, and $I(t)$ the input current. When V_m reaches the threshold V_{th}, the neuron emits a spike and V_m is reset to V_{reset}.

Fig. 1.4 shows the dynamics of the LIF model. The membrane potential V_m integrates the incoming spike signals coming from the presynaptic neuron, and when it crosses the threshold V_{th}, a spike will be generated at postsynaptic neuron. In the absence of incoming spikes, the membrane potential decays exponentially as a result of the leaky property of the model [22], [23].

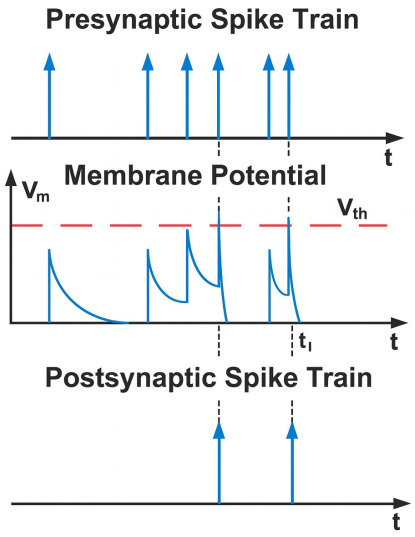


Figure 1.4: The dynamics of LIF Model. Source: [21].

The simplicity of the LIF model comes at the cost of omitting many biological details and complexities, including spike shape, refractory ion channel dynamics, and adaptive firing behavior. However, the model effectively captures the fundamental mechanism of threshold-based spiking and temporal integration. This simplicity makes it highly efficient for the implementation of both software and neuromorphic hardware. Furthermore, its analytical tractability makes it a valuable tool for theoretical investigations of neural coding, network dynamics, and synaptic plasticity [13].

1.1.3 Spike Encoding Schemes

SNNs process information in the form of discrete electrical pulses, or spikes, rather than continuous-valued activations. However, most real-world data, such as images, audio signals, and sensors, are not naturally represented as spike trains. Therefore, a crucial preprocessing step in SNNs is spike encoding, the process of converting input signals into temporally distributed spike events that can be

processed by spiking neurons. The way data are encoded into spikes has a direct impact on the computational efficiency, accuracy, and ability of the network to exploit temporal information. Encoding strategies can represent different aspects of the input signal, such as its amplitude, temporal profile, or spatial distribution. The choice of method depends on factors such as the nature of the data, the computational constraints of the system, and the desired balance between biological realism and engineering practicality. The following are the three primary spike encoding strategies, each briefly defined:

- **Rate Coding:** Represents information using the average firing rate of a neuron over a specific time window. Higher input values correspond to higher spike frequencies, while lower values produce fewer spikes.
- **Latency (Temporal) Coding:** Encodes information on the precise timing of spikes relative to a reference point (e.g., stimulus onset). The magnitude of the input is represented by the latency of the first spike; stronger stimuli produce earlier spikes.
- **Delta Modulation Coding:** Generates spikes based on changes in the input signal rather than its absolute value. A spike is produced when the signal increases (or decreases) by a predefined threshold, making it efficient for representing signals with few variations.

1.2 Astrocytes

In the nervous system, there exists a specific type of non-neuronal cells known as *glial cells*. The Key type of *glial cells* include *astrocytes*. For many years, astrocytes were thought to be a passive element in the brain, providing only structural and metabolic support, but increasing experimental evidence has shown that they actively contribute to information processing and synaptic regulation [9], [10]. Unlike neurons, astrocytes do not generate action potentials; instead,

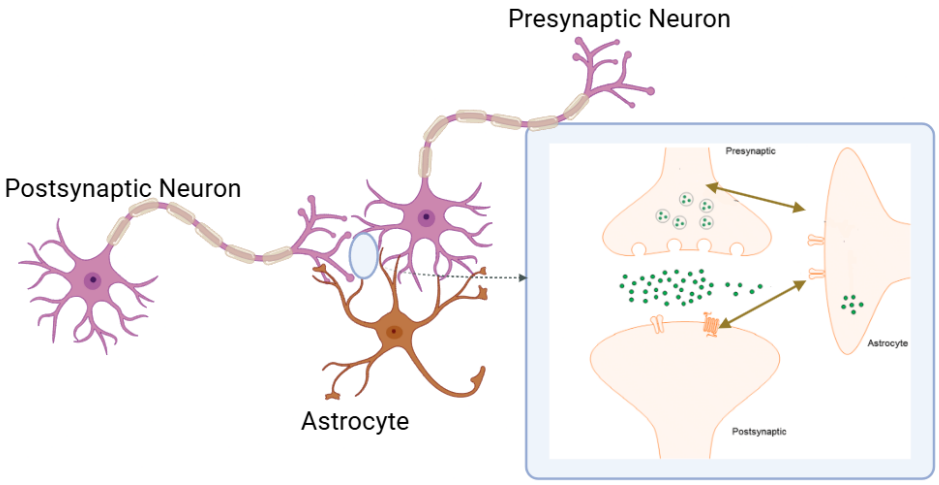


Figure 1.5: Astrocyte-Neuron Interaction.

they communicate through intracellular calcium (Ca^{2+}) signaling and the release of specialized chemical messengers known as gliotransmitters, allowing them to modulate synaptic efficacy and neuronal activity of nearby neurons and synapses.

Fig. 1.5 illustrates the interaction between two neurons and an astrocyte during synaptic communication. The presynaptic neuron transmits chemical signals (neurotransmitters) to the postsynaptic neuron through a small gap called synaptic cleft. Some of these neurotrnasmitters also bind to receptors in the astrocyte, leading to its activation. This activation initiates a series of intracellular events that involve the production of Ca^{2+} which in turn triggers the release of gliotransmitters, which can modulate the activity of pre- and postsynaptic neurons. This bidirectional feedback will help regulate and improve the interaction between neurons.

Bidirectional communication between astrocytes and neurons is often described as the *tripartite synapse*, enables the astrocytes in regulating different

function in synaptic homeostasis, Spike-Timing-Dependent Plasticity (STDP), and other learning mechanisms. Consequently, astrocytes introduce an additional layer of adaptive, modulatory control that extends beyond purely neuronal learning rules.

In the context of neuromorphic computing, astrocytes have recently attracted attention as potential modulators of synaptic plasticity, offering a biologically grounded mechanism for stability, self-regulation, and improved adaptability in SNNs. In a later chapter, we will explore how astrocyte-inspired models can be integrated with learning rules such as STDP and R-STDP to enhance the computational performance of artificial systems.

1.3 Advantages and Challenges of SNNs

SNNs offer several benefits over traditional artificial neural networks, but they also face important technical and practical limitations that must be addressed for broader adoption. In the following, the main advantages and challenges are summarized.

Advantages

- **Energy Efficiency:** Event-driven computation ensures neurons consume energy only when they fire, making SNNs ideal for low-power neuromorphic hardware such as Intel Loihi, IBM TrueNorth, and SpiNNaker.
- **Temporal Information Processing:** SNNs naturally handle spatio-temporal data streams (e.g., audio, video, event-camera signals) and can encode richer information through spike timing.
- **Biological Plausibility:** Closer to the behavior of real neurons, enabling applications in computational neuroscience and biologically inspired AI.

- **Sparse Communication:** Spikes are sparse events, reducing communication and memory usage in large-scale networks.
- **Compatibility with Neuromorphic Hardware:** Designed for asynchronous, event-driven architectures that offer low-latency processing.

Challenges

- **Training Complexity:** Traditional backpropagation is not directly applicable due to the non-differentiable nature of spikes, requiring alternative learning rules such as STDP or surrogate gradients.
- **Encoding Overhead:** Many real-world inputs are not spike-based, making efficient and effective encoding a critical challenge.
- **Limited Maturity of Tools and Frameworks:** Fewer established development environments compared to deep learning, and existing simulators (e.g., Brian2, NEST) can be slow for large-scale tasks.
- **Lack of Event-Based Datasets:** Most benchmark datasets are designed for frame-based models, complicating direct comparisons.
- **Hardware Availability:** Neuromorphic processors are still emerging technologies and are not widely accessible for mainstream deployment.

Chapter 2

Biological Learning Rules

2.1 Biologically Inspired Learning Rules

A key difference between SNN and conventional Artificial Neural Networks (ANNs) lies in their learning and weight-update mechanisms. While conventional networks depend on backpropagation, SNNs employ biologically inspired learning rules that replicate mechanisms observed in the brain. These rules regulate how synaptic weights update over time based on the correlation between pre- and post-synaptic neuronal activity, enabling online adaptation and learning through purely local interactions. In the following sections, we describe several prominent biologically inspired learning rules, including STDP, Reward-Modulated STDP (R-STDP), and notable variants such as triplet STDP and anti-STDP, highlighting their biological motivation, mathematical formulation, and computational properties.

2.1.1 Spike-Timing-Dependent Plasticity (STDP)

STDP is one of the most well-known biologically inspired learning rules in SNNs, directly modeled after experimental findings in neuroscience [5], [24]. It modifies synaptic weights according to the precise timing between pre-synaptic and post-synaptic spikes. If the pre-synaptic spike arrives shortly before the post-synaptic

spike, the connection is strengthened (long-term potentiation), while the opposite, when the presynaptic spikes arrives shortley after the postsynaptic spike, the connection is weakened (long-term depression). This rule reflects Hebb's principle of "cells that fire together, wire together" [25] but incorporates temporal precision on the scale of milliseconds.

Mathematically, the weight change Δw is often modeled as:

$$\Delta w = \begin{cases} A_+ \exp\left(-\frac{\Delta t}{\tau_+}\right), & \text{if } \Delta t < 0 \\ -A_- \exp\left(\frac{\Delta t}{\tau_-}\right), & \text{if } \Delta t > 0 \end{cases} \tag{2.1}$$

where $\Delta t = t_{\text{pre}} - t_{\text{post}}$ is the spike time difference, A_+ and A_- are the maximum potentiation and depression amplitudes, and τ_+ and τ_- are the respective time constants [26].

Beyond its sophisticated biological foundation, STDP provides powerful computational capabilities for learning from complex, time-varying input streams. This tool allows networks to naturally encode the temporal dependencies present in spiking input data. This temporal sensitivity enables SNNs to establish associations between casually related spikes, facilitating the learning of repeating spatiotemporal patterns without the need of labeled supervision. This makes STDP especially suitable for applications such as pattern recognition [27], event-based vision [28], and novelty detection [29], where input statistics are dynamic and sparse.

Fig. 2.1, illustrates the characteristic STDP learning window, where positive Δt results in synaptic depression and negative Δt leads to potentiation, with both effects decaying exponentially as the spike timing difference increases.

Furthermore, from a computational perspective, STDP functions as a form of Unsupervised Hebbian learning because it emphasizes statistical regularities in the input, therefore, leading to the creation of specialized neurons with selective receptive fields, which render STDP as an effective mechanism for feature extraction in the initial sensory layers.

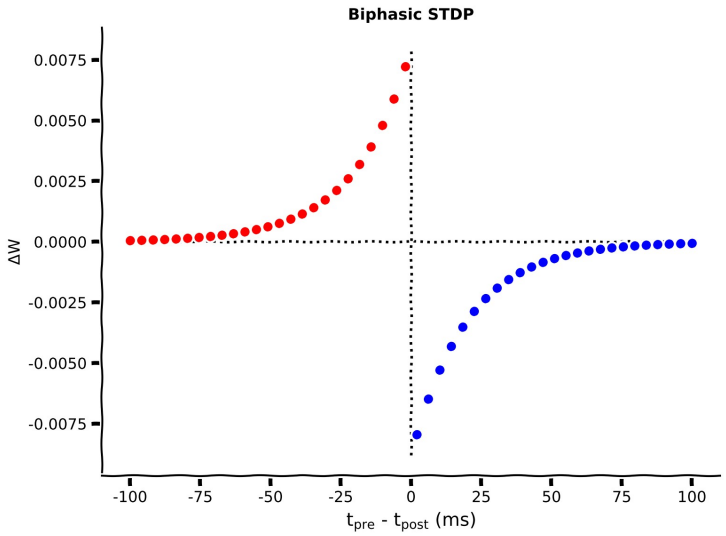


Figure 2.1: STDP Model. Source: [30].

2.1.2 Anti-Spike-Timing-Dependent Plasticity (Anti-STDP)

Anti-STDP is a variant of the standard STDP rule in which the sign of synaptic modification is reversed [31], [32]. In classical STDP, pre-synaptic spikes arriving before post-synaptic spikes lead to synaptic potentiation, and the opposite timing results in depression. In anti-STDP, this relationship is inverted: pre-before-post timing induces depression, while post-before-pre timing induces potentiation.

This learning rule has been observed in specific biological contexts, such as inhibitory synapses or particular sensory systems, and is believed to play a role in maintaining network stability, shaping receptive fields, or implementing predictive coding mechanisms.

Mathematically, anti-STDP can be formulated by simply reversing the sign of the standard STDP update:

$$\Delta w = \begin{cases} A_- \exp\left(-\frac{\Delta t}{\tau_+}\right), & \text{if } \Delta t < 0 \\ -A_+ \exp\left(\frac{\Delta t}{\tau_-}\right), & \text{if } \Delta t > 0 \end{cases} \tag{2.2}$$

where $\Delta t = t_{\text{pre}} - t_{\text{post}}$, and A_+, A_-, τ_+, and τ_- follow the same definitions as in classical STDP but produce the opposite plasticity effect.

Anti-STDP can be combined with standard STDP to deploy a balanced network architectures, where inhibitory neurons follow anti-STDP and excitatory neurons follow classical STDP. This balance is crucial in preventing runaway excitation and promoting stable self-organized network dynamics.

2.1.3 Reward-Modulated STDP (R-STDP)

Reward-Modulated STDP (R-STDP) extends the basic STDP rule by introducing a global reward or punishment signal that contributes to synaptic plasticity [6], [33]. This learning paradigm draws inspiration from dopaminergic signaling in the brain, where neuromodulators, particularly dopamine, are involved in regulating plastic changes in synapses, based on a feedback signal that indicates the success or failure of recent behavior. The core mechanism of R-STDP includes the application of synaptic updates in two separate steps: a local timing-based *eligibility trace* stores the relative timing of presynaptic and postsynaptic spikes, and a global scalar reward, which arrives later, determines whether these changes are consolidated or discarded.

A simplified formulation is:

$$\Delta w = \eta \cdot r(t) \cdot e(t) \tag{2.3}$$

where η is the learning rate, $r(t)$ is the reward signal and $e(t)$ is the eligibility trace calculated from spike timing. This allows the network to learn from sparse and delayed feedback rather than only from immediate spike correlations.

2.1.4 Triplet STDP / Other Variants.

Several variants of STDP have been proposed to address stability, improve performance, or model more complex biological processes:

- **Triplet STDP:** Extends the traditional pair-based STDP learning rule. While classical STDP considers only the relative timing between a single

pre-synaptic and post-synaptic spike (pairs), triplet STDP models interactions involving spike triplets—typically two pre-synaptic and one post-synaptic spike or vice versa. This enhancement allows the model to more accurately replicate frequency-dependent synaptic plasticity observed in biological experiments, which pair-based STDP fails to capture [7], [34], [35].

- **Homeostatic Plasticity:** a crucial neurobiological mechanism that acts to stabilize neural activity over time [36]. While Hebbian-like plasticity rules strengthen or weaken individual synapses based on activity, they can lead to runaway dynamics with some synapses growing excessively strong (leading to over excitation) and others vanishing entirely (leading to neuron silencing). To counteract this, homeostatic plasticity regulates synaptic strengths or neuronal excitability at a global scale, ensuring that the average firing rate of neurons remains within functional bounds. It serves as a regulatory process that keeps neural circuits balanced, preserving dynamic range and preventing saturation. [8], [37].

- **Voltage-Dependent STDP:** The voltage-based STDP rule has been introduces based on several experimental observations, in addition to explain the emergence of some connectivity patterns in the cerebral cortex [38]. In this model, potentiation and depression are two independent mechanisms, whose sum produces the total synaptic change. In particular, synaptic weight change depends not only on spike timing (as STDP), but also on the postsynaptic membrane voltage (V_{post}) at the time of presynaptic spike. For example, when a depression or potentiation event is triggered by presynaptic spike, the updates will be induced only if the (V_{post}) is above a predefined threshold (θ) [34].

Each of these variants retains the local, event-driven nature of STDP while modifying the update rule to suit specific computational goals or biological constraints.

Chapter 3

Astrocytes Inspired SNN

3.1 Intoduction and Motivation

Spiking Neural Networks have garnered increasing recognition for their energy efficiency, event-driven computation, and greater resemblance to biological neural processes. These characteristics make them a compelling choice for brain-inspired artificial intelligence and neuromorphic systems. However, despite their advantages, SNN still lag behind traditional artificial neural networks trained with BP and struggle in resolving complex tasks such as classification and pattern recognition. One key reason for this gap is that current SNN models, including those that employ biologically plausible learning rules, are only able to capture a subset of the intricate dynamics observed in the brain. These models are based exclusively on neuronal mechanisms, neglecting the critical contributions of other biological elements involved in neural processing.

In recent years, neuroscience has increasingly highlighted the active role of glila cells, particularly **astrocytes** [39], in modulating synaptic plasticity and information transmission. Astrocytes were once considered to be purely structural and metabolic support cells, but now are known to play a crucial role in synaptic regulation by influencing the strength and timing of neuronal communication through their dynamic calcium signaling and gliotransmitter release mechanisms

[40].

Motivated by these findings, we explore how the astrocyte-inspired components can be integrated into the artificial neural system to enhance learning stability, plasticity control, and overall network performance [41]. In particular, we examine whether the inclusion of astrocytic dynamics can provide a biologically grounded mechanism to regulate synaptic updates in a more efficient and adaptive manner.

In this chapter, we implement an SNN architecture that incorporates a simulated astrocyte unit into a reward-modulated STDP learning framework. The proposed model is evaluated on a supervised classification task to assess the potential benefits of astrocyte integration on learning performance.

3.2 Proposed Methodology

Numerous studies have shown that relying solely on STDP in SNN can be limiting. STDP enables neurons to act as coincidence detectors enabling them to respond selectively to recurring spike patterns that often stimulate the postsynaptic activity. While effective in capturing statistically frequent features from the input signal, STDP struggles to detect rare but highly informative patterns, particularly that are essential for decision making process.

In order to overcome these limitations, R-STDP [42], [43] has been proposed as a biologically inspired extension of traditional STDP, incorporating key principles from reinforcement learning (RL). In the brain, neuromodulators such as dopamine play a crucial role in guiding the decision-making and learning process, particularly in the presence of reward signals [44], [45]. These dopamine signals modulate synaptic plasticity by influencing both the magnitude and polarity of synaptic weight changes, and by affecting the temporal sensitivity of STDP updates [46], [47]. One research line utilizes R-STDP to train a SNN for visual classification task, relying on the timing of the first spike as decision-making cue

[43]. This strategy leverages the inherent temporal dynamics and sparse nature of spiking activity to achieve efficient and accurate classification. However, these models generally depend exclusively on dopaminergic modulation and do not take into account the various neuromodulatory dynamics or cellular interactions that exist in biological systems and may further enhance learning robustness and flexibility.

Given the biological plausibility and adaptive potential of R-STDP, in this work, we propose to further enhance it by integrating astrocyte-inspired modulation. Therefore, we will investigate whether the integration of astrocytes can enhance the performance and adaptability of SNNs. The resulting model, designated as Astro-Reward-SNN is evaluated in a supervised classification scenario to explore its feasibility and benefits in comparison to traditional approach.

3.2.1 Tripartite Synapse

The term **Tripartite synapse** refers to the functional and physical integration of presynaptic, postsynaptic neurons and astrcoytes [48]. Fig. 3.1 illustrates a classical tripartite synapse, demonstrating the existence of bidirectional communication between astrocytes and neurons. In summary, the presynaptic neuron encodes information in the form of an action potential (AP), which is then transimtted to the synaptic cleft via neurotransmitters known as glutamate Glu^{-}, a proportion of these molecules will bind to the astrocytic Glu^{-} receptors, while the remainder will bind to postsynaptic Glu^{-} receptors located on posynaptic neuron [49], thereby affecting its action potential. The activation of the astrocytic receptor will trigger a signaling cascade within the astrocyte, which consequently results in the production of a molecule called IP_3 (inositol triphosphate). IP_3 then stimulates the release of calcium ions (Ca^{2+}) from the internal stores of the astrocyte. When the (Ca^{2+}) concentration in the astrocyte cross a certain threshold [50], the astrocyte releases its own chemical signal known as gliotransmitters into the extracellular space. These gliotransmitter can then bind to the postsynaptic

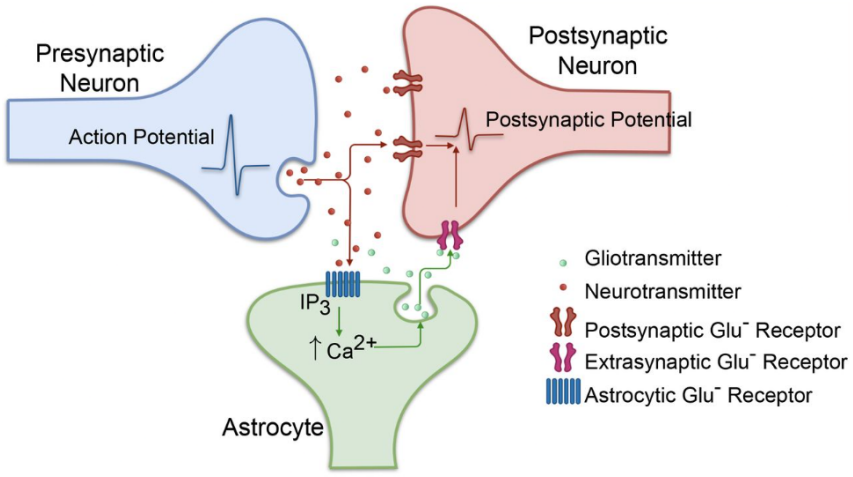


Figure 3.1: The Tripartite synapse.

Glu^{-} receptors, modulating its excitability and responsiveness to future inputs. Recent studies reveal that a single astrocyte can interact with a large number of synapses from multiple pre- and postsynaptic neurons.

3.2.2 Astrocyte Model

The Astrocyte dynamics in this work is described based on Li-Rinzel model [51], a simplified but powerful mathematical model that describes the (Ca^{2+}) fluctuation in response to IP_3 signaling.

The core dynamics of calcium changes based on Li–Rinzel model is described using the following two differential equations.

1- Calcium concentration in the cytosol, given as:

$$\frac{dc}{dt} = J_{\text{chan}} + J_{\text{leak}} - J_{\text{pump}} \tag{3.1}$$

where J_{chan}, J_{leak} and J_{pump} denote channel current, ER leak current and

SERCA pump current respectively, calculated using following equations:

$$J_{\text{chan}}(c,h,I) = v_1\, m_\infty(I)^3\, h^3\, \big(c_{\text{ER}} - c\big), \tag{3.2}$$

$$J_{\text{leak}}(c) = v_2\, \big(c_{\text{ER}} - c\big), \tag{3.3}$$

$$J_{\text{pump}}(c) = v_3\, \frac{c^2}{k_3^2 + c^2}. \tag{3.4}$$

where v_1 represents IP_3 channel rate, v_2 passive ER leak rate, and v_3 pump rate.

And fast activation gate of IP3 m_∞:

$$m_\infty(I) = \frac{I}{I + d_1}.$$

2- Fraction of open IP_3 receptor subunits (a gating variable):

$$\frac{dh}{dt} = \frac{h_\infty - h}{\tau_h}. \tag{3.5}$$

where

$$h_\infty = \frac{Q_2}{Q_2 + c}, \qquad \tau_h = \frac{1}{a_2\,\big(Q_2 + c\big)}.$$

The typical set of parameters used in Li-Rinzel model simulations is summarized in table 3.1, as commonly adopted in the literature.

3.2.3 Astro-Reward-SNN Architecture

The proposed Astro-SNN architecture integrates astrocyte-inspired modulation into a reward-based SNN to enhance learning and classification performance on the MNIST dataset. The complete pipeline consists of three key stages: data preprocessing, spike-based neural network, and output classification as depicted in in Fig. 3.2. The network is composed of three primary layers:

Table 3.1: Li-Rinzel Parameter set.

Parameter name	Unit	Value
v_1	$s(^{-1})$	6
v_2	$s^{-1})$	0.11
v_3	(μM s^{-1})	0.9
k_3	(μM)	0.1
c_0	(μM)	2.0
c_1	(–)	0.185
d_1	(μM)	0.13
d_2	(μM)	1.049
d_3	(μM)	0.9434
a_2	($\mu M^{-1} s^{-1}$)	0.2

- **Input Layer:** Including 784 Poisson neurons, each corresponding to a pixel in the input image.
- **Astrocyte Layer:** This layer includes 10 Astrocyte units, inspired by biological astrocytes that are implicated in the tripartite synapse. Each astrocyte receives input from all 784 input neurons. The calcium dynamics are modeled using Li-Rinzel model, modulating synaptic plasticity based on incoming spike activity. Each astrocyte has one-to-one connection with the corresponding neuron in the output layer.
- **Output Layer:** Contains 10 neurons, one for each digit class, these neurons receive both direct input from the input layer and the modulated signal from astrcoytes.

3.2.4 Learning Rule Mechanism: Astro-Reward-SNN

In the proposed model, we extend the classical R-STDP framework by integrating the astrocyte unit, thereby enhancing the biological plausibility and the overall performance of the network. Astrocytes are known to interact with neurons via calcium (Ca^{2+}) signaling and gliotransmitters release, modulating synaptic activity in response to neuronal input. In this study, we leverage this mechanism by

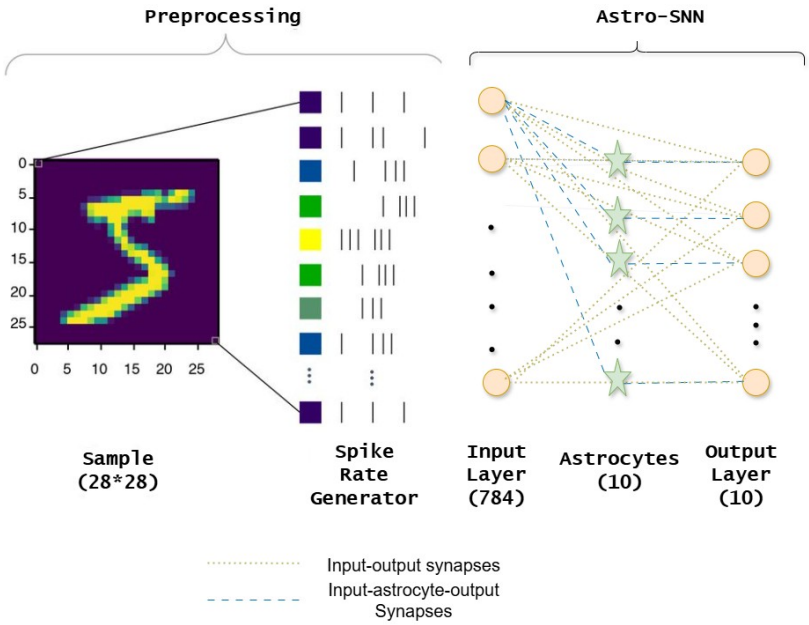


Figure 3.2: Astro-SNN Network Architecture.

allowing the astrocytic (Ca^{2+}) concentration to modulate the strength of synaptic update at the moment a reward signal is delivered. In particular, when the network correctly classifies an input, a positive dopamine reward is released, reinforcing the synaptic connections involved in the decision. The calcium level in the associated astrocyte acts as modulation signal influencing the magnitude of synaptic weight update. Conversely, when the network produces an incorrect classification, a negative dopamine signal is released, serving as a punishment to depress the synaptic weights associated with the error. Additionally, when the astrocytic calcium level exceeds a predefined threshold, the astrocyte triggers a **slow inward current (SIC)** toward the postsynaptic neuron through the tripartite synapse. This modulatory signal enhances postsynaptic excitability and can improve overall synaptic efficacy. The synaptic plasticity dynamics are defined according to the following update rule:

Positive Dopamine (Reward) – Long-Term Potentiation (LTP):

$$\frac{dw}{dt} = \frac{\eta}{\tau_w} \cdot (\mathrm{Ca}_{2+} \cdot dop \cdot e) \cdot (w_{\max} - w) \tag{3.6}$$

Negative Dopamine (Punishment) – Long-Term Depression (LTD):

$$\frac{dw}{dt} = \frac{\eta}{\tau_w} \cdot (\mathrm{Ca}_{2+} \cdot dop \cdot e) \cdot (w_{\min} - w) \tag{3.7}$$

where η is the learning rate, τ_w is the synaptic weight time constant, $w_{\min}$ and $w_{\max}$ are the minimum and maximum weight bounds, e is the correlation trace between pre- and post-synaptic spikes, Ca_{2+} is the astrocytic calcium level that gates synaptic plasticity.

3.3 Experiments and Results

A series of experiments was conducted to evaluate the effectiveness of the proposed Astro-Reward-SNN model. Our goal was to estimate the impact of astrocyte-driven modulation on network performance by comparing it with a baseline model consisting of a standard SNN trained with R-STDP in a classification task using MNIST dataset. The architecture used in both models was the same, with the only difference being the integration of astrocyte units in the proposed model.

Both models were trained using only a subset of the MNIST dataset, consisting of 3000 samples evenly distributed across all 10 digit classes. Each model was trained over 10 epochs. The input images were encoded into spike trains using Poisson encoding based on pixel intensities.

Fig. 3.3 shows the training and validation accuracy curves for both models over 10 training epochs. As illustrated the **baseline R-STDP SNN** achieved a maximum validation accuracy of **80%**, whereas the **astrocyte-integrated model** consistently outperformed it, reaching a peak validation accuracy of **86%**

Fig. 3.4 shows the distribution of the training and validation accuracy for the

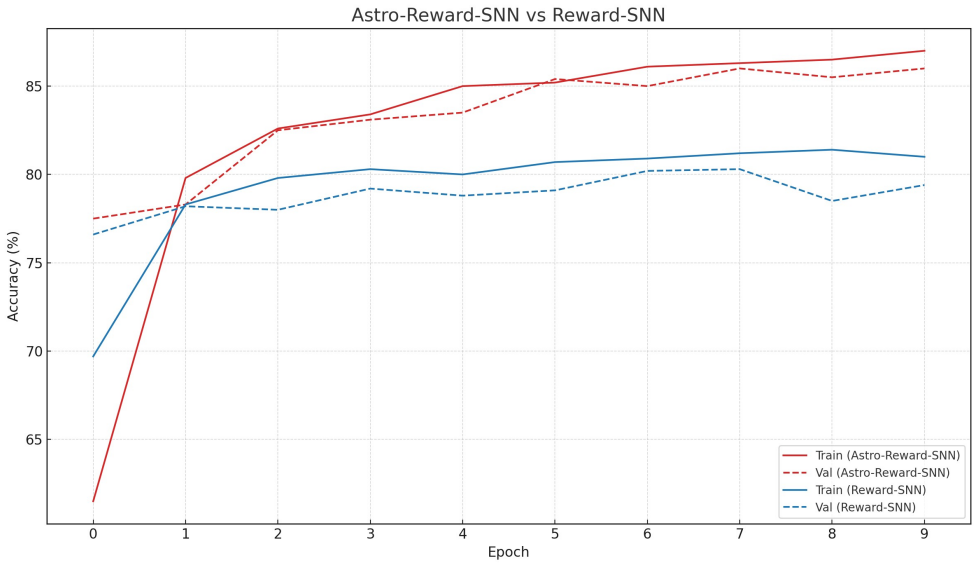


Figure 3.3: Training and validation accuracy across epochs for Reward-SNN and Astro-Reward-SNN.

Astro-Reward-SNN and Reward-SNN models. The former demonstrates higher median accuracy and a broader performance range compared to the latter, with both training and validation accuracies consistently outperforming the baseline model, highlighting the potential benefit of integrating astrocyte-inspired modulation in learning.

The results clearly demonstrate the benefits of integrating astrocytic modulation within the learning process. Astrocytes, due to their slower dynamics which can span several seconds, astrocytes serve as temporal integrators, adding a memory effect for recent neuronal activity and its correlation with reward signals. Combined with dopamine signaling, this facilitates reward-delayed synaptic plasticity, mimicking what is observed in biological reinforcement learning. Moreover, the learning sensitivity is significantly enhanced, as synaptic updates only become potent when both the dopamine reward signal and astrocytic calcium levels (Ca^{2+}) are sufficiently high. This dual-modulation mechanism ensures that

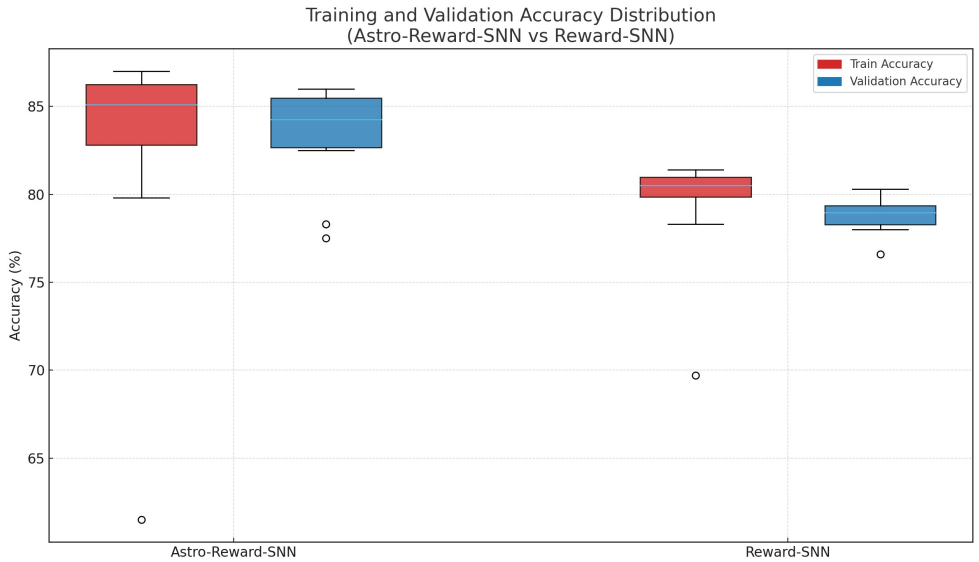


Figure 3.4: Distribution of accuracy values for both models across multiple runs.

synaptic changes occur in response to both neural activity and contextual relevance, leading to more stable, selective, and efficient learning outcomes.

Chapter 4

Biological Strategies for Anomaly Detection

4.1 Deep learning Models for Anomaly Detection

Anomaly detection systems play a key role in a variety of domains. Anomalies refer to rare but high-impact events, such as cyberattacks in computer networks, unexpected faults in industrial machinery, or irregularities in healthcare monitoring systems. Therefore, detecting such events should be quick and reliable to prevent catastrophic failures, financial loss, and even risks to human safety. However, anomaly detection remains a challenging task because discriminative features are often subtle and embedded within complex temporal or spatial correlations that are difficult to model.

To address these challenges, researchers have increasingly focused on deep learning techniques. Several architectures have become widely used for anomaly detection, each offering specific strengths:

- **Convolutional Neural Networks (CNNs):** Originally developed for image-related tasks, CNNs are powerful feature extractors capable of cap-

turing spatial and frequency-based patterns. In anomaly detection, they have been successfully applied to intrusion detection and malware traffic classification in cybersecurity [52], as well as bearing fault detection in industrial monitoring, where they learn discriminative spatial-frequency features from vibration signals [53]. More recently, CNN-based models have also been adapted for time-series anomaly detection, inspired by image segmentation architectures such as U-Net, which was originally designed for biomedical image segmentation. In this adaption,the U-Net structure is transformed to 1D data (time-series), treating anomaly detection as a segmentation problem; instead of identifying objects in image, the network detects anomalous temporal windows within the continuous signal. This approach highlights the flexibility of CNN architectures in handling structured temporal data beyond their original image-processing domain.

- **Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks:** RNNs are designed to model sequential dependencies, making them a natural fit for time-series anomaly detection. LSTM networks, in particular, address the problem of vanishing gradient and effectively capture long-term temporal dependencies. They have been applied to predictive maintenance of rotating machinery, where sequential vibration patterns record early mechanical failures [54], as well as in healthcare for ECG anomaly detection, where they accurately identify arrhythmias from long sequences of heartbeats [55]. Although RNNs have been the predominant choice for time-series tasks, their high computational demands have encouraged the exploration of alternative models such as CNNs and hybrid approaches.

- **Autoencoders (AEs):** Autoencoders are among the most widely used architectures for anomaly detection. they have been applied in divers domains, including predictive maintenance [56], intrusion detection [57], and computer vision anomaly detection. Both an unsupervised and semi-

supervised setting were explored. The principle behind AE-based Anomaly detection is to reconstruct the input data, Anomalies are detected when the reconstruction error exceeds a predefined threshold, indicating that input pattern deviates from the normal data distribution learned during training

- **Generative Adversarial Networks (GANs):** Generative models that learn the distribution of normal data and identify outliers as deviations from it. GANs have been applied in biomedical anomaly detection, such as EEG-based seizure detection, where adversarial training helps to capture rare spatiotemporal patterns [58].

In general, deep learning models have demonstrated remarkable success in anomaly detection, achieving state-of-the-art performance in many domains through automatic feature extraction and representation learning. Nevertheless, these approaches are often limited by high computational demands, dependence on large labeled datasets, and challenges in real-time or resource-constrained deployments.

4.2 Motivation for using SNNs in Anomaly Detection

As discussed in the previous sections, deep learning architectures such as CNNs, LSTMs, autoencoders, and GANs have achieved strong results in anomaly detection. However, their deployment remains limited by high computational and memory requirements, dependence on large labeled datasets, and challenges in real-time operation on embedded or edge devices. These limitations are particularly critical for anomaly detection, where data are often continuous, anomalies are rare by definition, and systems are expected to operate in an always-on, energy-efficient manner.

Spiking Neural Networks (SNNs) provide a promising alternative to this scenario. Their event-driven computation ensures that neurons are active only when

relevant spikes occur, offering significant energy savings compared to clock-driven deep networks. Moreover, the temporal dynamics inherent in spike-based coding makes SNNs especially suited for anomaly detection in streaming data such as vibration signals, biomedical monitoring, or network traffic, where irregularities manifest as deviations in timing patterns.

Finally, biologically inspired learning mechanisms such as STDP and its variants enable local, online adaptation without the need for costly backpropagation.

4.3 Proposed SNN Model for Anomaly Detection

In biological neural systems, the responsible unit of communication between neurons is the *synapse*, the junction that connects one neuron to another, allowing the transmission of signals. Synapses are broadly classified into two categories depending on their effect on the postsynaptic neuron: *excitatory* and *inhibitory*. Excitatory synapses increase the likelihood of postsynaptic firing by depolarizing the membrane potential, whereas inhibitory synapses reduce this likelihood by hyperpolarizing the postsynaptic membrane. Translating this principle into SNNs, excitatory synapses contribute positively to the membrane potential, while inhibitory synapses contribute negatively. In this context, an essential property of neural systems is homeostatic plasticity [59], a fundamental mechanism that maintains stable activity levels among neurons despite continuous changes in external inputs or stimuli. This process acts as self-regulating system that prevents neuron from becoming overactive or completely silent. Inspired by this principle, we developed a *Balanced Spiking Neural Network (BSNN)* in which excitatory and inhibitory activities are dynamically adjusted through a combination of STDP and anti-STDP learning rules [60]. This design enables the network to adapt to a normal operating condition by adjusting its synaptic weights such that the output neurons remain "quiet" (i.e., no spikes are emitted) indicating normal

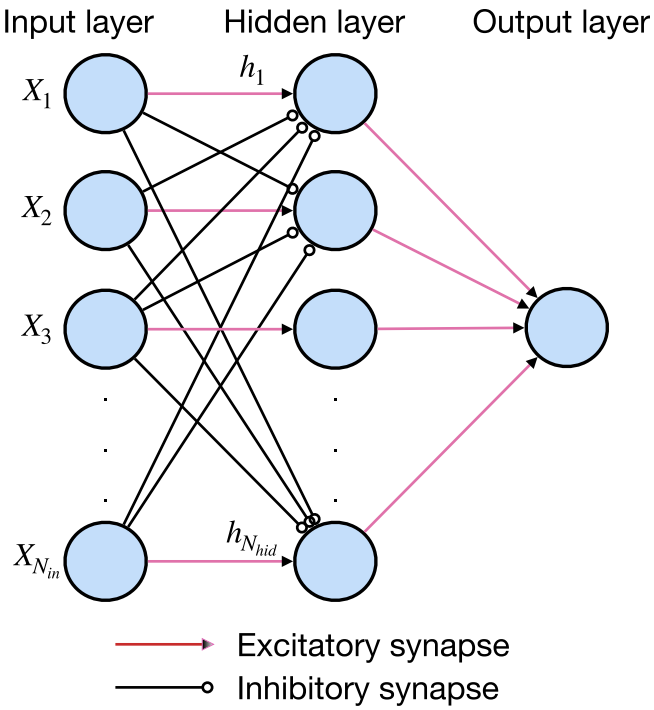


Figure 4.1: Balanced Spiking Neural Network Architecture.

circumstances. When the adapted network encounters input patterns that deviate significantly from the learned normal dynamics, this equilibrium is disrupted, causing the emission of spikes at the output layer. Since anomalies are rare events by definition, the system remains energy-efficient, being active only when abnormal conditions are detected.

Fig. 4.1 shows a typical BSNN architecture consisting of three layers: an input layer, a hidden layer, and an output layer. To achieve balance, the input and hidden layers contain the same number of neurons. Each hidden neuron is connected to one input neuron through a single excitatory synapse, while also receiving inhibitory synapses from all remaining input neurons. In this way, the balance is achieved by carefully setting these synaptic weights so that the total excitatory and inhibitory contributions are equal in magnitude, yet opposite in effect. This configuration ensures that the network maintains an overall balance between excitatory and inhibitory activities.

4.4 Methodology and Data Processing

Although STDP is inspired by biological learning and is widely used for SNN training, it is not sufficient to guarantee homeostasis state. Therefore, we explicitly enforce balance at the architectural level by initializing the synaptic weights as follows:

$$w_{exc} = w, \qquad w_{inh} = \frac{w}{N_{in} - 1}, \tag{4.1}$$

where w_{exc} and w_{inh} represent the excitatory and inhibitory synaptic weights, w is the base weight, and N_{in} is the number of input neurons. Similarly, the output synaptic weights are initialized as

$$w_{out} = \frac{K}{N_{hid}}, \tag{4.2}$$

where K is a constant proportional to the neuron threshold voltage V_{th}, and N_{hid} is the number of hidden neurons. This configuration provides an initial balanced starting point for learning. From this baseline, the BSNN can then adapt dynamically through plasticity rules.

Although this initialization and the BSNN design itself provide a well-structured starting point, feeding the network with real-world data(such as vibration signals encoded into spikes) does not guarantee stable behavior. In practice, noisy or irregular input can disrupt the initial equilibrium, leading to unstable activity. To address this, we propose the joint use of STDP and anti-STDP to continuously regulate synaptic dynamics and guide the network toward a homeostatic state. The principle is as follows:

- STDP is applied to inhibitory synapses, strengthening or weakening them depending on pre- and post-synaptic spike correlations following Eq. 2.1.

- Anti-STDP is applied to excitatory synapses, decreasing their strength in the presence of correlated firing and increasing it when correlations are

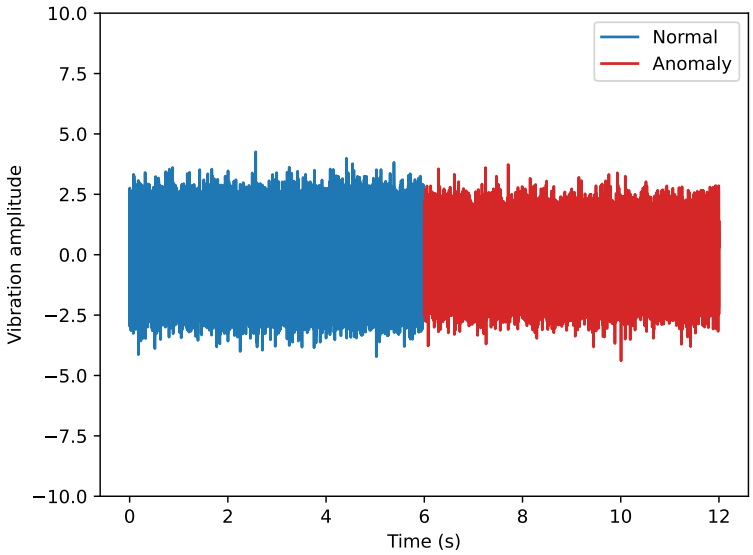


Figure 4.2: Example of Normal vs. Faulty vibration signal from the MFPT dataset.

inverse following Eq. 2.2

This complementary mechanism ensures that the BSNN maintains a low firing rate under normal conditions while producing strong activity in response to anomalous patterns.

The dataset used to evaluate the proposed model is provided by the Machinery Failure Prevention Technology (MFPT) society, consists of vibration signals sampled at 97.656kHz, recorded from bearing test rigs under different operating conditions. The dataset includes both normal conditions and several fault scenarios, such as inner race, outer race, and ball defects, collected under varying speeds and loads. This makes the MFPT dataset a representative benchmark for machinery health monitoring and anomaly detection, since it captures realistic operating variability and fault patterns that are often subtle and difficult to separate from background noise. Fig. 4.2 shows a concatenated sample of normal and defective vibration signals.

To prepare the raw vibration data for processing by the BSNN, we applied a two-step preprocessing pipeline. First, the vibration signals were passed through

a gammatone filterbank composed of 16 filters with center frequencies f_c logarithmically spaced between 40 Hz and 48,828 Hz. This step performs a form of feature extraction by decomposing the broadband signal into multiple frequency channels, each representing a distinct component of the original vibration pattern.

In the second step, the filtered signals were converted into spike trains, ensuring compatibility with the event-driven nature of SNNs. To achieve this, we adopted the *asynchronous delta modulation* process (ADM), a conventional method also used in analog-to-digital processing. In this scheme, a spike is generated whenever the signal crosses a predefined threshold, after which the crossed threshold is subtracted from the signal. This mechanism is implemented through an additive bias voltage V_{dc}, which resets the signal level every time a spike occurs. As a result, the spike train encodes the *fluctuations* of the input amplitude rather than its absolute value, providing a temporal representation suitable for SNNs.

Since vibration signals exhibit substantial amplitude variations across channels, selecting an appropriate threshold was critical. To address this, we set the threshold of each channel adaptively, defined as the product of that channel's variance and a scaling parameter. This normalization ensures consistency in the spike encoding process across all frequency channels.

After this preprocessing phase, the data were transformed into spike trains and were ready to be fed into the BSNN for training and inference. In the next step, the network undergoes an adaptive learning procedure during training to achieve a homeostatic balance between excitatory and inhibitory activities.

The algorithm. 1 summarizes the proposed learning procedure. It takes three inputs: the spike train data X, a spike tolerance threshold θ, and the BSNN architecture. Its output is the adapted network model. During training, the algorithm iterates over the input data and applies two learning rules separately: STDP to the inhibitory synapses and anti-STDP to the excitatory synapses. After each update, the output activity of the network is monitored using the

SpikeMonitor function. If the number of spikes falls below the tolerance threshold θ, the algorithm terminates, indicating that the network has reached the desired homeostatic state.

This iterative procedure ensures that the BSNN reaches a stable homeostatic regime before being tested on unseen data

Algorithm 1: Proposed learning algorithm.

```
Input:
  X: Input spike train
  θ: Spike tolerance threshold
  BSNN: Network architecture
Output:
  Adapted BSNN model
while True:
  STDP(BSNN.S_inh, X)
  Anti-STDP(BSNN.S_exc, X)
  s ← SpikeMonitor(BSNN)
  if s < θ:
    break
  end
end
```

4.5 Experiments and Results

4.5.1 BSNN Architecture and Parameters

The proposed BSNN was implemented using the same baseline architecture described in [61]. It consists of three layers: an input layer with $N_{in} = 16$ neurons, a hidden layer with $N_{hid} = 16$ neurons, and a single output neuron ($N_{out} = 1$). This configuration results in 16 excitatory synapses (S_{exc}) and 240 inhibitory synapses (S_{inh}) connecting the input and hidden layers, while 16 output synapses (S_{out}) project from the hidden to the output neuron. A summary of the full network configuration, including the parameters of neuron and synapses, is provided in Table 4.1. It should be noted that several neuron-related parameters, such as the

resting potential V_{rest}, the firing threshold V_{th}, and the membrane time constant τ_m, were derived from values reported in neuroscience literature to maintain biological plausibility. In contrast, parameters such as synaptic weights (w_{exc}, w_{inh}, w_{out}) and the spike tolerance threshold θ were empirically tuned to ensure stable training and effective anomaly detection performance.

Table 4.1: BSNN Parameters.

Network Parameters		
Parameter name	Value	Notes
N_{in}	16	# of input neurons
N_{hid}	16	# of hidden neurons
N_{out}	1	# of output neurons
S_{exc}	16	# of excitatory synapses, input to hidden
S_{inh}	240	# of inhibitory synapses, input to hidden
S_{out}	16	# of excitatory synapses, hidden to output
τ_m	140 ms	Membrane potential decay constant
τ	20 ms	STDP and Anti-STDP decay constant
V_{rest}	-70.6 mV	Neuron resting potential
V_{th}	-40.6 mV	Neuron threshold voltage
w_{exc}	3	Excitatory synaptic weight, input to hidden
w_{inh}	0.2	Inhibitory synaptic weight, input to hidden
w_{out}	2.52	Output synaptic weight
θ	20	Spike tolerance threshold

4.5.2 Training Phase

Following the preprocessing pipeline described earlier, the BSNN was first trained to adapt to normal operating conditions and achieve a homeostatic state. Figure 4.3 shows the evolution of the output neuron membrane potential during training. Since a single normal vibration signal was too short to achieve convergence, the signal was concatenated with itself to obtain a 12-second sequence. Over time, the network progressively reduced its spike activity, eventually stabilizing between 6 and 12 s. In general, we can observe that as time increases, the BSNN produces fewer output spikes. Moreover, after 2.87 s, the output membrane potential exhibits a decreasing trend that stabilizes from 6 to 12 s as a

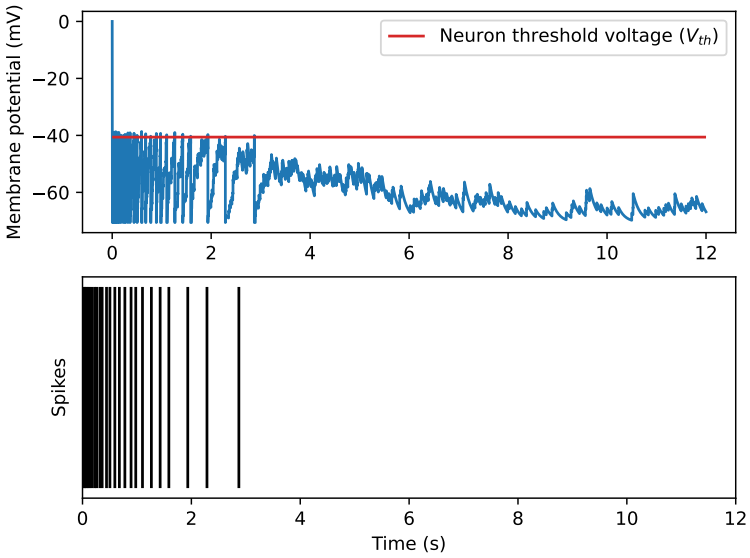


Figure 4.3: BSNN adaptation using the proposed learning algorithm.

consequence of the compensation between excitatory and inhibitory activities, thus demonstrating the effectiveness of our approach.

4.5.3 Inference Phase

Once adapted, the BSNN was tested on both normal and anomalous vibration signals after preprocessing with the gammatone filterbank and asynchronous delta modulation. Fig. 4.4 depicts the inference phase. During the first half of the experiment (normal condition), the network remained silent, maintaining the output membrane potential around -58 mV. At 6 seconds, the signal transitioned to an anomalous condition, which disrupted the balance and caused the membrane potential to rise above the threshold V_{th}. This led to the emission of spikes, marking the detection of the anomaly. The delay between the onset of the anomaly and the first spike was approximately 0.21 seconds, due to the time required to load the membrane potential, and is consistent with results reported in [61].

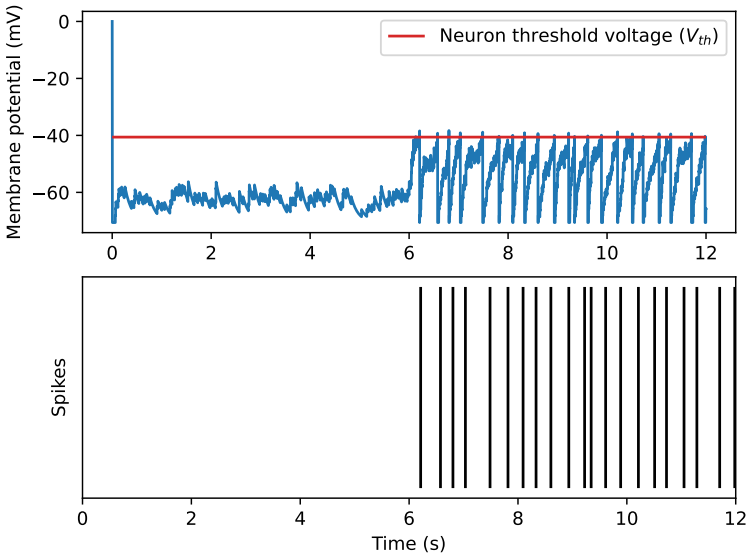


Figure 4.4: Adapted BSNN inference test.

4.5.4 Real-Time simulation

To further assess the robustness of the proposed system, we designed a second inference experiment aimed at emulating a more realistic deployment scenario. In practice, signals evolve continuously in real time, and the occurrence of an anomaly is not known a priori. Under these conditions, the preprocessing approach used in the first test, where normal and anomalous signals were fully processed and concatenated in advance, would not be feasible.

To mimic this real-world setting, we constructed new test sequences by concatenating a random-length slice of the normal vibration signal with an anomalous segment, as illustrated in Fig. 4.2, Unlike the previous experiment, where the entire input was preprocessed at once, here the raw signal was divided into multiple windows. Each window was independently converted into a spike train and sequentially analyzed by the BSNN.

This sliding-window approach reflects better the way the system would operate online, processing data as they become available. However, it also introduces a key design factor, which should be carefully selected which is the window size. A large window may improve accuracy by capturing more context, but it also

increases the detection delay. Conversely, smaller windows reduce latency but risk producing false positives due to limited information. For this reason, we systematically varied the window size to evaluate the trade-off between detection delay and predictive accuracy, in order to identify the most effective configuration.

Fig. 4.5 illustrates how the detection delay introduced by our system is computed. The overall process is divided into three parallel stages: signal sampling, window preprocessing, and BSNN inference. In our test scenario, the exact point where the anomalous segment begins (after concatenation with the normal signal) is known and denoted as t_a.

To calculate the total detection time, we first define t_w as the time required to acquire a signal window and t_p as the time needed to preprocess this window and convert it into a spike train. At the start of the experiment, the BSNN must therefore wait a shift time $t_{shift} = t_w + t_p$ before inference can begin. Once a spike train is available, the BSNN processes it for an inference duration $t_e = t_w$. To emulate realistic deployment on a neuromorphic device, where inference runs for the same duration as the input window, we set the Brian2 simulation time to t_e.

Because sampling and preprocessing are performed in parallel with inference, new data become available at the end of each window without additional delay. Eventually, one of the windows will contain the anomalous portion of the signal. At this point, the homeostatic balance of the BSNN is disrupted and the network begins to emit output spikes. Let t_s represent the time elapsed from the beginning of inference until the first spike occurs. The overall detection delay can then be expressed as

$$T_{detection} = t_w + t_p + t_s - t_a \tag{4.3}$$

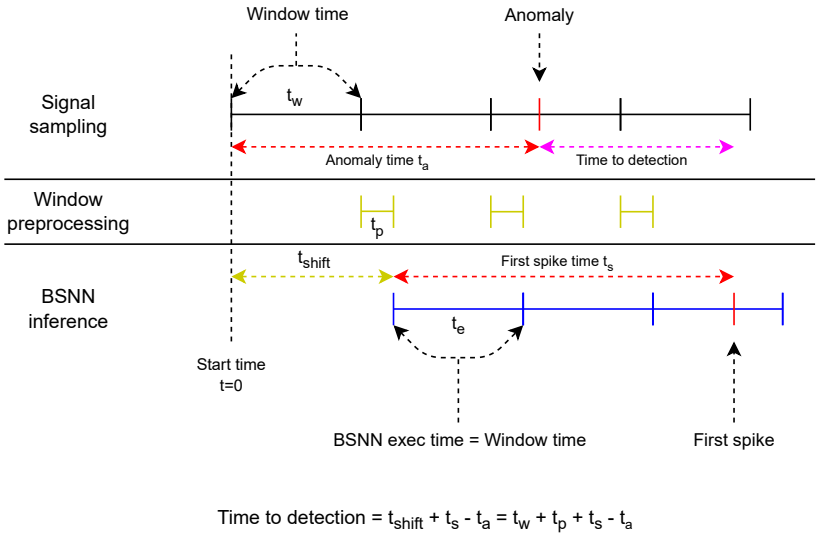


Figure 4.5: Time to detection diagram.

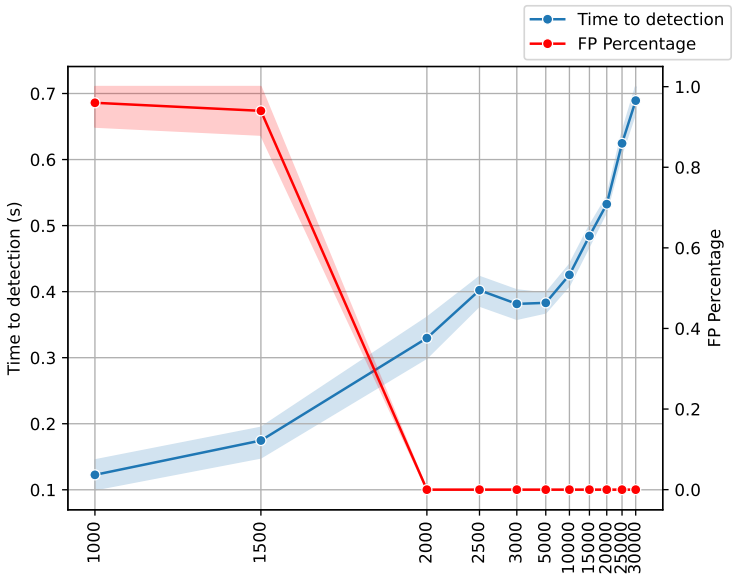


Figure 4.6: Adapted BSNN inference test.

4.5.5 Performance Evaluation

Fig. 4.6 shows the trade-off between the time to detection and the false positive (FP) rate as a function of the size of the window. Each configuration was tested in more than 50 independent experiments with randomly selected anomaly onset times. As expected, smaller window sizes reduced the detection delay, with values dropping from 0.69 s at 30,000 samples to 0.32 s at 2,000 samples. However, excessively small windows (1,000–1,500 samples) degraded performance, producing false positives in 94–96% of the cases. This was due to insufficient spike activity after delta modulation, which led the BSNN to incorrectly interpret imbalances as anomalies.

From these results, a window size of 2,000 samples was identified as the best trade-off, achieving fast detection while maintaining a low FP rate. This demonstrates the ability of the BSNN to operate reliably under realistic conditions while preserving energy efficiency.

4.6 Stop Learning Mechanism

4.6.1 Proposed Modulated BSNN (MBSNN)

An intrinsic mechanism to modulate learning and enables automatic switching from training mode to inference mode is important, especially in online learning context, where continuous exposure to data can lead to instability and performance degradation. In this section, we applied a biologically inspired "stop-learning" mechanism on the previously implemented BSNN using a global signal related to the network performance.

Although the BSNN model proved to be effective in learning normal input patterns, it exhibited several limitations that affect its long-term performance. First, the absence of method that govern the ongoing synaptic updates which lead to overfitting in case of extended training, gradually silencing the network even in

the presence of anomalous inputs. Second, the model suffered from catastrophic forgetting, where newly introduced patterns interfered with or replaced previously learned information, thereby compromising the network's memory retention and stability over time.

To address these challenges, we extend the original BSNN architecture by integrating a global modulatory signal [60].

Specifically, a modulatory learning signal from the output neuron will adjust the learning rate of excitatory and inhibitory synapses between the input and hidden layers. This biologically inspired feedback ensures that learning slows as the network converges on stable representations, preventing further weight updates that could lead to overfitting. This regulated learning approach not only preserves homeostatic balance, but also enhances the ability of the network to perform robust and continuous anomaly detection in real time within dynamic environments.

Fig. 4.7 illustrates the extended BSNN architecture, which preserves the original three-layers structure including an input layer, a hidden layer, and a single neuron output layer. The key modification introduced in this work is a set of modulatory synapses projects from the output neuron to all excitatory and inhibitory synapses between the input and hidden layers mimicking biologically inspired feedback mechanisms. Although in practice these modulatory connections influence every synapse in the input–hidden pathway, only a single representative link is shown in the figure to maintain visual clarity. These modulatory synapses dynamically regulate synaptic plasticity in conjunction with the STDP learning rule, enabling online and adaptive learning based on the output activity of the network.

In this sense, when output activity is high, the learning rate increases, allowing synaptic weights to change more significantly and promoting faster convergence. Conversely, during low output activity, plasticity is suppressed, resulting in minimal weight updates. This mechanism enables the network to adapt efficiently to

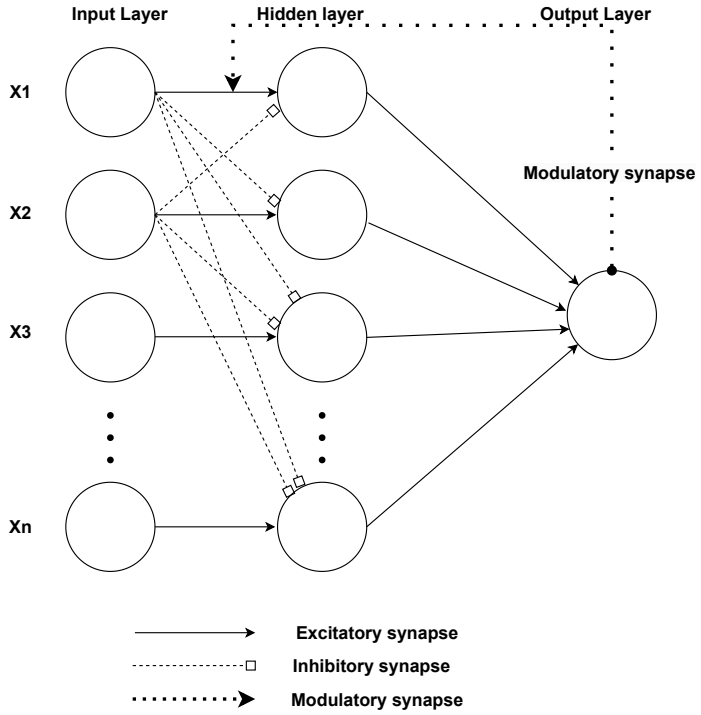


Figure 4.7: Balanced Modulated Spiking Neural Network Architecture.

new patterns while maintaining stability during periods of familiar or redundant input. An important feature of this learning rule is that it operates strictly online, in line with the biological learning observed in the human brain. To support this, we introduce two interacting modulatory variables, both driven by the output neuron's activity and designed to decay exponentially over time:

1. **Short-Term Modulation (`lr_mod_inc`):**

   The variable lr_mod_inc serves as a short-term modulation signal that is directly related to the output neuron activity. It is incremented by 1 each time an output spike occurs and evolves over time according to:

$$\frac{d\,\mathrm{lr_mod_inc}}{dt} = \frac{1 - \mathrm{lr_mod_inc}}{\tau_{\mathrm{lr_mod_inc}}} \tag{4.4}$$

2. **Long-Term Modulation (`lr_mod`):** This variable serves as a long-term modulation signal that accumulates recent output activity, effectively acting

as a memory trace over time. It is applied during synaptic updates between the input and hidden layers, scaling the weight changes based on prior output responses according to Eq. 2.1 . This modulation signal decays exponentially in the absence of new stimulation, following the differential equation:

$$\frac{d\,\mathrm{lr_mod}}{dt} = -\frac{\mathrm{lr_mod}}{\tau_{\mathrm{lr_mod}}} \tag{4.5}$$

The dynamics between these two modulatory signals is described by the following differential equations.

$$\mathrm{lr_mod} \leftarrow \mathrm{lr_mod} + \mathrm{lr_mod_inc} \tag{4.6}$$

This coupled mechanism ensures that the modulation signal is strong during high activity and progressively weakens during inactivity, providing a stable and adaptive learning process.

4.6.2 Results and Discussion

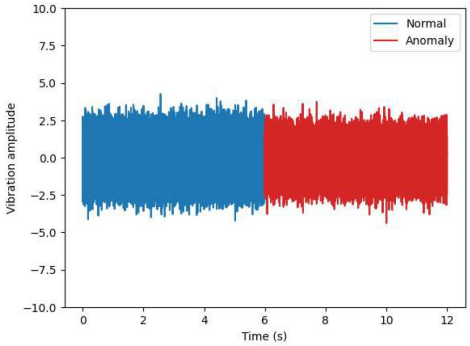
The results of anomaly detection experiments performed using the MBSNN model trained with the proposed regulation mechanism. All simulations were carried out using Brian2, as in BSNN model [62]. The same dataset was used to ensure consistency in the results and to enable a fair comparison of performance across models. In particular, we used the MFPT dataset.

The aim of our experiments is to compare the anomaly detection performance of the baseline BSNN against the enhanced version integrated with the proposed modulation mechanism. Specifically, the results highlight the system's ability to overcome overfitting, adapt to new patterns, and retain previously learned information, ensuring consistent and accurate behavior over time.

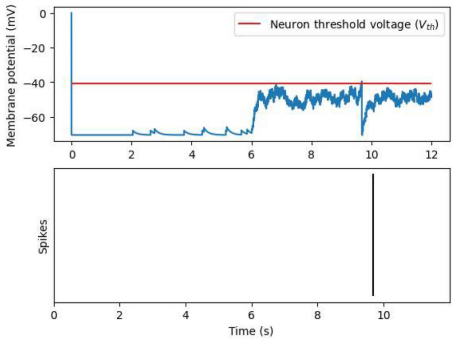
Regarding network architecture, we adopted the same configuration used in BSNN [60], the LIF model was used to govern the dynamics of all neurons as well.

In the first experiment, we trained both MBSNN and BSNN on the same dataset including only normal samples for an extended period to allow the network to adapt to normal conditions and be sensitive to anomalous states. During training, MBSNN progressively reduced its learning rate, thereby decreasing synaptic plasticity after convergence. As a result, it effectively mitigated overfitting and preserved the ability of the network to generalize. These results are illustrated in Fig. 4.8, which shows the spiking output of both networks during inference on a concatenated dataset containing both normal and anomalous samples. Specifically, Fig. 4.8b shows the inference results of BSNN, which maintains a consistently low firing rate across both types of input, reflecting its tendency to overfit. In contrast, Fig. 4.8d, shows that MBSNN maintains the same low baseline activity on normal data but shows a clear spiking activity when encountering anomalies. In other words, MBSNN improves the sensitivity of the network to anomalous events, preserving its performance on normal events.

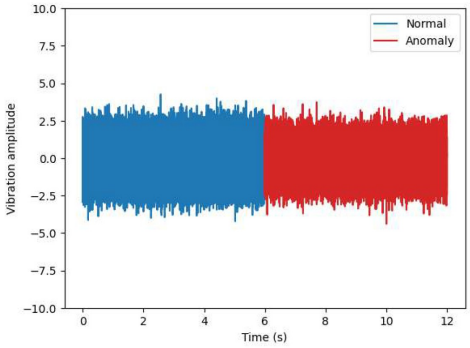
The second experiment was conducted to evaluate the effectiveness of the proposed technique in mitigating the catastrophic forgetting problem. BSNN tends to forget previously learned patterns and shows a reduced performance in anomaly detection when trained on a new task. To assess this, both models were first trained on a normal dataset (Task 1) until convergence. Subsequently, they were trained on a new task using a new normal sample(Task 2). The results presented in Fig. 4.9 highlight the performance of both models under these conditions. As shown in, Fig. 4.9b, BSNN's performance significantly degrades when tested on a mixed dataset of normal and anomalous samples, failing to distinguish between the two categories. In contrast, Fig 4.9d demonstrates that MBSNN maintains its ability to correctly identify both normal and anomalous inputs, illustrating its robustness and resistance to catastrophic forgetting.



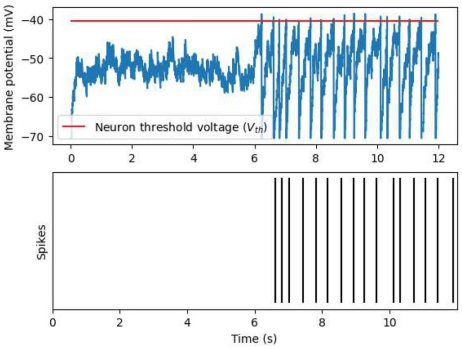
(a) Mixed data (normal + anomaly).



(b) BSNN inference test.

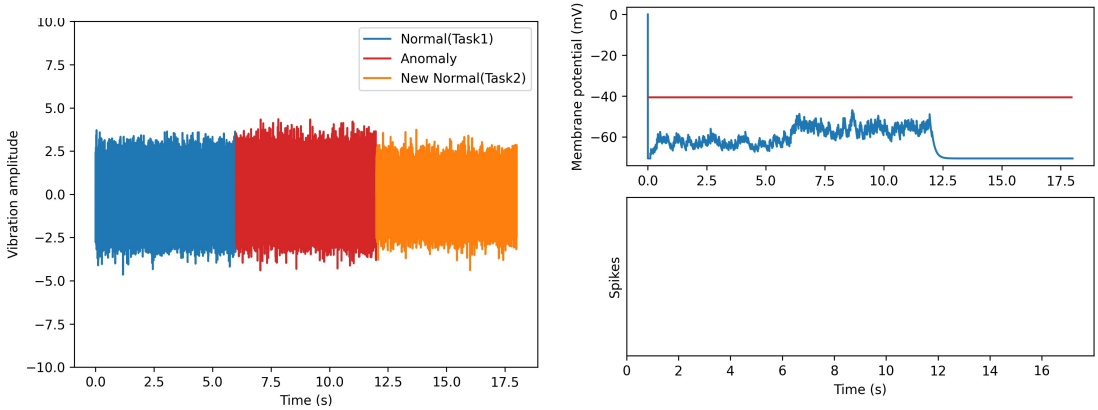


(c) Mixed data (normal + anomaly).



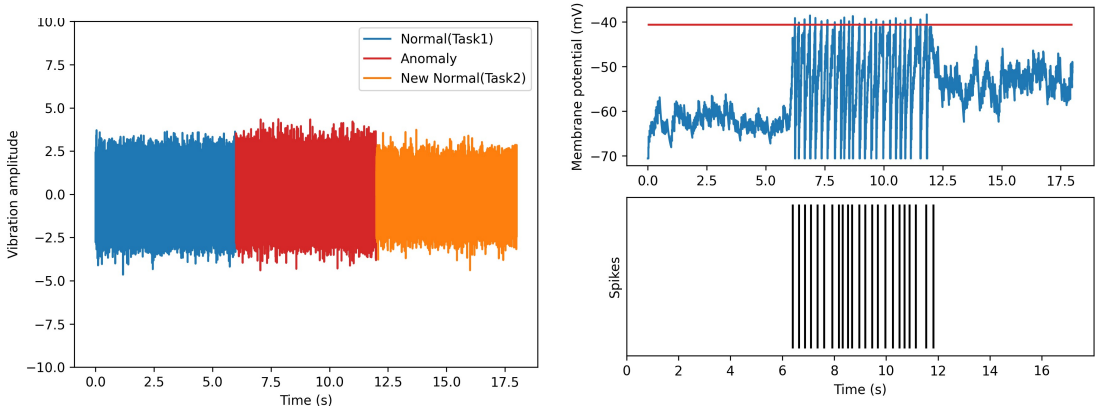
(d) MBSNN inference test.

Figure 4.8: Overfitting experiment showing inference on a concatenated stream of normal and anomalous samples after prolonged training on normal data, comparing BSNN and MBSNN to highlight overfitting resistance.



(a) Test data.

(b) BSNN inference test.



(c) Test data.

(d) MBSNN inference test.

Figure 4.9: Catastrophic forgetting experiment on three sequential sample sets. The MBSNN inference results demonstrate the network's ability to learn new tasks while retaining knowledge from previous tasks.

4.7 Summary

In this chapter, we introduced a biologically inspired anomaly detection framework based on a Balanced Spiking Neural Network (BSNN). The model leverages the homeostatic properties of neural systems by regulating excitatory and inhibitory synapses through a novel combination of STDP and anti-STDP. This self-regulating mechanism allows the network to remain in a low-activity state under normal conditions and become selectively responsive when deviations or anomalies occur, enabling efficient and interpretable anomaly detection in time series data.

Our findings, based on the analysis of vibration signals from the MFPT dataset, indicate that the BSNN can maintain stable homeostasis state during the training phase and reliably detect anomalies during the inference phase. Real-time emulation tests revealed a favorable trade-off between detection delay and false positives, with optimal performance observed at a window size of 2,000 samples. However, despite its strengths, the original BSNN model exhibited two major limitations: the risk of overfitting due to continuous synaptic updates and catastrophic forgetting when exposed to new input patterns, both of which compromise long-term performance and stability.

To address these issues, we proposed an extended version of the model, termed as Modulated BSNN (MBSNN). This enhanced architecture introduces a biologically inspired modulatory feedback signal that dynamically scales synaptic plasticity according to the activity of the output neuron. By doing so, MBSNN can switch from learning to inference once convergence is detected, preventing unnecessary synaptic updates and thus mitigating overfitting. Moreover, the modulation mechanism preserves previously learned patterns, effectively reducing catastrophic forgetting and improving the model's capacity for continuous learning.

In particular, the MBSNN achieves improved performance without requiring ongoing learning or external supervision during inference, further enhancing its

energy efficiency and suitability for deployment in low-power, online applications. In general, the contributions presented in this chapter highlight the promise of biologically plausible self-regulating SNNs for sustainable anomaly detection.

Chapter 5

Biological Strategies in Spiking Autoencoders (SAEs)

5.1 Introduction of Spiking Autoencoders

Autoencoders (AE) are a class of neural networks designed to learn efficient representations of data in an unsupervised manner. Initially popularized for their ability to perform dimensionality reduction [63] they have since become a versatile tool with applications ranging from denoising corrupted data [64], to generative modeling with probabilistic extensions such as Variational Autoencoders (VAEs) [65], and feature extraction for downstream tasks. Their ability to learn compact latent representations makes them a fundamental building block in many modern deep learning architectures. Beyond their utility in machine learning, AEs have also been linked to neuroscience, as their encoder–decoder structure mirrors how the brain compresses and retrieves sensory information [66].

Building on this analogy, Spiking Autoencoders (SAEs) extend the autoencoder paradigm into the spiking domain. They preserve the classical encoder–decoder structure but process information through discrete spike trains rather than continuous activations, thereby inheriting the temporal coding and energy efficiency benefits of Spiking Neural Networks (SNNs). Recent studies have investigated

SAE training with backpropagation through time and surrogate gradients, where temporal coding is used for image reconstruction [67]. Although these approaches achieve performance comparable to that of Artificial Neural Networks (ANNs), they come at the cost of significant memory consumption due to gradient storage, reliance on global error signals, and offline batch-based training. To overcome these limitations, alternative biologically inspired strategies have been explored. In particular, mirrored STDP has been proposed as a plausible mechanism for autoencoder-like learning in SNNs, offering a fully local and online adaptation process autoencoder learning in SNNs [68]. However, despite their biological grounding, purely STDP-based approaches often produce unstable weight distributions and performance that lag behind conventional ANN autoencoders if not carefully controlled. For example, the work in [69] proposed an efficient sparse SAE using error-modulated STDP combined with a Time-to-First-Spike (TTFS) encoding scheme for the image reconstruction task. Although this approach reduces computational overhead and promotes sparsity, it struggles to represent complex temporal patterns reliably.

In summary, spiking autoencoders represent a promising direction for merging data compression and reconstruction paradigms provided by conventional AEs with the temporal precision and low power benefits of neuromorphic computing. Providing biologically inspired alternative architectures, in different applications such as classification and anomaly detection.

5.2 Proposed methodology

In the context of Spiking Autoencoders (SAE) learning, inspired by the main object of AE; reconstructing the input based on the loss function, we proposed a learning rule that is able to reconstruct the input spike trains by modulating the synaptic weights of the network using online biologically plausible techniques [70]. These techniques exploit the local synaptic traces and modulatory signals offered

by biological SNNs to achieve efficient weight adaptation without the need for global error propagation. A key feature of our approach is the use of tied weights, where the encoder weights are defined as the transpose of the decoder weights.

This design drastically reduces the number of trainable parameters, optimizing the memory footprint, and simplifies the training process, making it suitable for real-time learning and deployment on neuromorphic hardware.

The primary objective of the proposed learning rule is to align the firing rates of the input and output layers. To accomplish this, the output layer must approximate the activity patterns of the input, allowing any mismatch between the two to serve as an error signal. This error is then used to govern the adaptation of synaptic weights and to improve the representation of the input in the output layer. To enable this mechanism, we introduce *modulatory connections* that connect the input layer to the code–output synapses, providing local access to input activity. Biologically, such modulatory pathways resemble neuromodulation processes, where neurotransmitters like dopamine regulate the direction and strength of synaptic plasticity. [71]

5.2.1 SAE Achitecture

The proposed Spiking Autoencoder (SAE) follows the classical encoder–decoder paradigm but operates entirely in the spiking domain. As shown in Fig. 5.1, the architecture consists of:

- An input layer with n spiking neurons,
- A latent code layer with m neurons ($m < n$), and
- An output layer with n neurons.

All neurons are modeled as leaky integrate-and-fire (LIF) units, chosen for their balance between biological plausibility and computational efficiency.

A key feature of the model is the use of *tied weights*, where the encoder weights are defined as the transpose of the decoder weights ($W_E = W_D^\top$). This design

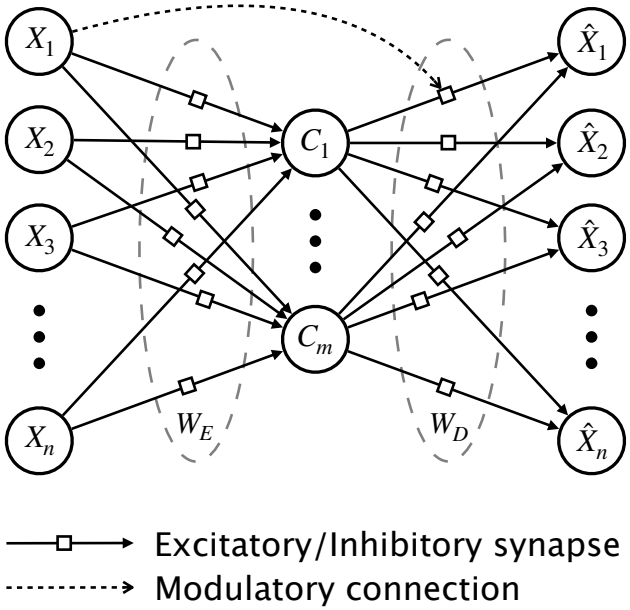


Figure 5.1: Proposed Spiking-AE architecture.

introduces a natural symmetry between the encoder and the decoder, similar to bidirectional cortical projections in the brain.

Several experiments have been carried out to determine the most effective network configuration. Based on these results, we adopted an *undercomplete topology*, where the code layer has a lower dimensionality than the input. This forces the network to extract only the most meaningful features, reducing the magnitude of synaptic connections and resulting in a more lightweight model. The latent code layer therefore provides a compressed spike-based representation of the input, enabling both reconstruction and downstream tasks such as classification and anomaly detection.

Input data are converted into spike trains using Poisson rate coding , ensuring compatibility with the event-driven dynamics of the network. The reconstruction process itself is driven by the proposed local learning rule, which is described in Section 5.2.2.

5.2.2 Learning Rule

The proposed learning rule enables *strictly online weight adaptation*, relying only on local information at each synapse rather than global error backpropagation. This design makes it suitable for real-time neuromorphic implementations.

To model neuronal activity, we use *synaptic traces* as an approximation of the instantaneous firing rate. For a given synapse, let A_{in}, A_{code}, and A_{out} denote the input, code, and output neuron traces, respectively. These traces increase whenever a spike occurs and decay exponentially over time. The temporal evolution of the synaptic traces is defined as:

$$\frac{dA_{in}}{dt} = -\frac{A_{in}}{\tau_{in}} + \sum_k A_{in}^+ \, \delta(t - t_{pre}^k), \tag{5.1}$$

$$\frac{dA_{code}}{dt} = -\frac{A_{code}}{\tau_{code}} + \sum_k A_{code}^+ \, \delta(t - t_{pre}^k), \tag{5.2}$$

$$\frac{dA_{out}}{dt} = -\frac{A_{out}}{\tau_{out}} + \sum_k A_{out}^+ \, \delta(t - t_{post}^k), \tag{5.3}$$

where τ_{in}, τ_{code}, and τ_{out} are decay constants, and the A^+ values denote the increments added at each spike event, each trace is like a memory of spikes that continuously approximate the firing rates without explicit spike-by-spike matching.

The weight dynamics is defined as:

$$\frac{dw}{dt} = \frac{\alpha}{\tau_w} A_{code} \cdot (A_{in} - A_{out}), \tag{5.4}$$

where w is the synaptic weight, α the learning rate, and τ_w the weight decay constant.

We can interpret the term

$$e = (A_{in} - A_{out}) \tag{5.5}$$

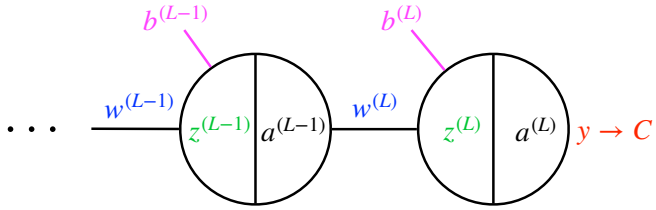


Figure 5.2: ANN network made of a single chain of neurons.

as a *local error signal*. If A_{out} is lower than A_{in}, the input activity is under-reconstructed, and the weights are increased to improve the representation. Conversely, if A_{out} exceeds A_{in}, the weights are weakened to suppress overestimation.

Finally, A_{code} acts as a multiplicative factor that scales the error-driven update, ensuring that only active code neurons contribute to synaptic adaptation. This mechanism makes weight changes both activity-dependent and locally regulated.

Eqs. (5.1)–(5.4) define an online plasticity mechanism in which weight updates depend only on spike-driven local activity, avoiding the need for storing global gradients or batch-based computations.

It is interesting to note that the weight update defined in Eq. (5.4) is similar to the gradient computation when using MSE as a loss function. For simplicity, without loss of generality, let us consider a simple network made of a single "chain" of neurons as shown in Fig. 5.2.

If we consider the MSE as the cost function C, then we can define the main equations describing the network as follows:

$$C = (a^{(L)} - y)^2, \tag{5.6}$$

$$z^{(L)} = w^{(L)} \cdot a^{(L-1)} + b^{(L)}, \tag{5.7}$$

$$a^{(L)} = \sigma(z^{(L)}), \tag{5.8}$$

where $a^{(L)}$ is the neuron activation, y is the network desired output, $z^{(L)}$ is the neuron summation, $w^{(L)}$ is the weight of the connection between the L^{th} and L^{th-1} neuron, $a^{(L-1)}$ is the activation of the L^{th-1} neuron, $b^{(L)}$ is the bias and σ is the activation function. Using the traditional chain rule, we can compute the gradient with respect to C as:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}}. \tag{5.9}$$

Hence, if we re-write eq. (5.9) expressing all the terms, we obtain:

$$\frac{\partial C}{\partial w^{(L)}} = 2(a^{(L)} - y) \cdot \sigma'(z^{(L)}) \cdot a^{(L-1)}, \tag{5.10}$$

Where the dependence of the gradient is evident on the difference between the network output and the desired one, and on the activity of the previous layer $a^{(L-1)}$ as already described in eq. (5.4).

5.2.3 Application Scenarios: Reconstruction and Anomaly Detection

Fig. 5.3 illustrates the complete architecture of the proposed Spiking-AE, which is structured into two main functional blocks: a *reconstruction block* and an *anomaly detection block*.

The reconstruction block corresponds to the core Spiking-AE described earlier, whose task is to compress the input spike trains into a latent code and reconstruct them at the output. This mechanism enables the network to learn compact spike-based representations that retain the most prominent features of the input, supporting downstream tasks such as classification or signal analysis.

The anomaly detection block extends the architecture with two layers: a comparison layer composed of $2n$ neurons (i.e., twice the number of neurons in the reconstruction output layer) and a single output neuron. Each pair of comparison neurons receives both the original input and its reconstructed counterpart,

with one neuron connected via excitatory synapses and the other via inhibitory synapses. This balanced arrangement is crucial: When the reconstruction closely matches the input, excitatory and inhibitory influences cancel out, resulting in stable activity and no spikes at the output. Conversely, any mismatch between input and reconstruction disrupts this balance, driving the output neuron to fire and signaling the presence of an anomaly. Since the anomaly detection block relies on static weights (W_B and W_O), it remains lightweight and modular, and can be attached to a trained reconstruction block without additional training.

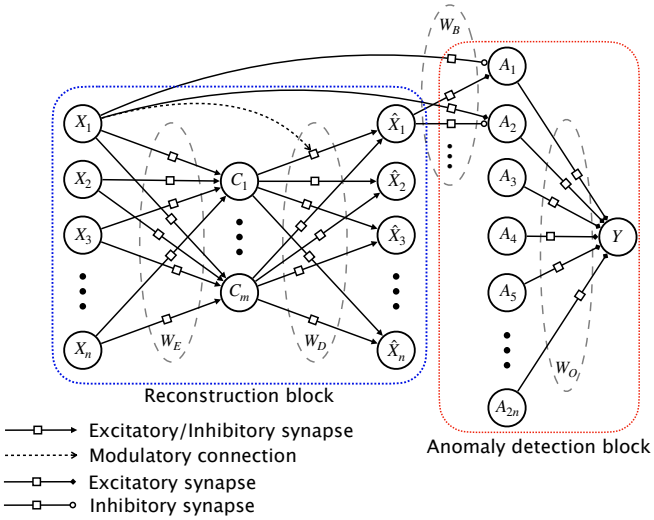


Figure 5.3: Proposed Anomaly Detection Spiking-AE architecture.

5.3 Results and Discussion

All experiments were conducted using the Brian2 simulator [62]. We report two main case studies to demonstrate the performance of the proposed learning rule and Spiking-AE architecture: reconstruction of handwritten digits (MNIST dataset) and anomaly detection in vibration signals (MFPT dataset).

5.3.1 Reconstruction Results (MNIST)

To assess reconstruction, MNIST images were encoded into Poisson spike trains using a rate-based encoding scheme. Original MNIST dataset consists of 28×28 images represented in a grayscale format with pixel values ranging from 0 to 255. However, firing rates in biologically plausible networks are below 200 Hz [72], for this reason a typical preprocessing step is to scale down pixel values by a factor β in order to reduce this effect; to this aim, we opted for a factor $\beta = 4$, resulting into spike trains with firing rates between 0 and 63.75 Hz. The proposed Spiking-AE architecture consists of input and output layers made of 784 (i.e., 28×28) neurons, and a code layer of 300 neurons, where this value has been found by proportionally increasing the bottleneck size from 100 to 500 with steps of 100 units, until we reached a good trade-off in terms of performance and trainable parameters.

For the LIF parametrization we used a resting potential V_{rest} of -70 mV and a membrane time constant of 100 ms. While the latter parameter diverges from typically observed biological values, it allows the neurons to have a better estimation of the input firing rate as information related to past spikes will vanish more slowly [27]. With respect to the A^+_{in}, A^+_{out}, and A^+_{code} update values we used 0.015, 0.015 and 1, respectively, a time decay constant of 20 ms for τ_w, τ_{in}, τ_{code} and τ_{out}, and a learning rate of 0.003 for the weights update.

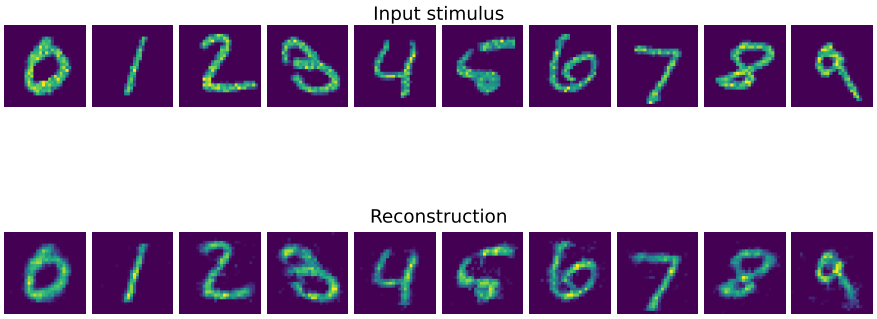


Figure 5.4: Spiking-AE MNIST reconstructions.

Fig. 5.4 shows our Spiking-AE reconstruction capabilities on the MNIST

dataset after a training phase of 20 epochs using the proposed learning rule. Input stimuli and reconstructions are obtained considering the total number of spikes during a simulation for each neuron in both the input and the output layers, reshaping the result in a 28×28 matrix. The network is able to reproduce the entire range of classes, demonstrating the effectiveness of our solution.

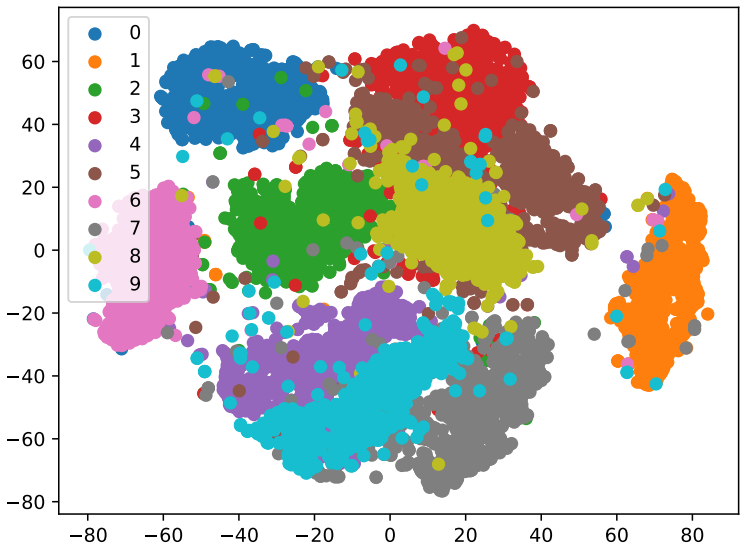


Figure 5.5: MNIST embeddings produced by the Spiking-AE.

Fig. 5.5 illustrates the latent representations generated by Spiking-AE, where each point corresponds to the spike count activity of neurons in the code layer. Since the code layer consists of 300 neurons, the resulting embeddings naturally reside in a 300-dimensional space. To visualize these high-dimensional features, we applied t-distributed Stochastic Neighbor Embedding (t-SNE), a nonlinear dimensionality reduction technique that projects complex data into two dimensions while preserving local neighborhood relationships.

As shown in the figure, the Spiking-AE successfully organizes the MNIST input into 10 well-separated groups, each corresponding to a distinct digit class. This outcome highlights the model's ability to extract and compress salient features from spiking activity into an efficient low-dimensional latent code. More-

over, clear class separation validates the effectiveness of the proposed learning rule, which not only minimizes the reconstruction error, but also enforces meaningful structure in the latent space, facilitating downstream tasks such as classification or anomaly detection.

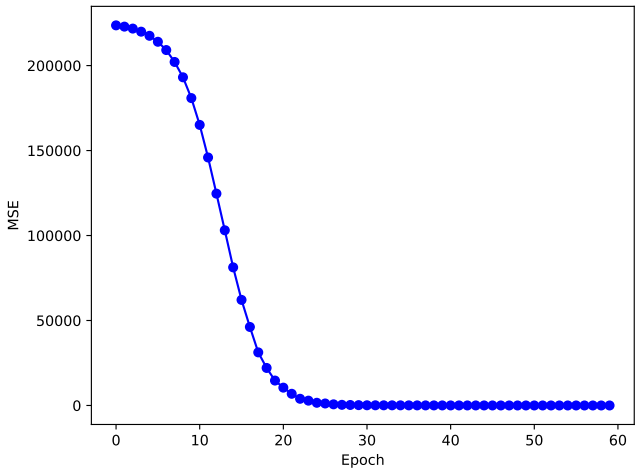


Figure 5.6: MSE curve during the Spiking-AE training phase.

Fig. 5.6 depicts the MSE curve obtained during the reconstruction block training using only normal data. MSE is calculated as the squared difference between the output and the input spike count, averaged over all neurons in the output layer. As we would expect, it tends to decrease with the number of epochs, demonstrating the correctness of the proposed learning in adjusting the Spiking-AE weights so that the reconstruction error is minimal.

5.3.2 Anomaly Detection Results (MFPT)

In this section, we evaluate the anomaly detection capabilities of the proposed Spiking-AE using the MFPT dataset, which contains vibration signals from bearings under normal and faulty conditions. The preprocessing pipeline applied to this dataset - including feature extraction with a gammatone filterbank and spike train encoding through asynchronous delta modulation - [73] has already been described in detail in Chapter 4 and is therefore not repeated here.

Unlike the reconstruction task, anomaly detection requires the full architecture shown in Fig. 5.3, which combines the reconstruction and detection blocks. The reconstruction block consists of 16 input neurons, 8 code neurons, and 16 output neurons, where the code size was selected through empirical trials to balance performance and model complexity. The anomaly detection block is composed of 32 comparison neurons (i.e., $2 \cdot n$) and a single output neuron. Each pair of comparison neurons receives both the raw input and the reconstructed output, connected through excitatory and inhibitory synapses, respectively.

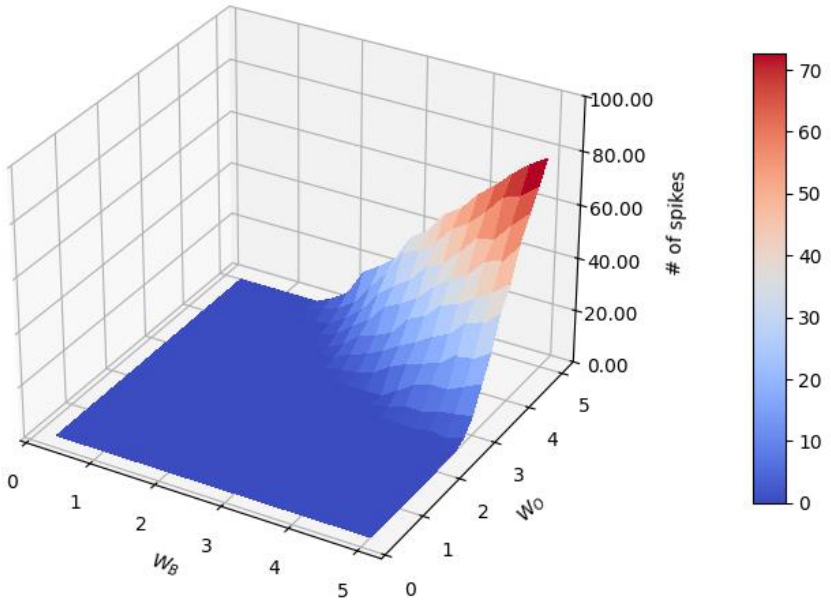


Figure 5.7: Anomaly detection block spiking activity with normal data when varying W_B and W_O weights.

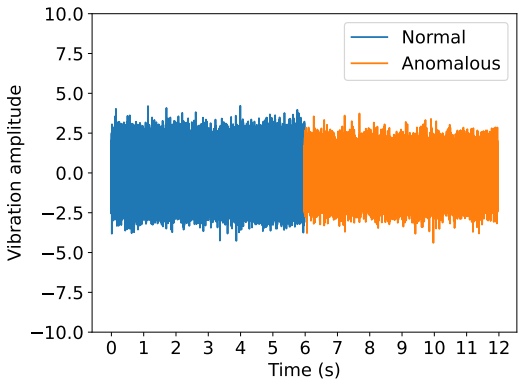
In the anomaly detection block, as discussed in 5.2.2, no training procedure is applied. Instead, synaptic weights W_B and W_O must be carefully tuned to prevent undesired spiking activity during normal operating conditions. If these weights are not properly chosen, the block could incorrectly fire even when only normal data is presented, leading to false positives.

To address this, we used an additional set of normal MFPT signals for weight calibration and validation. The goal of this process was to: (i) ensure that the

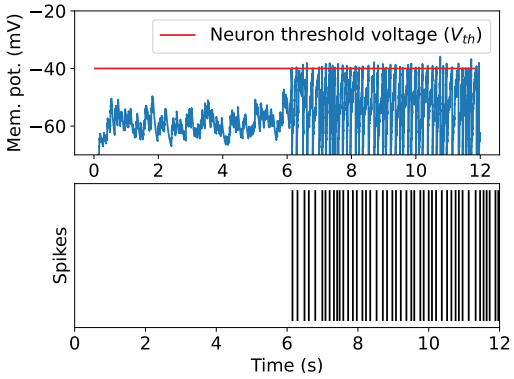
anomaly detection block remains silent when processing normal data, and (ii) maximize its spiking response when anomalous signals are encountered. Fig. 5.7, illustrates the effect of varying the values of W_B and W_O (from 0.25 to 5) on the block firing activity. As expected, increasing the weight magnitudes amplifies the overall activity of the block, making it more sensitive to input perturbations.

This tuning step therefore plays a crucial role in balancing sensitivity and stability. In our experiments, we identified $W_B = W_O = 3$ as the most suitable configuration, since it provided a stable system (no spiking with normal data) while maintaining strong responsiveness to anomalies.

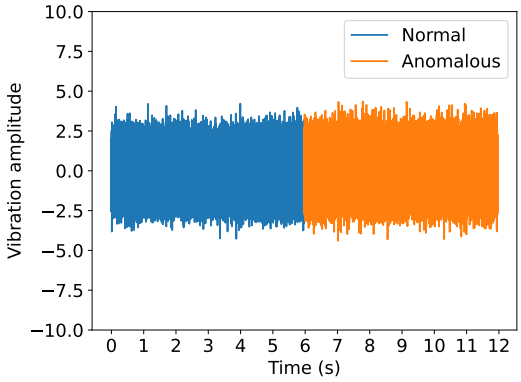
After the weights validation phase, we tested our model to evaluate its detection capabilities when fed with normal and anomalous data. Having an unseen set of normal data and three sets of anomalous data, we concatenated them to obtain every possible normal-anomaly pair, resulting in the behavior depicted in Fig. 5.8. The image shows that the model performs just as we expected, maintaining the membrane potential of the anomaly detection block output neuron around -60 mV for the first part of the experiments without the emission of any spike, thus demonstrating the capability of our model to correctly detect a normal behavior. After 6 s, we can observe the insurgence of an anomalous condition. Since the reconstruction block is trained only on normal data, it is not able to properly reconstruct the anomalous signals, hence producing a large mismatch between the input and output layer firing rates. Such a condition generates a huge imbalance between the excitatory and inhibitory activities in W_B synapses, abruptly changing the output neuron membrane potential, and causing the emission of spikes in the anomaly detection block with a firing rate of about 10 Hz. Such a result demonstrates the correctness of the weights selection process and the effectiveness of the proposed approach in quickly detecting the insurgence of anomalies.



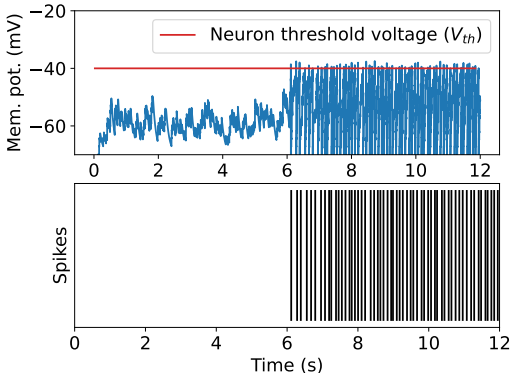
(a) First experiment.



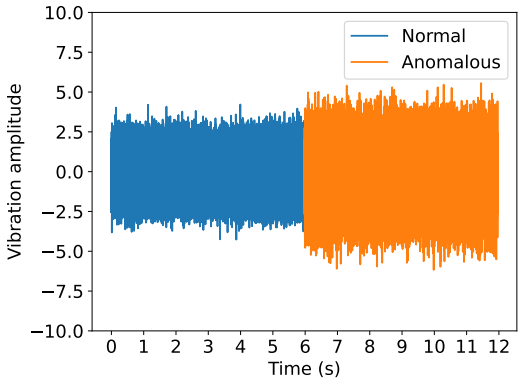
(b) Spiking-AE inference on the first experiment



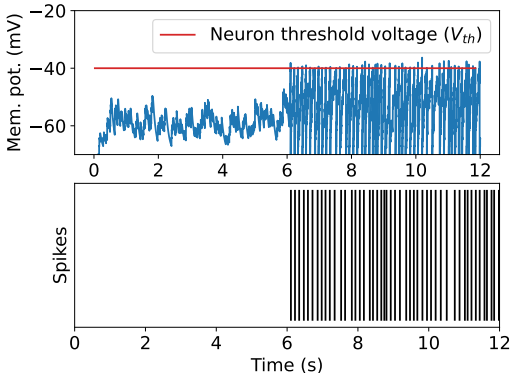
(c) Second experiment.



(d) Spiking-AE inference on the second experiment



(e) Third experiment.



(f) Spiking-AE inference on third experiment

Figure 5.8: Inference tests.

5.3.3 Comparative Analysis

From a qualitative point of view, in Table 8.3 we present the results obtained from comparing the proposed approach with other methods we found in the literature,

Table 5.1: Comparisons with other solutions found in the literature

Method	MSE	PC	Code size	# Trainable parameters
Mirrored STDP [68]	-	0.91	$5,000$	$3,920,000$
Sparse Spiking-AE [69]	56.65	–	$5,000$	$7,840,000$
Proposed solution	45.24	0.97	300	$235,200$

considering the MNIST test set. In this sense, our solution achieved a MSE of 45.22, resulting in the lowest value when compared with the other spiking methods. In this case, MSE is calculated as the squared difference between the output and the input spike count, averaged over all neurons in the output layer. In terms of Pearson Coefficient (PC), our model reached comparable results with the other solutions, while drastically reducing the number of training parameters thanks to the tied weights, thus resulting in the best trade-off in terms of performance and memory occupation.

5.4 Summary

In this chapter,a novel learning rule for training vanilla Spiking Autoencoder (Spiking-AE) models was presented. The proposed rule leverages presynaptic and postsynaptic activity traces to support strictly online synaptic weight adaptation, ensuring biologically inspired and locally computable learning dynamics. Furthermore, a modular tied Spiking-AE architecture is presented with the objective of efficiently addressing both reconstruction and anomaly detection tasks.

Extensive experiments conducted on the MNIST and MFPT datasets validated the effectiveness of the proposed framework. The results demonstrated competitive performance compared to existing approaches in the literature, while simultaneously reducing memory and computational overhead through the tied-weight mechanism.

Chapter 6

Background on Bio-Inspired Approaches for Edge Intelligence

6.1 From Cloud-Centric AI to Edge Intelligence

Over the past decade, machine learning (ML) has progressed significantly thanks to advances in large-scale cloud infrastructures and high-performance computing systems. These developments enabled the training and deployment of deep neural networks at an unprecedented scale, supporting applications across domains such as computer vision, natural language processing, and healthcare analytics.

However, this *cloud-centric paradigm* comes with limitations: it relies heavily on continuous connectivity, introduces latency in real-time applications, and raises privacy concerns when sensitive data must be transmitted to remote servers. These issues motivated a shift toward *Edge Intelligence*, where computation is moved closer to the data source. In this setting, inference tasks are executed directly on the device that generates the data (e.g., embedded systems, smartphones, or wearable), or on nearby nodes, reducing latency and improving privacy.

In practice, many modern systems still rely on a *hybrid approach*, where computationally expensive training is performed in the Cloud, and pre-trained models are deployed at the Edge for inference [74]–[76]. This approach leverages the

Cloud's virtually unlimited resources for model optimization while taking advantage of the Edge for low-latency, privacy-preserving inference. Yet, such systems remain dependent on periodic Cloud connectivity for model updates, which restricts their adaptability to dynamic and personalized environments.

More recently, research has extended Edge Intelligence beyond inference toward *on-device training*. In this paradigm, devices are not only consumers of pre-trained models but also capable of local learning. By enabling training directly on constrained devices, it becomes possible to achieve real-time personalization, continual adaptation, and privacy-preserving intelligence without relying on constant Cloud access. This marks a critical step toward the vision of fully autonomous and adaptive Edge AI.

6.2 On-Device Training: Importance and Challenges

The rapid proliferation of Internet of Things (IoT) devices has led to an enormous volume of data generated at the network edge. Applications such as autonomous driving, smart homes, wearable health monitoring, and industrial automation increasingly require immediate decision-making and adaptation close to the data source. On-device training addresses this demand by allowing models to learn directly from local data streams, enabling fast, personalized, and private intelligence.

At the same time, advances in specialized hardware, including GPUs, FPGAs, and AI accelerators for embedded platforms have made it feasible to run increasingly complex models on resource-constrained devices. This evolution paves the way not only for efficient on-device inference but also for enabling training process locally accompanied by careful architecture designs and algorithm selection.

Despite these opportunities, enabling training directly on constrained devices remains challenging. Below, we outline some of the key obstacles:

- **Mismatch between resource demand and availability:** IoT and embedded devices typically have memory capacities in the order of kilobytes or megabytes, whereas training deep models often requires storage on the scale of gigabytes. Gradient-based optimization with backpropagation (BP) exacerbates this issue, as it requires storing intermediate activations and loss values across all layers, consuming significant memory and computational resources.

- **Excessive parameter and computation requirements:** State-of-the-art models often involve millions of trainable parameters. Updating them not only increases energy consumption but also leads to long execution times, conflicting with the strict latency and efficiency requirements of on-device applications.

- **Backward error transport:** BP relies on the backward propagation of error signals through the network. This mechanism introduces additional communication overhead inside the device and significantly raises latency, which is particularly problematic for real-time edge applications.

- **Precision mismatch with hardware:** Most embedded devices are designed around low-power, low-precision arithmetic, while BP typically assumes high-precision floating-point operations. This mismatch reduces efficiency and hinders the deployment of standard training pipelines on microcontrollers.

- **Device heterogeneity:** IoT devices are diverse and heterogeneous. Therefore, finind a single and general solution for all devices is a challenging task.

- **Limited tailored optimization algorithms:** Most classical optimization techniques (e.g., Batch Normalization, Adam) focus on improving accuracy or convergence speed without considering resource constraints such as memory, power consumption, or real-time adaptability, limiting their practical

adoption on Edge hardware.

These limitations highlight the necessity of exploring alternative training mechanisms that are lightweight, scalable, and compatible with constrained hardware. adaptability, and hardware feasibility.

6.3 Toward Bio-Inspired Alternatives for Edge Learning

To overcome the bottlenecks of backpropagation, researchers have proposed a variety of biologically inspired learning mechanisms that align more closely with the local and parallel nature of neural processing in the brain. These methods aim to provide scalable learning rules that reduce memory overhead and avoid explicit backward error transport.

Among them, the *Forward–Forward (FF)* algorithm, which has emerged as a promising candidate [11]. The key idea in this algorithm is to alternate between forward and backward passes, FF replaces backpropagation with two forward passes: one on positive data and one on negative data. The network learns by maximizing the "goodness" of positive activations while minimizing that of negative activations. A detailed overview of the FF algorithm, as proposed for the first time by [11], is provided in the following section.

6.4 Forward–Forward Algorithm (Hinton, 2022)

For completeness, as discussed in Section 6.2, backpropagation (BP) remains the *de facto* standard for training deep neural networks, but its biological implausibility and computational demands make it unsuitable for constrained environments. To address these limitations, Hinton introduced the *FF* algorithm [11]. Basically, it replaces the forward–backward procedure with two forward passes, each with opposite objectives. As a result, the network is decomposed into independent

layers, where each layer can be trained locally using only its forward activations. This significantly reduces memory and energy requirements, as there is no need to store gradients or activations from previous layers, making FF more hardware-friendly and particularly suited for edge scenarios. The trade-off is that FF can be slower to converge and may generalize less effectively than BP, but it remains highly relevant in contexts where energy efficiency is critical.

The core concept of FF is based on the notion of *goodness*, defined as the sum of squared activations in a given layer l. If a_j^l is the activation of the j-th neuron at layer l, we have:

$$a_j^l = \sigma\Big(\sum_i w_{i,j}^l x_i + b_j^l\Big), \tag{6.1}$$

$$g^l = \sum_j (a_j^l)^2, \tag{6.2}$$

where $w_{i,j}^l$ and b_j^l are the weights and bias terms, x_i the input of the previous layer, and $\sigma(\cdot)$ the activation function.

Training consists of two forward passes:

- On *positive data* (i.e., real data), weights are updated to increase the layer goodness.
- On *negative data* (i.e., corrupted or mismatched data), the weights are updated to decrease layer goodness.

The learning objective is thus to make each layer highly "excited" (high goodness) when processing real input, and minimally activated when exposed to corrupted ones. This is achieved via the following layer-wise loss function:

$$\mathcal{L}^l = \alpha \cdot \ln\left(1 + e^{(\theta - g_{pos}^l)}\right) + (1 - \alpha) \cdot \ln\left(1 + e^{(g_{neg}^l - \theta)}\right), \tag{6.3}$$

where α is a scaler value set to 1 when passing the positive data and 0 in the negative pass, g_{pos}^l and g_{neg}^l are the goodness values for positive and negative

samples, respectively, and θ is a threshold that separates the two regimes. As highlighted in [11], the threshold θ can be set to any value that drives the layer goodness toward higher values for positive data and lower values for negative data. In practice, this effect is typically achieved by selecting symmetric threshold values, which ensure that the model adjusts its weights in "opposite directions" depending on whether the input is real or corrupted.

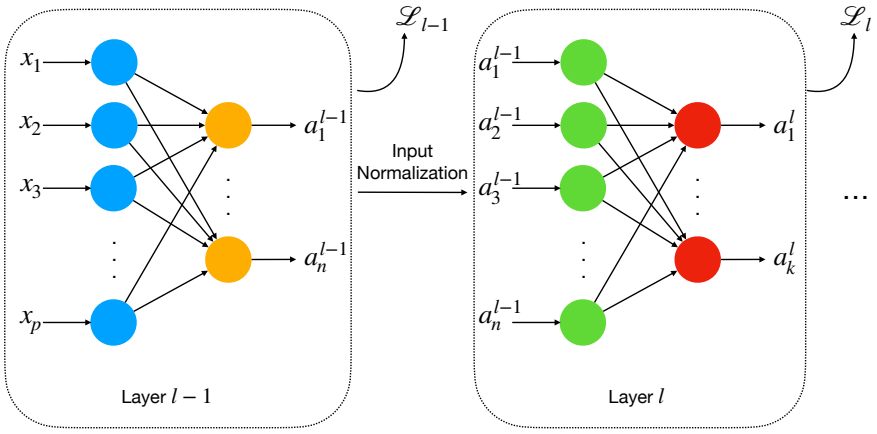


Figure 6.1: A neural network architecture trained using the FF algorithm.

Fig. 6.1 depicts a neural network architecture that is trained using the FF algorithm. as illustrated, each layer has its own loss function $\mathcal{L}$ and is individually trained without the need for any gradient backpropagation. A further distinction from BP lies in the normalization step applied to the output of each layer before passing it forward. This step is essential to prevent feature redundancy and ensures that subsequent layers focus on extracting new complementary information rather than reusing what has already been captured by earlier ones.

Since models trained with FF do not rely on an explicit output classifier (such as a softmax layer), supervised learning tasks require a label injection mechanism. As suggested in [11], the labels are concatenated with the input so that positive samples correspond to correctly labeled data and negative samples correspond to mislabeled data. During inference, predictions are made by testing each possible label and selecting the one that yields the highest goodness score, as shown in Algorithm 2.

Algorithm 2: Forward-Forward prediction.

1 **Input**:
2 d: data sample
3 K: # of classes
4 **Output**:
5 $\hat{y}$: prediction
6
7 G $\leftarrow \{\varnothing\}$
8 **for** $k \in K$:
9 $d_k = insert_label(d, k)$
10 $a \leftarrow FF_model(d_k)$
11 $g_k \leftarrow \sum a^2$
12 $update_set$(G,g_k)
13 $\hat{y} \leftarrow \arg\max(G)$
14 **return** $\hat{y}$

The algorithm takes as input a data sample d, for which the model must predict the label, and the total number of classes K. An empty set G is initialized to store the goodness values for each candidate class (line 7). The algorithm then iterates over all classes $k \in K$ (line 8), where the input d is combined with the label k using the $insert_label(\cdot)$ function (line 9). The resulting labeled input d_k is passed through the model, which outputs the activations of the last layer (line 10). From these activations, the goodness score g_k for class k is computed using Eq. 6.2 (line 11) and appended to the set G (line 12). Finally, the prediction $\hat{y}$ is obtained by applying the arg max operator to G (line 13), returning the class with the highest goodness value (line 14).

This approach enables FF-based models to perform classification tasks in a lightweight manner. Although they may require more data to reach the same level of accuracy as conventional methods, their efficiency in terms of memory usage and hardware demands makes them particularly well-suited for *on-device learning* on MCUs and other resource-constrained platforms.

The FF algorithm not only enhances efficiency but also offers a degree of biological plausibility, as it relies on local weight updates derived from activity patterns rather than global error signals. These characteristics make FF partic-

ularly appealing for Edge scenarios, where resource limitations and the need for online adaptation demand alternatives to conventional training.

In the next chapter, we build upon these ideas and present the *μ-FF framework*, a bio-inspired learning strategy tailored for resource-constrained microcontrollers. By combining the Forward–Forward algorithm with lightweight neural architectures, we demonstrate how on-device training can be realized in practice under strict hardware limitations.

Chapter 7

Forward–Forward Learning on Microcontrollers

7.1 Introduction

One of the main challenges in on-device training lies in the limited computational resources of edge devices such as microcontrollers (MCUs). Traditional deep learning training relies on algorithms like BP (see Sec. 6.2), which require large memory, high-precision arithmetic, and powerful computing resources that are simply not available on most embedded platforms.

Several techniques have been proposed to address these limitations. Weight compression, quantization, and lightweight neural network architectures reduce memory footprint and model complexity [77], but they mainly optimize the inference stage rather than training itself. Furthermore, these methods often demand careful hyperparameter tuning or even complete model re-training, which is time-consuming and impractical in dynamic environments.

Another promising approach is federated learning, which enables distributed training of DNNs directly on MCUs without sharing raw data, thereby preserving privacy [78]. Despite its advantages, this paradigm becomes less effective in scenarios where stable connectivity cannot be guaranteed or when models need to

adapt to new patterns in real time. Transfer learning provides another alternative by reusing pre-trained models for related tasks [79]. However, such techniques still partially rely on the computationally demanding backpropagation process, and their performance can degrade significantly when the target dataset distribution differs from that of the pre-training samples.

Within this context, the *Forward–Forward (FF)* algorithm [11] has emerged as a bio-inspired and hardware-friendly alternative to traditional backpropagation, particularly suited for resource-constrained environments. Unlike BP, which requires both forward and backward passes with costly gradient computations, FF relies only on two forward passes with opposite objectives. This eliminates the need for storing activations and propagating errors across the entire network, making the learning process simpler, more efficient, and better aligned with the limited memory and energy budgets of MCUs. By reducing computational overhead, FF opens the door to training directly on-device, where conventional backpropagation is often infeasible.

7.2 Problem Statement

Despite its advantages, the original FF formulation still requires gradient computations with respect to the loss function, which can be too demanding for devices with limited arithmetic capabilities. Moreover, it remains in an exploratory phase and has not yet been validated in diverse scenarios, such as MCUs deployment, leaving open questions regarding its feasibility in terms of memory footprint and real-time adaptability.

7.3 Use Case

To address this gap, we introduce μ-FF, a variation of the Forward–Forward algorithm specifically tailored for MCUs. In μ-FF, the training process is reformulated as a multivariate Ridge regression problem with a closed-form solution,

using the Mean Squared Error (MSE) as the loss function. This approach eliminates the need for explicit gradient computations and iterative optimization, thereby reducing memory requirements and avoiding gradient storage. As a result, μ-FF enables efficient on-device training even on highly resource-constrained platforms such as the STM32H747I-DISCO board. To validate its feasibility, we conducted experiments on the MNIST dataset for a supervised classification task, demonstrating that μ-FF can effectively support learning directly on embedded hardware.

7.4 Proposed μ-FF

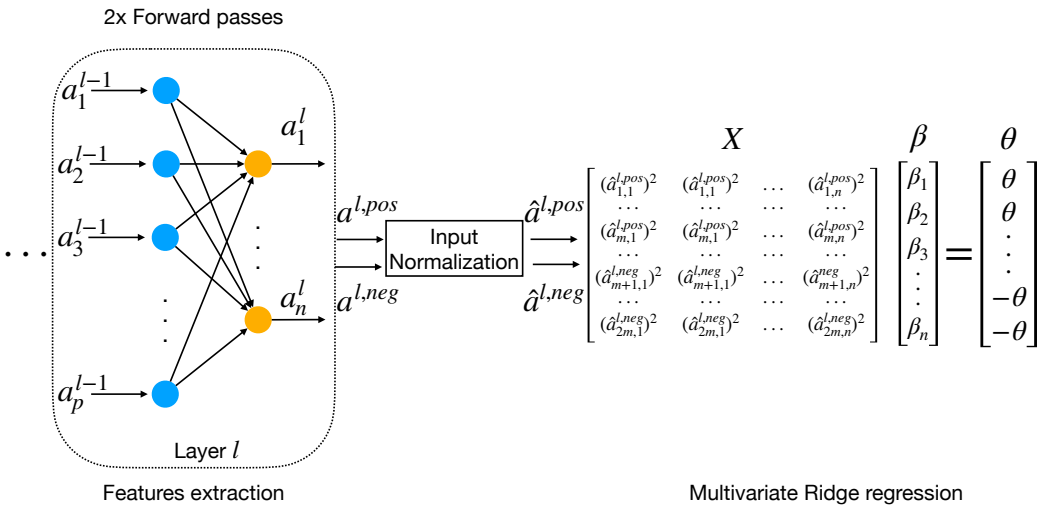


Figure 7.1: Proposed μ-FF model architecture.

Fig. 7.1 illustrates the μ-FF model architecture, which is composed of two main blocks. The first is dedicated to feature extraction; an important step for handling non-linear input data. At this stage, layer l processes the input twice: once with a positive representation and once with a negative representation of the data, producing the corresponding activations. These activations are then forwarded to the second block as input. Its important to note that this first block does not involve any training; its sole purpose is to apply non-linear transformations to the data.

The actual training process is performed in the second block by solving a multivariate Ridge regression problem. This is achieved by using the MSE as loss function which defined as:

$$\begin{aligned}\mathcal{L} =& \frac{1}{m}\sum_{i=1}^{m}[\alpha(\sum_{j=1}^{n}\beta_j(a_{i,j}^{l,pos})^2-\theta)^2 \quad +(1-\alpha)(\sum_{j=1}^{n}\beta_j(a_{i,j}^{l,neg})^2+\theta)^2] \\ &+\lambda\cdot\sum_{j=1}^{n}\beta_j^2\end{aligned} \tag{7.1}$$

where m is the number of training samples, α is the same scalar value defined in eq. (6.3), θ is a threshold value, λ is the Ridge regularization term, β_j is the j-th predictor coefficient, and $a_{i,j}^{l,pos}$ and $a_{i,j}^{l,neg}$ are the j-th positive and negative neuron activations referred to the i-th training sample; these are generated by layer l-th when fed with positive and negative data, respectively.

The are two reasons behind adapting the MSE as a loss function in our framework; First, it enables us to reformulate the training process as a multivariate Ridge regression problem, where the input features correspond to the squared activations of the l-th layer neurons, and the targets are defined as θ for positive activations and $-\theta$ for negative ones (see Fig. 7.1). Using opposite target values enforces a clear separation. Second, this formulation admits a closed-form solution, eliminating the need for gradient descent or other iterative optimization methods. This property is particularly advantageous in on-device scenarios, as it ensures predictable and relatively constant training time while significantly reducing computational complexity, making the approach well-suited for resource-constrained hardware such as MCUs.

The proposed μ-FF training procedure is detailed in Algorithm 4. It requires four inputs: the positive and negative training sets (X_{pos} and X_{neg}), obtained by merging data with their labels, the regularization parameter λ, and a threshold value θ. The target values Y_{pos} and Y_{neg} are then assigned as θ and $-\theta$, respectively (lines 8–9). For each sample, the positive and negative activations (a_{pos} and

Algorithm 3: μ-FF training.

Input:
  X_{pos}: positive training data
  X_{neg}: negative training data
  λ: Ridge regularization parameter
  θ: threshold
Output:
  β: solution vector

$Y_{pos} \leftarrow \theta$
$Y_{neg} \leftarrow -\theta$

$a_{pos} \leftarrow Dense(X_{pos})$
$a_{neg} \leftarrow Dense(X_{neg})$

$\hat{a}_{pos} \leftarrow normalize(a_{pos})$
$\hat{a}_{neg} \leftarrow normalize(a_{neg})$

$X \leftarrow concatenate(\hat{a}^2_{pos}, \hat{a}^2_{neg})$
$Y \leftarrow concatenate(Y_{pos}, Y_{neg})$

$\beta \leftarrow (X^\top X + \lambda I)^{-1} X^\top Y$
return β

a_{neg}) are computed using the $Dense(\cdot)$ function (lines 11–12) and subsequently normalized (lines 14–15). Considering m training samples and n hidden neurons, a design matrix $X \in \mathbb{R}^{2m \times n}$ and a target vector $Y \in \mathbb{R}^{2m \times 1}$ are constructed by concatenating the activations with their corresponding targets (lines 18–19). The optimal solution vector β that minimizes the MSE loss is then computed via the normal equation (line 21) and returned as output (line 22). Once β is obtained, the layer goodness can be evaluated as $g = (a^l \cdot \beta)^2$, consistent with 2

7.5 Experiment and Results

7.5.1 Dataset

The proposed algorithm was evaluated using the Fashion-MNIST dataset, a widely adopted benchmark in machine learning research. The dataset consists of 70,000 grayscale images of size 28×28, categorized into 10 product classes: T-shirt/Top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. It is pre-divided into 60,000 training samples (6,000 per class) and 10,000 test samples (1,000 per class).

Due to the limited amount of memory available on the device, tests were conducted on a reduced version of the above-described dataset, using half of the classes, namely: $T\text{-}shirt/Top$, $Trouser$, $Sandal$, $Shirt$, and $Dress$. Specifically, we designed three different experiments considering three, four, and five classes, where for a fair comparison the following scheme has been adopted:

- **3 classes**: $T\text{-}shirt/Top$, $Trouser$, $Sandal$;
- **4 classes**: $T\text{-}shirt/Top$, $Trouser$, $Sandal$, $Shirt$;
- **5 classes**: $T\text{-}shirt/Top$, $Trouser$, $Sandal$, $Shirt$, $Dress$.

To assess robustness, we conducted three independent experiments by varying the number of training samples per class, specifically using 100, 200, and 300 samples.

7.5.2 Hardware Settings

Enabling TinyML requires specialized tools that simplify and optimize the deployment process on highly resource-constrained devices. While popular Python-based frameworks such as TensorFlow and PyTorch are widely used for model development and training, deploying these models onto MCUs often requires re-implementation in low-level languages (e.g., C) and the application of memory optimization techniques. This process is both time-consuming and demands deep expertise in embedded systems. To address this, several target-dependent optimization and deployment tools have been developed.

In this project, we adopted X-CUBE-AI [80] to port the proposed ML algorithm onto the STM32 MCU. X-CUBE-AI is part of the STM32Cube.AI ecosystem and extends STM32CubeMX capabilities by automatically converting pre-trained AI models (neural networks and classical ML) into optimized C libraries that can be directly integrated into user applications. The package also provides tools for desktop and on-device validation, performance benchmarking, and memory footprint analysis, without requiring manual C implementation.

X-CUBE-AI supports two of the most popular ML formats, TensorFlow™ Lite and Keras, as well as any framework exportable to the ONNX standard (e.g., PyTorch™, MATLAB®, Microsoft® Cognitive Toolkit). It leverages advanced code optimization techniques, structured into three key blocks:

- **Graph Optimizer:** Simplifies and optimizes the computational graph to better fit STM32 hardware architectures.
- **Quantizer:** Supports multiple quantization formats, from 32-bit floating point to 8-bit and even down to 1-bit (QKeras, Larq). Quantization significantly reduces latency, flash, and RAM usage while preserving accuracy.
- **Memory Optimizer:** Manages memory allocation between internal and external resources, with the option to dedicate memory specifically to the model, enabling easy updates.



Figure 7.2: STM32H747I-DISCO development platform.

More recently, the STM32Cube.AI Developer Cloud [81] was introduced, extending these capabilities with an online platform that:

- Provides a complete web-based interface (no local software installation needed).
- Offers remote benchmarking on real STM32 boards hosted by ST (board farm), eliminating the need for developers to own the hardware.
- Includes an STM32 model zoo with pre-trained reference models and training scripts.

For these experiments, we used the STM32H747I-DISCO development platform (Fig. 7.2) produced by STMicroelectronics. It integrates an Arm® Cortex®-M7-based STM32H747XIH6 MCU featuring 2 MB of flash memory (with 1024 KB available for AI) and 1 MB of RAM (with 512 KB available for AI). The board also includes a 4" LCD capacitive touchscreen, dual digital microphones, 512-Mbit SPI flash, 256-Mbit SDRAM, and multiple expansion connectors (Ethernet RJ45, camera, microSD™, audio, USB, Arduino V3). This makes it particularly well-suited for evaluating AI algorithms under real embedded constraints.

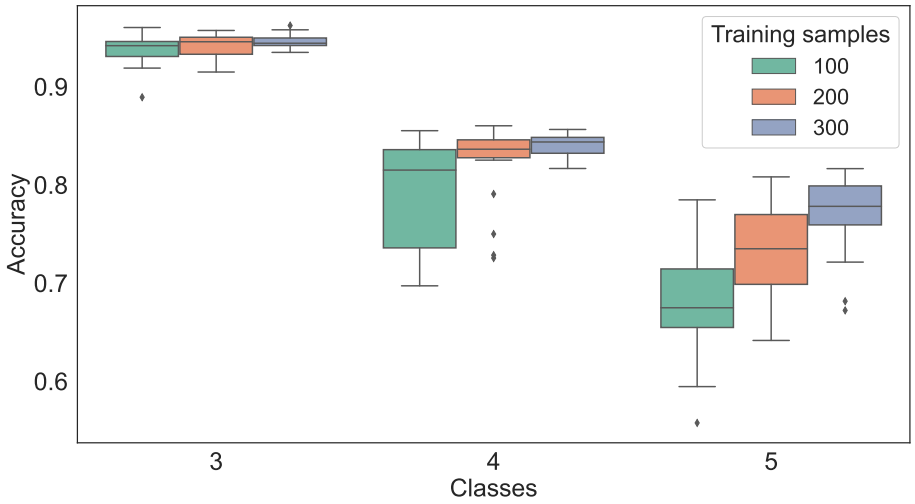


Figure 7.3: Accuracy reached by the proposed μ-FF approach when varying the number of classes and training samples.

7.5.3 Results

The classification task was addressed using a single dense (fully connected) layer neural network with 500 neurons and a Rectified Linear Unit (ReLU) activation function. A key factor influencing the model's performance is the choice of the regularization parameter λ, which controls the strength of the penalization term in the loss function (see Eq. 7.1). After testing values in the range $[0.001, 0.01, 0.1, 1.0, 10]$ through 20 runs for each configuration, $\lambda = 0.01$ was selected as it produced the best accuracy.

Fig. 7.3 reports a boxplot comparison of the accuracy achieved by the proposed method when varying both the number of classes (3, 4, and 5) and the number of training samples per class (100, 200, and 300). As expected, accuracy decreases as the classification task becomes more complex, though this effect is mitigated by larger training sets, which consistently improve performance. Overall, the results are encouraging, with average accuracies ranging from 94% to 73%, particularly given the very limited number of samples used for on-device training.

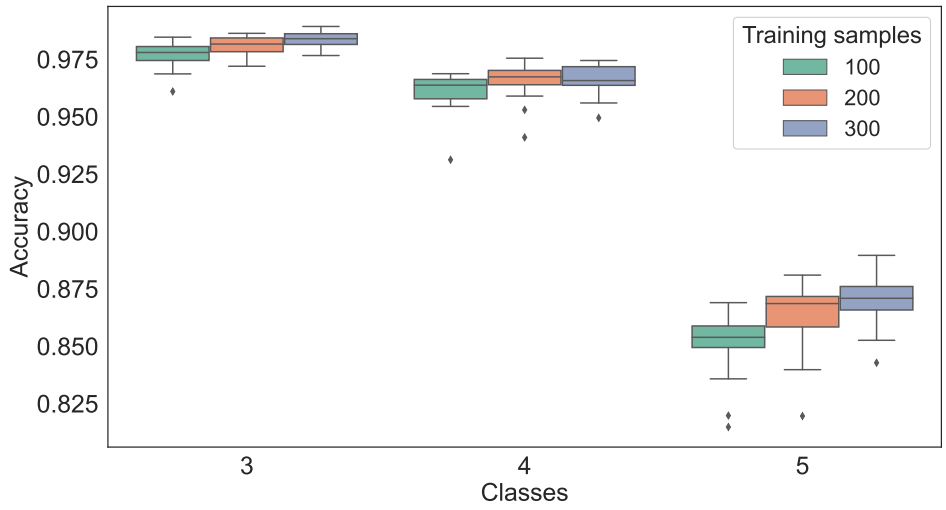


Figure 7.4: Accuracy reached by Dense models trained BP when varying the number of classes and training samples.

For comparison, we conducted the same experiments using a dense model trained offline with backpropagation (BP). The results, shown in Fig. 7.4, follow the same trend as those obtained with the proposed μ-FF algorithm but exhibit higher accuracy values (between 98% and 86%). Nevertheless, it should be emphasized that BP is computationally prohibitive for MCUs, making our approach a more practical trade-off between accuracy and resource efficiency. More detailed accuracy values with the corresponding confidence intervals 95% are reported in Tables 7.1 and 7.2.

Table 7.1: Accuracy reached by the model trained with the proposed μ-FF algorithm when varying the classification task and the number of training samples.

		Training samples		
	# of Classes	100	200	300
Accuracy	3	0.936 ± 0.014	0.942 ± 0.010	0.945 ± 0.006
	4	0.791 ± 0.050	0.822 ± 0.038	0.840 ± 0.011
	5	0.679 ± 0.055	0.732 ± 0.044	0.770 ± 0.037

While the performance metrics obtained from the X-CUBE-AI tool, including MACC operations, training time, inference time, and memory footprint are reported in Table 7.3. The comparison involves our on-device classification model

Table 7.2: Accuracy reached by the model trained with BP algorithm when varying the classification task and the number of training samples.

		Training samples		
	# of Classes	100	200	300
Accuracy	3	0.977 ± 0.005	0.981 ± 0.003	0.983 ± 0.002
	4	0.960 ± 0.007	0.965 ± 0.007	0.966 ± 0.006
	5	0.851 ± 0.013	0.863 ± 0.013	0.869 ± 0.009

trained on Fashion-MNIST with the proposed μ-FF algorithm against three neural network models trained using traditional BP. For fairness, all models employ the same hidden layer with 500 ReLU neurons. In the BP case, however, a $softmax$ output layer must be added, with the number of neurons equal to the number of target classes. We evaluated models configured to classify three, four, and five classes.

Table 7.3: Models performance comparison in terms of MACC, training time, inference time, and memory footprint.

Model	Training time (ms) 100 samples	Training time (ms) 200 samples	Training time (ms) 300 samples	Inference time (ms)
μ-FF (3 classes)	6.012	12.004	17.994	15.76
μ-FF (4 classes)	8.009	16.000	23.989	15.78
μ-FF (5 classes)	10.007	19.994	29.981	15.80
BP (3 classes)	–	–	–	15.73
BP (4 classes)	–	–	–	15.74
BP (5 classes)	–	–	–	15.75

Model	MACC (Feature extraction)	MACC (Ridge) 100 samples	MACC (Ridge) 200 samples	Memory footprint
μ-FF (3 classes)	393K	150.5M	300.5M	1.5 MB
μ-FF (4 classes)	393K	200.5M	400.5M	1.5 MB
μ-FF (5 classes)	393K	250.5M	500.5M	1.5 MB
BP (3 classes)	394.5K	–	–	2.0 MB
BP (4 classes)	395K	–	–	2.0 MB
BP (5 classes)	396K	–	–	2.0 MB

7.6 Summary

In this chapter, we introduced μ-FF, a variation of the FF algorithm designed for highly constrained devices. The proposed method reformulates the training process as a multivariate Ridge regression problem, solved in closed form using MSE as the loss function. By eliminating the need for gradient computations, μ-FF significantly reduces memory usage and computational complexity, making it feasible to deploy on STM32-family microcontrollers.

Experimental results on the Fashion-MNIST benchmark confirm the effectiveness of the approach, showing that both training and inference can be performed directly on resource-limited hardware while maintaining competitive performance. This demonstrates the potential of μ-FF as a lightweight and hardware-friendly alternative to backpropagation for on-device learning.

Chapter 8

Extended Forward–Forward Framework with ESN-ViT

8.1 Motivation for Extending FF

Building on the foundation laid in the previous chapter 7, where we introduced μ-FF as a lightweight alternative to BP for on-device training, this chapter extends our exploration of the FF paradigm to more advanced neural architectures and application scenarios. The goal is to enhance performance while preserving the feasibility of running models on resource-constrained microcontrollers (MCUs).

Specifically, we investigate how FF can be integrated with different architectures that balance expressiveness and efficiency. Among these, we focus on two representative models: Vision Transformers (ViTs), which have achieved state-of-the-art results in visual recognition by capturing long-range dependencies across image regions, and Echo State Networks (ESNs), lightweight recurrent networks designed for efficient temporal processing with minimal training overhead. By combining the strengths of these models with the FF algorithm, we aim to demonstrate improved adaptability and scalability without compromising hardware constraints.

8.1.1 Vision Transformers (ViTs)

Transformers are a type of deep learning architecture, originally introduced for Natural Language Processing (NLP), and have recently been adapted to computer vision tasks, giving rise to the *Vision Transformer (ViT)* [82]. Unlike traditional Transformers applied to text, ViTs work with 3D inputs $\in \mathbb{R}^{H\times W\times C}$ (H and W are the height and the width while C is the number of channels). ViTs operate on sequences of image patches, enabling the network to focus on specific image parts in order to better extract local patterns, and to capture possible dependencies between patches.

The main steps of the ViT architecture are illustrated in Fig. 8.1:

1. **Patch extraction:** The input image is divided into fixed-size patches (e.g., 16×16 pixels), and then organized into a sequence of S flattened patches (with $S = \frac{H\cdot W}{P^2}$) $\in \mathbb{R}^{S\times P^2\times C}$ [83].

2. **Linear projection:** Each patch is linearly mapped into an embedding space of fixed dimension. A special learnable vector, known as the `[class]` token is added as the first element of the sequence of patch embeddings. During self-attention process, this token interacts with all patch embeddings and gradually collects a summary of the entire image, which will be used later by the classification head to predict the image label.

3. **Positional encoding:** Since the transformer architecture itself is permutation-invariant, positional encodings are added to each patch embedding to retain spatial information.

4. **Transformer encoder:** The sequence of embeddings is passed through a stack of transformer encoder blocks, each consisting of *multi-head self-attention*, normalization, and *feed-forward MLP* layers. This stage enables the model to capture long-range dependencies across patches.

5. **Classification head:** After several layers of encoding, the `[class]` token

is fed into an MLP head or a fully connected layer that outputs the final class prediction (e.g., digit, object type).

This mechanism allows ViTs to model global interactions between all image regions, often leading to state-of-the-art results in visual recognition tasks [84]. However, this expressiveness comes at the cost of requiring millions of trainable parameters and large-scale datasets for training, making direct deployment on edge devices highly challenging.

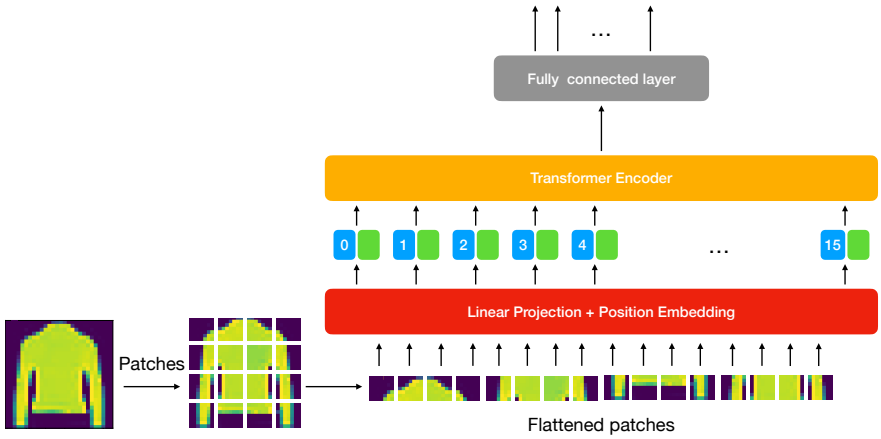


Figure 8.1: Standard Vision Transformer Architecture.

8.1.2 Echo State Networks (ESNs)

Echo State Networks (ESNs) are a class of Recurrent Neural Networks (RNNs) introduced by [85], belonging to the broader Reservoir Computing (RC) framework. They were designed to simplify the training of recurrent models while retaining the ability to model temporal dependencies.

An ESN consists of three main components: an input layer, a recurrent reservoir, and a trainable output layer. The reservoir is a large pool of sparsely and randomly connected neurons whose internal weights remain fixed after initialization. This reservoir acts as a high-dimensional dynamical system, projecting the input sequence into rich spatiotemporal representations. Unlike traditional RNNs, only the output weights are trained (often through linear regression),

which drastically reduces the number of trainable parameters and makes training computationally efficient, lightweight, and well-suited for resource-constrained environments.

Fig. 8.2 depicts an ESN architecture where the solid arrows indicate fixed-weight connections, while the blue dashed lines represent the trainable output weights.

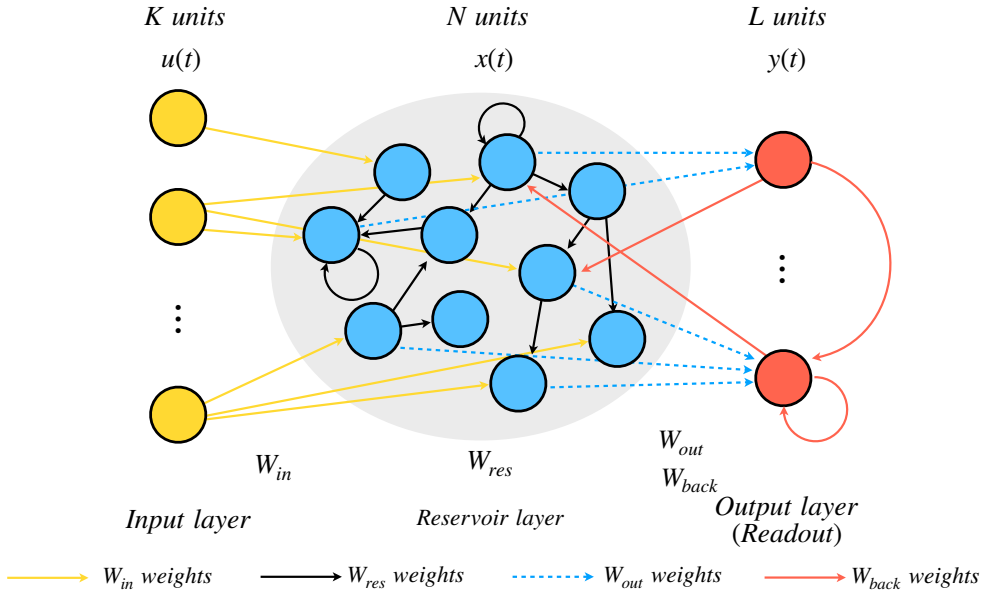


Figure 8.2: ESN Architecture.

Considering an ESN with K inputs, N reservoir units, and L outputs, the weight matrices are defined as: $W_{in} \in \mathbb{R}^{N \times K}$ (input to reservoir weights), $W_{res} \in \mathbb{R}^{N \times N}$ (reservoir recurrent connections), $W_{out} \in \mathbb{R}^{L \times N}$ (weights from reservoir neurons to output layer), and $W_{back} \in \mathbb{R}^{N \times L}$ (feedback connections between the output layer and reservoir). The dynamics (see [85], [86]) are given by:

$$x(t) = f(W_{in} \cdot u(t) + W_{res} \cdot x(t-1) + W_{back} \cdot y(t-1)), \tag{8.1}$$

$$y(t) = g(W_{out} \cdot [u(t), x(t), y(t-1)]), \tag{8.2}$$

where $x(t)$ is the reservoir state computed at the timestep t that acts as a "memory" element, which stores the temporal evolution of the network. In particular,

this state depends on the current input $u(t)$, the previous state $x(t-1)$ and the previous output $y(t-1)$. The equation is finally wrapped around an activation function $f(\cdot)$ (typically the hyperbolic tangent or sigmoid) that introduces a non-linearity.

The output is instead obtained as the concatenation of the new computed state $x(t)$, the current input $u(t)$ and the previous output $y(t-1)$. Similarly to RNNs the output of an ESN is affected by the "history" of the network itself. Also in this case an activation function $g(\cdot)$ is applied before producing the actual output of the network.

A distinctive property of ESNs is the *echo state property*, which ensures that the influence of past input fades over time, stabilizing the reservoir dynamics. Empirical evidence shows that a necessary condition for this property is that the spectral radius of W_{res} (the largest absolute eigenvalue) is less than one. Another desirable feature is *separation*, which ensures that different inputs lead to sufficiently different reservoir states, allowing meaningful feature extraction.

Unlike traditional RNNs, most ESNs weight matrices such as W_{in} and W_{res} are fixed sparse randomly initialized values. This peculiarity of ESNs greatly reduces the number of trainable parameters, since only W_{out} and W_{back} are optimized, thus significantly accelerating the training phase. However, this advantage comes with the drawback that ESN performance is sensitive to the initial random weights of the reservoir, making careful hyperparameter selection essential to ensure stability and reliability. A common strategy suggested in the literature is to employ large reservoirs, which can naturally give rise to small independent sub-networks capable of capturing a wide variety of input dynamics [85].

8.2 Proposed Algorithm

This section introduces the proposed variation of the ViT architecture that integrates Echo ESNs for feature extraction and adapts the FF paradigm to a

lightweight training strategy, which we call μ-FF. This framework is specifically designed for constrained devices such as MCUs, where both computational and memory resources are highly limited.

As discussed in section 6.2, BP is unsuitable for embedded environments due to its high memory requirements (storing activations, propagating gradients) and computational demands. While the FF algorithm [11] addresses these limitations by replacing the forward/backward passes of BP with two forward passes (on positive and negative data), directly applying it to complex architectures such as ViTs still remains prohibitive on MCUs.

To this end, we propose the following extension of [87]:

- An ESN–ViT hybrid model that replaces the heaviest components of ViT with ESN reservoirs to drastically reduce the number of trainable parameters while preserving feature extraction capability.
- An adapted μ-FF algorithm that reformulates the training process as a regression task with a Mean Squared Error loss (MSE), simplifying gradient computations and enabling efficient online learning.

8.2.1 Model Architecture(ESN-ViT)

Fig. 8.3 illustrates the proposed architecture. The key modifications are:

1. **Reservoir-based patch embedding:** Instead of a fully connected projection, each image patch is passed through an ESN cell. The reservoir transforms the input sequence into a high-dimensional spatio-temporal representation via its recurrent dynamics, enhanced by the non-linear tanh activation. This reduces the number of parameters while enriching features.
2. **Reservoir-based encoder:** The Transformer encoder block, which typically contains the majority of ViT parameters due to self-attention, is replaced with another ESN cell. Exploiting the separation property [85],

a larger reservoir can capture diverse input dynamics by forming quasi-independent sub-networks.

With these modifications, trainable parameters are limited to the final fully connected layers, simplifying the training process and substantially reducing the memory footprint. This design is particularly suitable for MCUs, where storing and updating millions of parameters is impractical.

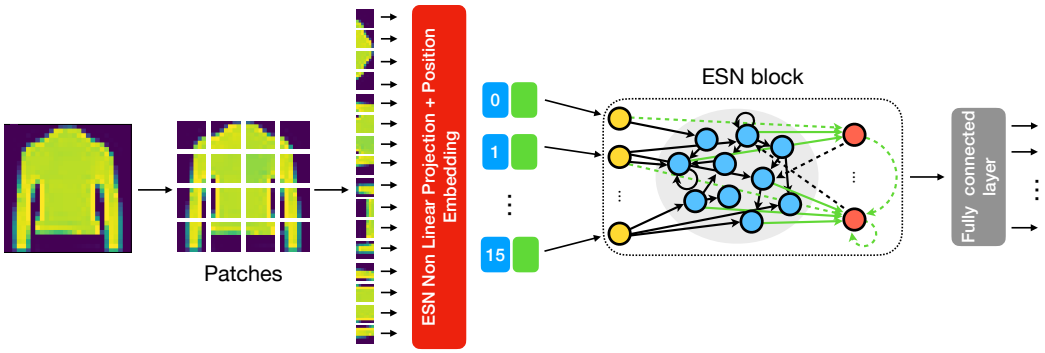


Figure 8.3: Proposed ESN-ViT architecture.

8.2.2 The μ-FF Algorithm

While the original FF loss function [11] is effective, it requires non-trivial computations for gradient estimation. To make the process feasible on constrained hardware, we propose μ-FF, which formulates training as a regression problem using the MSE loss. The loss for the l-th layer is defined as:

$$\mathcal{L}^l = \frac{1}{2m} \sum_{i=1}^{m} \alpha \cdot (g^l_{pos} - \theta)^2 + (1 - \alpha) \cdot (g^l_{neg} + \theta)^2 \tag{8.3}$$

where m is the batch size, α is set to 1 during the positive pass and 0 during the negative pass, θ is the target threshold, and g^l_{pos}, g^l_{neg} are the layer goodness values under positive and negative samples, respectively. Using symmetric targets ($+\theta$ for positive and $-\theta$ for negative) encourages the network to learn opposing responses to real and corrupted data, while keeping the computation lightweight.

The full training loop is reported in Algorithm 4. Key aspects include:

- Support for loading previously trained weights, enabling continual learning.
- Use of mini-batch gradient descent to reduce memory requirements and avoid large matrix inversions.
- Computation of activations and goodness values per batch, followed by local weight updates.

Algorithm 4: μ-FF algorithm.

Input:
  X_{pos}: positive training data
  X_{neg}: negative training data
  Y_{pos}: positive labels
  Y_{neg}: negative labels
  E: training epochs
  B: batch size
  η: learning rate
Output:
  W: weights matrix
  b: bias vector

if W **and** b *available*:
  $W, b \leftarrow load_parameters(\cdot)$
else:
  $W \leftarrow Random_init(\cdot)$
  $b \leftarrow Random_init(\cdot)$

$X \leftarrow concatenate(X_{pos}, X_{neg})$
$Y \leftarrow concatenate(Y_{pos}, Y_{neg})$

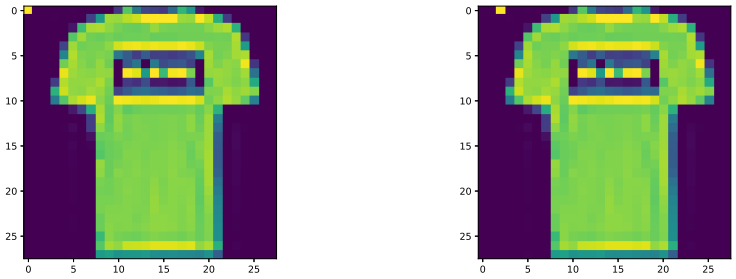
for $e \in E$:
  $shuffle(X)$
  $shuffle(Y)$
  for $batch \in B$:
    $a_{batch} \leftarrow ESN_ViT(X_{batch}, W, b)$
    $g_{batch} \leftarrow \sum a_{batch}^2$
    $\Delta \leftarrow (g_{batch} - Y_{batch}) \cdot \sigma'(X_{batch} \cdot W + b)$
    $W \leftarrow W - \eta(X^\top \cdot \Delta)$
    $b \leftarrow b - \eta \sum \Delta$
return W, b

8.3 Experimental Results

This section presents the experiments conducted to evaluate the proposed ESN-ViT model trained with the μ-FF algorithm. Our aim is twofold: i) to assess the classification performance of the approach on a standard benchmark dataset, and ii) to demonstrate the feasibility of on-device training directly on a constrained MCUs platform.

8.3.1 Dataset and Experimental Setup

We conducted experiments on the Fashion-MNIST dataset, a well-known benchmark for image classification tasks. The dataset contains $70,000$ grayscale images of size 28×28, evenly distributed across 10 product classes (e.g., *T-shirt/Top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot*). Following the standard protocol, the training set consists of $60,000$ samples ($6,000$ per class), while the test set contains $10,000$ samples ($1,000$ per class).



(a) Positive data sample. (b) Negative data sample.

Figure 8.4: Example of a positive data belonging to class 0 and the corresponding negative sample adopted during the FF training process.

During training, the Forward–Forward algorithm requires both *positive* and *negative* data. Labels are embedded in input images using one-hot encoding by modifying the top left corner pixels, as originally proposed in [11]. Positive samples include the correct label, while negative samples are generated by associating each image with an incorrect label (see Fig. 8.4). This strategy allows the same image to be reused for both passes, reducing memory requirements, since only

labels need to be stored.

8.3.2 Hardware Considerations

The on-device experiments were conducted on the STM32H747I-DISCO board. The device is equipped with 2 MB of flash memory (of which 1,024 KB are available for AI tasks) and 1 MB of RAM (with 512 KB dedicated to AI) This platform provides a realistic setting for evaluating resource-constrained training. Further technical details on the board were provided and discussed in Section 7.5

8.3.3 Discussion

The classification task was addressed using the proposed hybrid ESN–ViT model. In a standard ViT, the linear projection layer (a fully connected layer) maps each flattened image patch to a fixed-dimensional latent space, after which positional embeddings are added. In our modified ESN–ViT, this projection is replaced by an ESN reservoir cell: each patch is fed into the reservoir, where its recurrent dynamics and tanh non-linearity map it into a non-linear 300-dimensional latent space. This is followed by a second ESN block with 200 neurons, providing an additional level of feature extraction. While larger reservoirs could further improve feature separation, the hardware limitations of the MCU required a trade-off, with 200 neurons chosen as the best balance between accuracy and memory footprint. The features extracted by the reservoirs were passed to a fully connected layer of 100 neurons with ReLU activation. In this configuration, the only trainable parameters reside in the final fully connected layer, resulting in approximately 21 k parameters and a 83 KB memory footprint. For comparison, a ViT trained with BP required 23k parameters and 87 KB due to the additional softmax classifier. All hyperparameters were selected empirically using a validation set of 10,000 samples.

8.3.4 Classification Performance

Fig. 8.5 compares the classification accuracy of the ESN–ViT model trained with the proposed μ-FF algorithm against traditional BP across different training set sizes, ranging from $6,000$ to 500 samples per class. As anticipated, the accuracy of both approaches declines as the number of training samples decreases, which reflects the greater difficulty of generalization under data scarcity. While BP consistently achieves higher accuracy (from 84% down to 81%), its computational and memory requirements render it unsuitable for execution on MCUs. By contrast, μ-FF achieves competitive performance (from 78% down to 71%), but with the crucial advantage of being fully executable on-device. This demonstrates that μ-FF, although slightly less data-efficient, offers a practical and sustainable trade-off between accuracy and hardware feasibility. In real-world scenarios where energy efficiency and local adaptability are critical, this property makes μ-FF particularly valuable for Edge AI deployment. The detailed results with their 95% confidence intervals are provided in Table 8.1

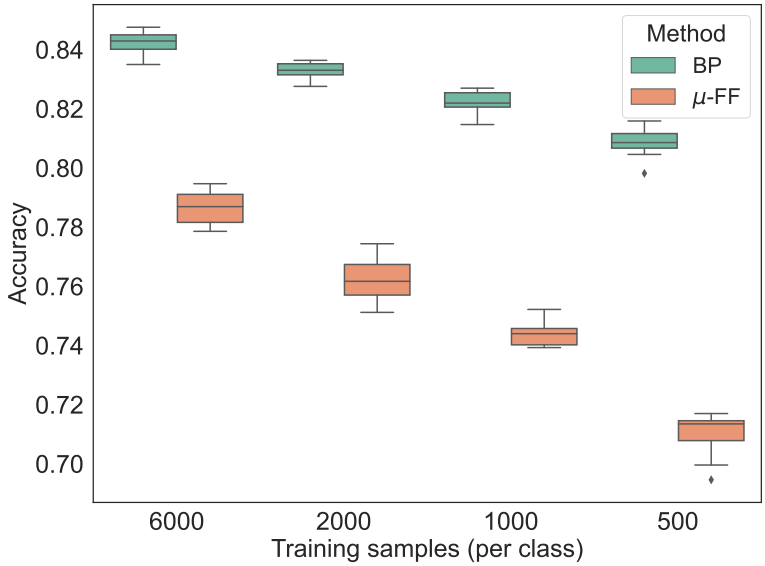


Figure 8.5: ESN-ViT accuracy performance when varying the training set size.

Fig. 8.6 illustrates the validation curves obtained during the training of the ESN–ViT with the proposed μ-FF algorithm. Both training and validation losses

Table 8.1: Comparison between BP and μ-FF when varying the training samples for each class

	Training samples (per class)			
Method	6000	2000	1000	500
BP	0.84 ± 0.005	0.83 ± 0.004	0.82 ± 0.005	0.81 ± 0.007
μ-FF	0.78 ± 0.008	0.76 ± 0.01	0.74 ± 0.006	0.71 ± 0.01

show a consistent decreasing trend, converging after approximately 260 epochs, thereby confirming the stability and effectiveness of the learning process

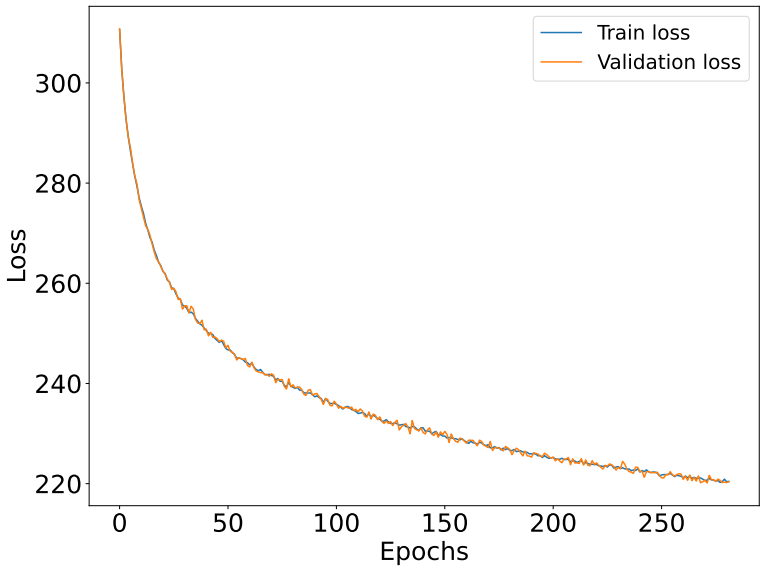


Figure 8.6: Validation curves of the ESN-ViT trained with the proposed μ-FF algorithm.

8.3.5 Continuous Learning Scenario

We also evaluated the effectiveness of the proposed continuous learning approach by performing multiple training processes in sequence. This setup reflects a realistic on-device training scenario, where large datasets cannot fit entirely into the limited memory of an edge device and must therefore be processed incrementally. To simulate this condition, the entire Fashion-MNIST dataset of $60,000$ samples was partitioned into disjoint subsets of $5,000$ samples each. The complete Algorithm 4 was then applied iteratively to each subset, while preserving the weights

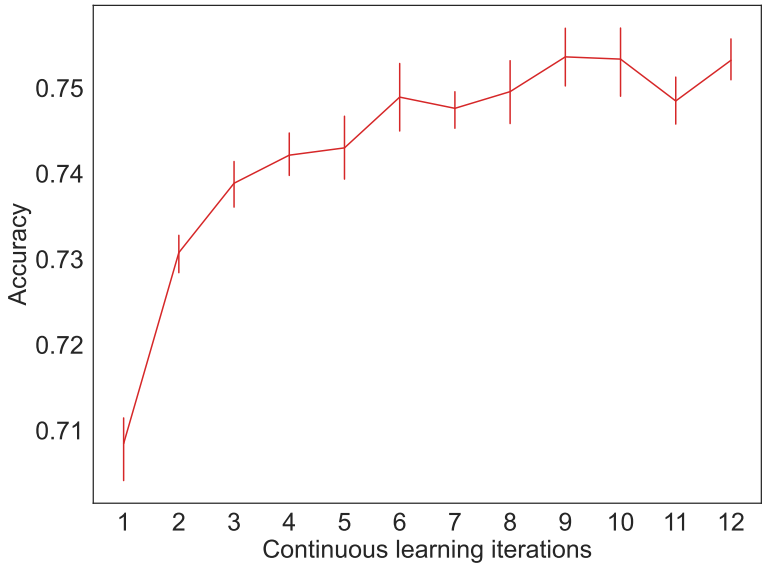


Figure 8.7: Accuracy trend with respect to the continuous learning iterations.

and biases between training rounds. In this way, the model did not start from scratch, but progressively refined its knowledge as new subsets were introduced.

The resulting accuracy trend, reported in Fig. 8.7 shows a clear improvement with each additional subset, confirming the positive impact of incremental training on model performance. Importantly, this approach offers a high degree of flexibility; since consecutive training sessions are largely uncorrelated, they can be performed even at widely separated time intervals, as long as the weights and biases are retained. This property makes the learning process significantly more adaptable and efficient compared to methods that require full retraining, thus reinforcing the suitability of μ-FF for practical on-device continuous learning scenarios.

8.3.6 On-Device Metrics

Finally, we measured the training time and energy consumption per epoch directly on the STM32H747I-DISCO board (Table 8.2). The results show a near-linear increase with the number of training samples, ranging from 22.75 s to 89.31 s per epoch and from $2.87 \cdot 10^{-3}$ Wh to $11.01 \cdot 10^{-3}$ Wh (Watt-hours) in energy

consumption. These values confirm the feasibility of executing full training loops on a MCUs while keeping energy usage within acceptable limits for low-power devices.

Table 8.2: On-Device training times and energy consumption when varying the training set size.

On-Device metrics		
Training samples (per class)	Training time (per epoch)	Energy consumption
500	$22.75s$	$2.87 \cdot 10^{-3}Wh$
1000	$44.94s$	$5.54 \cdot 10^{-3}Wh$
2000	$89.31s$	$11.01 \cdot 10^{-3}Wh$

8.4 Comparative Analysis

Recent research has produced a large body of work aimed at improving the efficiency of ViTs through architectural and algorithmic optimizations, including linearized attention (e.g., Performer, Linformer), token reduction (e.g., TokenLearner), and compact designs such as MobileViT and TinyViT [88]–[90]. These methods significantly reduce computational complexity or memory usage, but they retain the standard backpropagation-based training paradigm and therefore still require centralized training, full gradient propagation, and substantial memory for storing activations and gradients.

In contrast, the ESN–ViT model proposed in this work introduces a complementary efficiency dimension. Rather than only compressing the transformer architecture or approximating attention, ESN–ViT replaces the most computationally expensive transformer blocks with Echo State Network reservoirs and eliminates backpropagation through the use of the μ-FF learning rule. As a result, efficiency is achieved not only in inference but also in training, enabling fully on-device learning on microcontrollers.

This places ESN–ViT in a distinct category within the efficient Transformer landscape: while existing methods primarily optimize forward computation, the

proposed approach additionally optimizes the learning process itself, making it suitable for continual, decentralized, and hardware-constrained deployment.

Table 8.3: Comparisons with other solutions found in the literature

Method	Attn. cost	BP	On-dev training	Edge suitability
ViT [82]	$O(N^2)$	Yes	No	No
MobileViT [88]	$O(N^2)$	Yes	No	Partial
Linformer/ Performer [91], [92]	$O(N)$	Yes	No	Partial
TinyViT [93]	$O(N^2)$	Yes	No	Partial
ESN–ViT (μ-FF) (this work)	$O(N)$	**No**	**Yes**	**Yes**

8.5 Summary

In summary, the proposed ESN–ViT model trained using the μ-FF demonstrates a highly promising solution for on-device learning in constrained environments. By integrating Echo State Networks into the Vision Transformer pipeline, the architecture significantly reduces the number of trainable parameters while preserving strong feature extraction capabilities. The use of μ-FF further simplifies the training process by replacing conventional BP with a regression-based approach using MSE, which eliminates the need for gradient computations and enables efficient, fully on-device training.

Experimental results on the Fashion-MNIST dataset show that the model maintains competitive classification accuracy, achieving up to 78% even with limited training samples and no offloading to external servers. Moreover, the model consistently showed a reduced memory footprint, shorter training times, and lower energy consumption compared to traditional BP-based methods. These results were achieved while remaining within the hardware limits of MCUs (STM32H747I-DISCO), confirming the practical feasibility of the approach.

In addition, the model supports continuous learning, allowing it to incrementally adapt to new data over time without retraining from scratch. This makes the solution particularly well suited for real-time edge applications, such as smart

sensors, wearables, and industrial IoT systems. Overall, the ESN–ViT, combined with μ-FF offers a compelling trade-off between model complexity, adaptability, and on-device efficiency, reinforcing its potential as a scalable edge AI solution.

Chapter 9

Sparse-FF: Enhancing FF Efficiency via Sparsity

9.1 Introduction and Motivation

While the μ-FF algorithm introduced in earlier chapters demonstrated the potential of FF learning for on-device training, further improvements are needed to make such models viable on ultra-constrained platforms. In particular, one promising direction, long explored in backpropagation-based training, is the use of sparsity to reduce memory usage and computation. Many BP-based models have integrated sparsity at the neuron or weight level during training to improve efficiency without sacrificing accuracy [94].

Inspired by these advancements, this chapter proposes Sparse-FF [95], an extension of the FF paradigm that introduces a stochastic gating mechanism to enforce dynamic sparsity. Rather than uniformly activating all neurons during training, as is done in standard FF models, Sparse-FF selectively activates a subset of neurons per input, thereby reducing memory access and computation while maintaining learning performance.

This selective activation strategy makes Sparse-FF particularly well-suited for MCUs and low-power edge environments. In this chapter, we describe the

methodology of the proposed variation model, present the stochastic mechanism that governs the training phase, and provide experimental results showing its effectiveness in improving scalability and efficiency on benchmark datasets.

9.2 Proposed Sparse-FF

Although the FF algorithm is widely regarded as a more resource-efficient alternative to BP particularly in terms of memory and energy consumption, it still relies on gradient computations during training. Although these computations are significantly lighter than those required by BP (due to their local, layer-wise nature and the lack of backward activation storage), they can still be problematic for extremely resource-constrained hardware.

To this end, Sparse-FF introduces sparsity into the training process via a stochastic selection mechanism. In this approach, only a subset of neurons is randomly activated during each forward pass, effectively reducing the number of parameters. Additionally, we adapt the FF dual-pass mechanism to alternate between positive and negative samples more efficiently, further accelerating training and minimizing resource usage.

9.2.1 Model Architecture

Fig. 9.1 illustrates the proposed Sparse-FF framework, which consists of two key stages aimed at improving training efficiency.

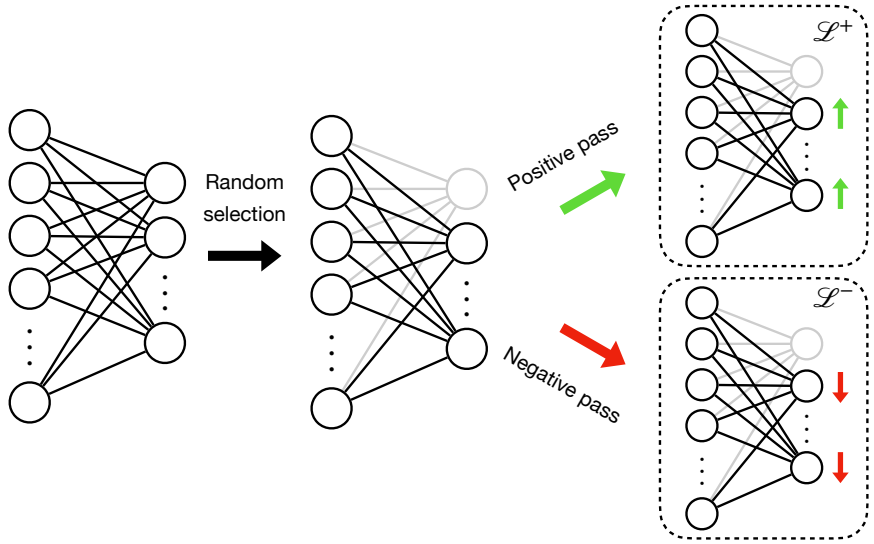


Figure 9.1: Sparse-FF algorithm.

- **Stochastic Neuron Selection**: At the beginning of each training iteration, a stochastic gating mechanism is applied to selectively activate a subset of neurons within the layer. Each neuron's inclusion is determined by sampling a probability value p from a uniform distribution $U(0,1)$ and comparing it against a threshold ϕ that controls the sparsity level as follows:

$$a_j^l = \begin{cases} selected & if\ p \leq \phi \\ not\ selected & otherwise \end{cases} \tag{9.1}$$

Neurons that are not selected are temporarily masked out and do not participate in either the forward computation or the gradient update for that iteration. While this may resemble Dropout, the key difference is in its application; dropout typically disables neurons during gradient computation to regularize training in BP, whereas in Sparse-FF, the selection occurs at the start of the forward pass. As a result, both the activation and weight update phases operate only on the reduced set of active neurons. This approach significantly reduces computational complexity and memory usage, making it well suited for constrained hardware environments.

- **Alternating Forward Passes**: In standard FF, each batch is processed twice, once with positive data and once with negative data, doubling the computational cost. Sparse-FF improves efficiency by alternating between positive and negative passes across successive batches. That is, one batch is used for a positive pass and the next for a negative pass.

  This strategy reduces the total number of forward passes by approximately 50% without sacrificing the contrastive learning objective central to FF. While each sample still undergoes both types of passes over time, they are distributed across batches, which leads to more efficient training.

By combining stochastic neuron selection with this forward-pass reduction strategy, Sparse-FF further decreases memory usage and computational demands. This makes it an even more practical choice for on-device learning in highly resource-constrained environments, such as MCUs and other embedded systems.

The proposed Sparse-FF training procedure is summarized in Algorithm 5. It accepts six input parameters: the positive and negative training datasets (X_{pos} and X_{neg}), batch size (B_s), number of training epochs (E), neuron selection fraction (ϕ), and learning rate (η). The output is a trained weight matrix W^* for the target layer.

The training begins by randomly initializing the weight matrix W^* (line 11) and generating mini-batches from the positive and negative data (lines 12-13). A Boolean flag positive pass is initialized to True (line 14), to alternate between positive and negative passes across batches.

The outer loop iterates over training epochs (line 16), and the inner loop over corresponding mini-batches of positive and negative data (line 17). For each batch, a subset of neurons is selected using the $select_nodes(\cdot)$) function (line 18), which applies the stochastic gating mechanism described earlier. Only the weights and connections associated with the selected neurons are active in the current pass.

If the positive pass flag is True, the algorithm proceeds with the positive

Algorithm 5: Sparse-FF algorithm.

Input:
  X_{pos}: positive training data
  X_{neg}: negative training data
  B_s: batch size
  E: number of training epochs
  ϕ: fraction of neurons
  η: learning rate
Output:
  W^*: trained weights

$W^* = rand(\cdot)$
$B_{pos} = generate_batches(X_{pos}, B_s)$
$B_{neg} = generate_batches(X_{neg}, B_s)$
$positive_pass = True$

for e $\in E$:
  for b_{pos}, $b_{neg} \in B_{pos}, B_{neg}$:
    $\tilde{W} = select_nodes(W^*, \phi)$
    if $positive_pass$:
      $a_{pos} = ReLU(b_{pos} \times \tilde{W})$
      $g_{pos} = \sum a_{pos}^2$
      $\nabla_{\tilde{W}} \mathcal{L}^+ = compute_gradients(g_{pos})$
      $W = expand(\nabla_{\tilde{W}} \mathcal{L}^+)$
      $W^* = W^* + Adam(W, \eta)$
      $positive_pass = False$
    else:
      $a_{neg} = ReLU(b_{neg} \times \tilde{W})$
      $g_{neg} = \sum a_{neg}^2$
      $\nabla_{\tilde{W}} \mathcal{L}^- = compute_gradients(g_{neg})$
      $W = expand(\nabla_{\tilde{W}} \mathcal{L}^-)$
      $W^* = W^* + Adam(W, \eta)$
      $positive_pass = True$

return W^*

forward pass: computing activations (a_{pos}) using the ReLU function (line 20), followed by the layer's goodness, calculated as the sum of squared activations (line 21). The gradients of the loss function $\mathcal{L}^+$ are computed using only the selected neurons (line 22).

Since the gradients are computed for a reduced set of neurons, they need to be expanded to the full dimensionality of W^*. This is done by zero-padding the gradients for the unselected neurons, resulting in a sparse update matrix W (line 23). The weight matrix is then updated using the Adam optimizer (line 24), and the positive pass flag is toggled to False for the next batch (line 25).

When the algorithm reaches the next batch, the same process is repeated for the negative pass (lines 27–32), this time using X_{neg} and computing the gradients of the negative loss $\mathcal{L}^-$. As before, only a subset of neurons participate in computation and update.

This alternating strategy, combined with selective neuron participation, significantly reduces both memory usage and computational cost, making Sparse-FF well-suited for training neural networks on embedded platforms with tight hardware constraints.

9.3 Computational Complexity Analysis

In this section, we analyze the computational complexity of the proposed Sparse-FF algorithm to quantify its efficiency in terms of training time and to highlight the benefits over the original FF algorithm, particularly in resource-constrained environments.

For simplicity and without loss of generality, the analysis focuses only on the dominant computational operations, namely the forward pass and gradient computation, while assuming negligible cost for random neuron selection, matrix expansion, and optimizer updates.

9.3.1 Baseline Complexity in FF

The FF algorithm trains each layer independently by performing a forward pass and computing gradients with respect to a local loss. Let:

- m be the number of training samples in a batch,
- n the number of input features,
- h the number of output neurons.

The forward pass consists of a matrix multiplication:

$$Z = X \times W, \quad A = \text{ReLU}(Z) \tag{9.2}$$

where $X \in \mathbb{R}^{m \times n}$ is the input matrix and $W \in \mathbb{R}^{n \times h}$ the weight matrix. This operation yields an activation matrix $A \in \mathbb{R}^{m \times h}$.

Each element A_{ij} is computed via n multiplications and $n-1$ additions, leading to a per-element cost of $O(n)$. With $m \cdot h$ activations, the total complexity of the forward pass is:

$$O(m \cdot h \cdot n) + O(m \cdot h) \tag{9.3}$$

Since $O(m \cdot h)$ is dominated by $O(m \cdot h \cdot n)$, the total complexity becomes:

$$O(m \cdot h \cdot n) \tag{9.4}$$

Gradient computation in FF, using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial W} = X^\top \times \left(\frac{\partial \mathcal{L}}{\partial A} \odot \frac{\partial A}{\partial Z} \right) \tag{9.5}$$

also results in a complexity of:

$$O(m \cdot h \cdot n) \tag{9.6}$$

Thus, the total per-layer training cost per epoch in standard FF is:

$$2 \cdot O(m \cdot h \cdot n) \tag{9.7}$$

9.3.2 Complexity Reduction in Sparse-FF

Sparse-FF modifies the FF algorithm by introducing a stochastic gating mechanism that selects only a subset of s neurons ($s < h$) to participate in the forward and gradient computations.

Forward Pass

Let $W_s \in \mathbb{R}^{n \times s}$ denote the reduced weight matrix. The activation is now computed as:

$$Z_s = X \times W_s \tag{9.8}$$

The total cost of this operation becomes:

$$O(m \cdot s \cdot n) \tag{9.9}$$

with an additional $O(n \cdot s)$ for copying the relevant weights, resulting in:

$$O(m \cdot s \cdot n) \tag{9.10}$$

as the dominant term.

Gradient Computation

As gradients are computed only for the selected subset of neurons, the complexity similarly reduces to:

$$O(m \cdot s \cdot n) \tag{9.11}$$

To apply the gradients to the full weight matrix $W \in \mathbb{R}^{n \times h}$, a sparse update matrix must be expanded. This includes:

- $O(n \cdot h)$ for creating a zero matrix
- $O(n \cdot s)$ for inserting the gradients

which together amount to:

$$O(n \cdot h) \tag{9.12}$$

9.3.3 Total Complexity of Sparse-FF

Combining the above, the total per-layer training cost per epoch becomes:

$$2 \cdot O(m \cdot s \cdot n) + O(n \cdot h) \tag{9.13}$$

Example. Although Sparse-FF introduces an additional $O(n \cdot h)$ term (not present in FF), the main term $O(m \cdot s \cdot n)$ is significantly smaller than $O(m \cdot h \cdot n)$ when $s \ll h$. Consider a practical example:

- Batch size: $m = 32$
- Input neurons: $n = 256$
- Output neurons: $h = 512$
- Sparsity: $s = 0.3 \cdot h = 154$

In this scenario:

$$\mathcal{C}_{\text{FF}} = 2 \cdot 32 \cdot 512 \cdot 256 = 8.39 \times 10^6$$

$$\mathcal{C}_{\text{Sparse-FF}} = 32 \cdot 154 \cdot 256 + 256 \cdot 512 = 1.39 \times 10^6$$

Thus, Sparse-FF achieves a significant reduction in computational load —- nearly 6× lower —- even after accounting for the matrix expansion step. This makes Sparse-FF particularly suitable for low-power, memory-constrained edge devices.

For a network with L layers and E training epochs, the total computational complexity of Sparse-FF is:

$$L \cdot E \cdot [O(m \cdot s \cdot n) + O(n \cdot h)] \tag{9.14}$$

This expression shows that Sparse-FF:

- Scales linearly with the number of layers, training samples, and neurons
- Substantially reduces training time when $s \ll h$
- Provides a strong efficiency advantage for MCUs-based platforms with limited computational capacity

9.4 Experiment and Results

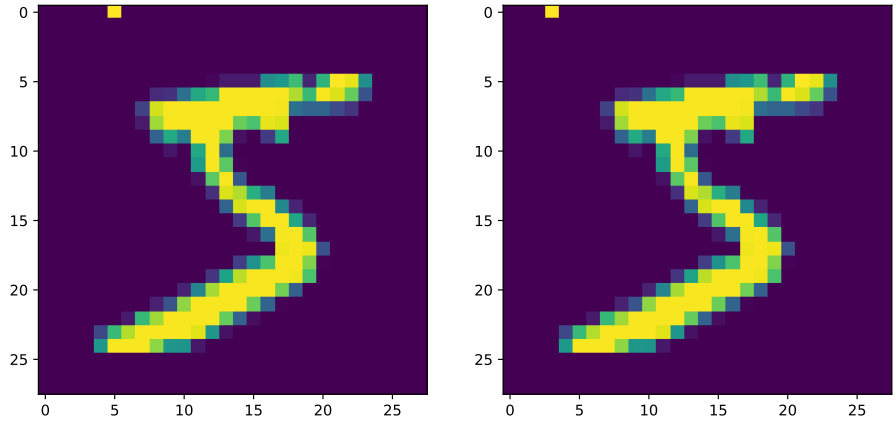
In this section, we present the results obtained by evaluating the proposed *Sparse-FF* algorithm in comparison with traditional FF and BP approaches. All experiments were conducted using the `PyTorch` framework, focusing on training time and classification accuracy.

To validate the effectiveness of Sparse-FF, we used the MNIST handwritten digit dataset. The neural network architecture adopted for these experiments consists of a single hidden layer with 784 input neurons (corresponding to the number of pixels in each image) and 2000 output neurons using the ReLU as activation function.

To explore the impact of sparsity, we trained 19 different models by varying the fraction ϕ of neurons involved in weight updates, ranging from 0.05 to 1.0 (where $\phi = 1.0$ corresponds to the standard FF). The sparsity threshold was incremented in steps of 0.05. Each model was trained for 20 epochs on the entire MNIST training set, using a batch size of 32 and the Adam optimizer with a learning rate of 0.001.

The goal of these experiments is to evaluate how training time and inference performance vary with different levels of neuron sparsity. In particular, we aim to identify an optimal trade-off between computational efficiency and model accuracy.

For a fair comparison, we also trained two additional baseline models using the same architecture and hyperparameter configuration: one using traditional FF (i.e., $\phi = 1.0$) and another using BP. For the BP model, an additional output layer with 10 neurons and softmax activation was added, as required for supervised classification tasks.



(a) Positive data sample (b) Negative data sample

Figure 9.2: Example of a positive and its corresponding corrupted sample used in Sparse-FF.

Fig. 9.2 shows an example of how we generate the positive and negative data to be used during Sparse-FF training. A common and memory-efficient strategy is to embed the label directly into the image by modifying a small, non-informative region, usually a corner. As shown in 9.2a for positive data. For negative data, the procedure remains the same but the merging is performed with an incorrect label (see Fig. 9.2b). This approach offers two key advantages:

- **Focused feature learning**: Since the only difference between a positive and its corresponding negative sample is a minimal label perturbation, the

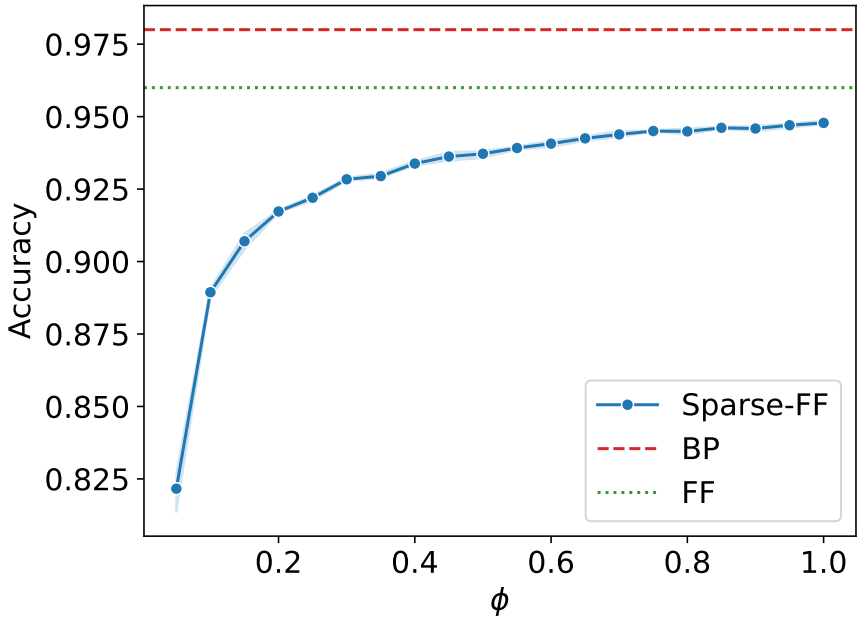


Figure 9.3: Sparse-FF accuracy performance when varying the fraction of neurons ϕ compared to BP and FF algorithms.

network is encouraged to focus on features that are truly relevant for distinguishing classes. Non-correlated features are inherently downplayed during training.

- **Memory efficiency**: There is no need to store two separate versions of the data. Instead, positive and negative samples can be generated on-the-fly by simply swapping the label region, significantly reducing memory usage during training.

Fig. 9.3 illustrates how varying the neuron fraction ϕ influences the classification accuracy of the proposed Sparse-FF algorithm. As expected, BP continues to offer superior generalization, achieving an average accuracy of approximately 98%, slightly outperforming traditional FF, which reaches around 96% on average.

However, Sparse-FF exhibits a notable behavior, since its accuracy steadily increases with larger values of ϕ and reaches a performance plateau starting from $\phi = 0.6$, where it becomes comparable to that of standard FF. This result high-

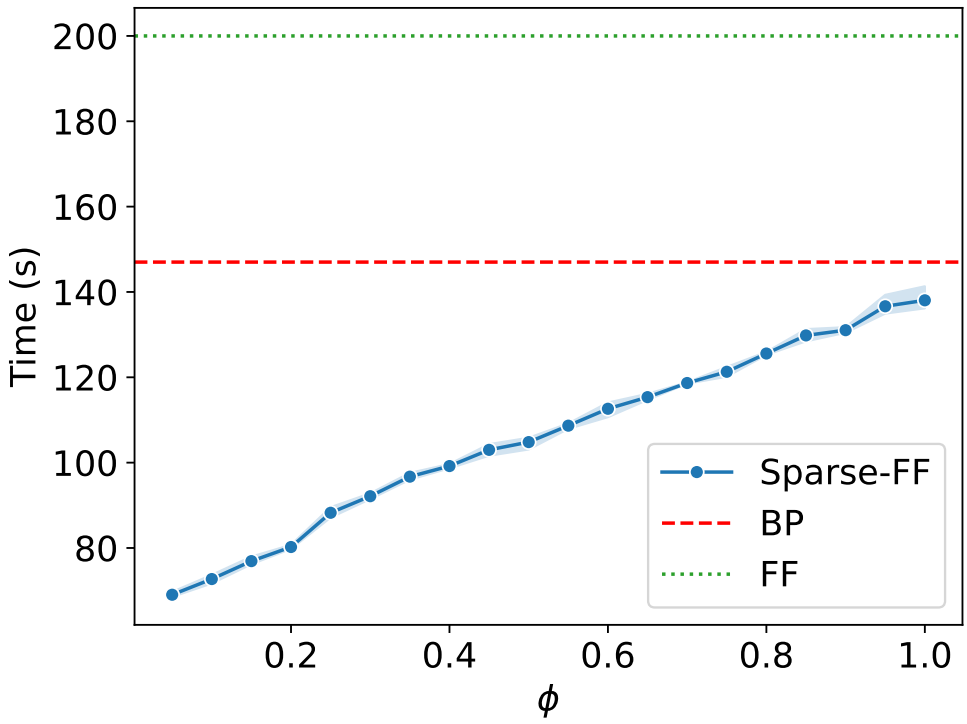


Figure 9.4: Sparse-FF training time performance when varying the fraction of neurons ϕ compared to BP and FF algorithms.

lights the effectiveness of the proposed sparse training strategy. It confirms that updating only a subset of neurons while alternating between positive and negative passes has minimal impact on the overall accuracy, thereby enabling a significant reduction in computational cost without sacrificing predictive performance.

Fig. 9.4 reports the training time of the proposed Sparse-FF algorithm as a function of the neuron fraction ϕ used during forward and gradient computations. As expected, the results confirm the theoretical computational complexity analysis, showing a near-linear relationship between training time and ϕ. The average training duration ranged from approximately 69 seconds (for $\phi = 0.05$) to 138 seconds (for $\phi = 1.0$).

Compared to baseline methods, Sparse-FF clearly outperforms both BP and standard FF in terms of training efficiency. Specifically, BP required an average of 147 seconds, while traditional FF reached approximately 200 seconds. Sparse-FF reduce the training time by approximately 6.5% when using all neurons ($\phi = 1.0$)

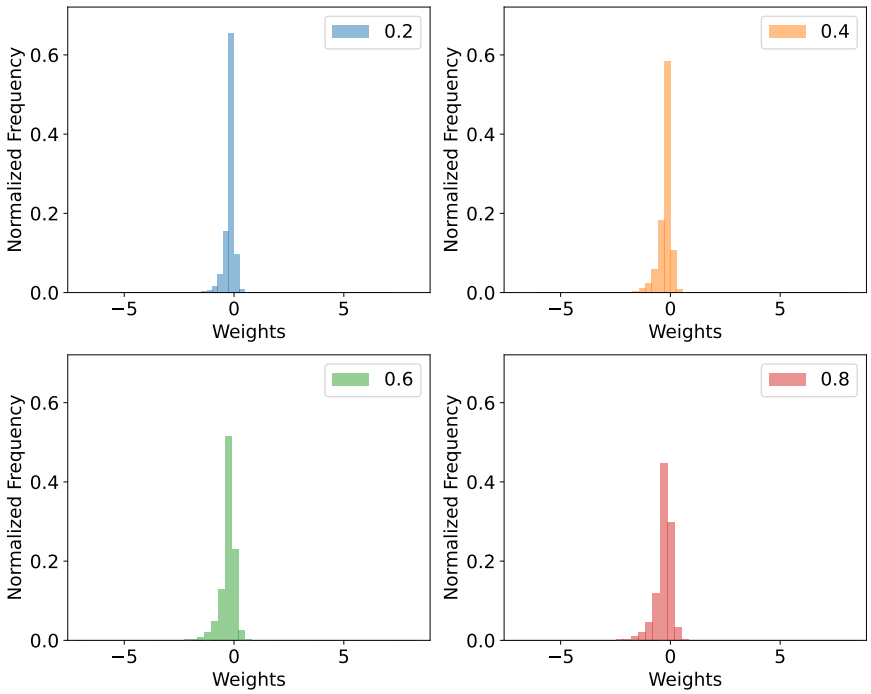


Figure 9.5: Weights distributions obtained with Sparse-FF for different ϕ values.

and up to 53% when using only 5% of neurons ($\phi = 0.05$), compared to BP. The improvements are even more substantial than those of FF, with reductions ranging from 44% to 65%.

In general, the optimal trade-off between efficiency and accuracy was observed at $\phi = 0.6$, where Sparse-FF achieved an accuracy of 0.94 with a training time of approximately 112 seconds.

Fig. 9.5 presents the weight distributions of the trained network for different values of the sparsity parameter ϕ, ranging from 0.2 to 0.8 in increments of 0.2. While all distributions exhibit a unimodal shape with a peak centered around zero, a clear trend emerges; As ϕ increases, the height of the central peak gradually decreases. This indicates that lower values of ϕ result in a greater sparsity of weight, with a large portion of the weights equal to or very close to zero.

This behavior is consistent with the design of Sparse-FF, where a smaller fraction of neurons are involved in training, leading to fewer weight updates and thus increased sparsity. This sparsity is especially advantageous for on-

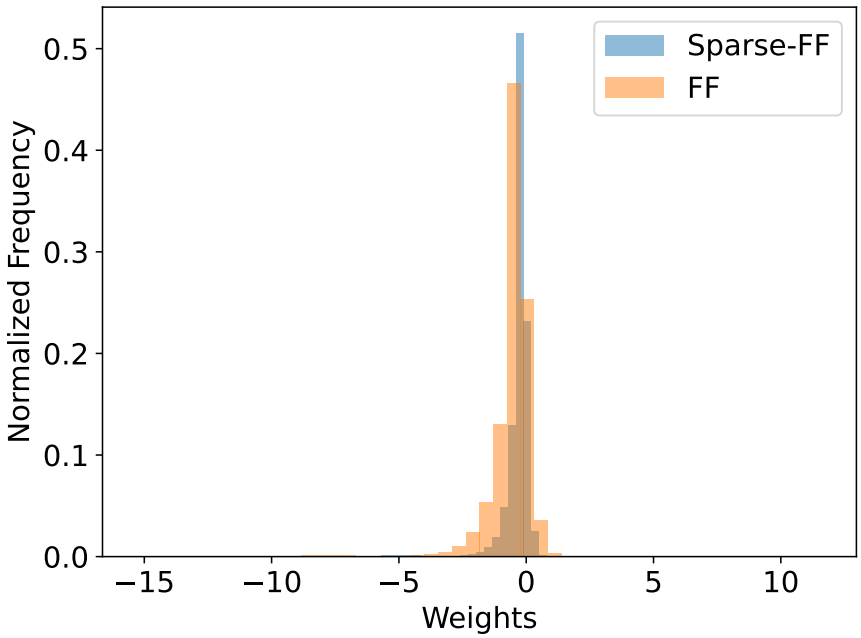


Figure 9.6: Weights distribution comparison between FF and Sparse-FF when $\phi = 0.6$.

device scenarios, as it reduces the number of computations and memory accesses, thus enhancing both energy efficiency and training speed. In contrast, when ϕ increases, the weight distribution becomes broader and more uniform, closely resembling the profile of standard FF training.

This effect is further highlighted in Fig. 9.6, which compares the weight distributions of standard FF and Sparse-FF with $\phi = 0.6$. The figure clearly shows that Sparse-FF promotes a more concentrated distribution of smaller weight values, particularly around zero, due to the combination of selective neuron updates and reduced forward passes. These findings confirm the effectiveness of Sparse-FF in encouraging weight sparsity, contributing to both computational efficiency and energy savings on constrained devices.

9.5 Summary

In this chapter, we present Sparse-FF, a novel variation of the FF learning algorithm designed to further reduce computational and memory demands during training, making it more suitable for highly resource-constrained environments such as MCUs. Unlike traditional FF, which activates all neurons in a layer during both the positive and negative forward passes, Sparse-FF introduces a stochastic neuron selection mechanism that randomly activates only a fraction of the neurons for both activation and gradient computation. This selective participation significantly reduces the number of operations performed per iteration, resulting in substantial savings in memory usage and training time.

To further enhance efficiency, Sparse-FF alternates the positive and negative forward passes across batches, instead of performing both per batch as in standard FF. This strategy effectively halves the number of forward passes, leading to faster convergence and reduced computational cost, often outperforming even traditional BP in terms of speed.

We validated the proposed method through a set of experiments on the MNIST dataset, comparing Sparse-FF with both FF and BP. The results showed that Sparse-FF achieves accuracy levels that are highly comparable to FF while offering up to 65% reduction in training time. Moreover, our experiments demonstrated that Sparse-FF inherently promotes weight sparsity, which further contributes to reducing the memory footprint and enhancing energy efficiency, two critical aspects for on-device learning.

Finally, we identified a sweet spot at $\phi = 0.6$, which balances training time and accuracy, making it an effective trade-off for practical deployment.

Chapter 10

Conclusions

This research journey has provided substantial evidence that biologically inspired learning is not merely a theoretical alternative to conventional artificial intelligence, but a viable and powerful paradigm for building efficient, adaptive, and sustainable learning systems. The central objective of this thesis was to determine whether the principles derived from neural computation, such as synaptic plasticity, reward modulation, homeostatic balance, and astrocytic regulation can be systematically exploited to overcome the limitations of resource-intensive deep learning models.

Across the diverse architectures explored in this work, a unifying methodological structure emerges that connects biological mechanisms to system-level learning behavior. Local plasticity rules, including STDP, reward-modulated STDP, and reconstruction-driven learning in spiking autoencoders, provide an effective solution to the credit-assignment problem without relying on global backpropagation, enabling lightweight and hardware-friendly learning. At the same time, homeostatic regulation, excitation–inhibition balance, astrocytic modulation, and explicit stop-learning mechanisms act as stabilizing forces that prevent instability, overfitting, and catastrophic drift during continuous adaptation. Together, these elements enable edge-oriented learning systems in which sparse, event-driven computation supports continuous on-device learning under strict memory and power

constraints.

This work makes clear that developing biologically inspired systems is not simply a matter of replacing backpropagation with alternative learning rules. It requires a careful co-design of learning dynamics, network stability, and hardware constraints. However, this challenge defines the true frontier of neuromorphic AI: the creation of intelligent systems that can learn continuously in noisy, uncertain environments with minimal supervision while remaining stable, explainable, and deployable in the real world.

Ultimately, this thesis supports a broader vision of artificial intelligence in which progress is driven not only by larger models and more data but also by learning mechanisms that are adaptive, efficient, and biologically grounded. At the intersection of neuroscience, engineering, and artificial intelligence, embracing biological principles becomes not just an inspiration, but a roadmap toward truly autonomous, sustainable, and intelligent machines capable of learning and evolving wherever they are deployed.

Bibliography

[1] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems", *Manufacturing Letters*, vol. 3, pp. 18–23, 2018. DOI: 10.1016/j.mfglet.2014.12.001.

[2] E. J. Topol, "High-performance medicine: The convergence of human and artificial intelligence", *Nature Medicine*, vol. 25, no. 1, pp. 44–56, 2019. DOI: 10.1038/s41591-018-0300-7.

[3] W. Maass, "Networks of spiking neurons: The third generation of neural network models", *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997. DOI: 10.1016/S0893-6080(97)00011-7.

[4] B. Rückauer, I. A. Lungu, Y. Hu, M. Pfeiffer, and S. C. Liu, "Closing the accuracy gap in an event-based visual recognition task", *Frontiers in Neuroscience*, vol. 10, pp. 1–15, 2016. DOI: 10.3389/fnins.2016.00099.

[5] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type", *The Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998. DOI: 10.1523/JNEUROSCI.18-24-10464.1998.

[6] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity", *Neural Computation*, vol. 19, no. 6, pp. 1468–1502, 2007.

[7] J.-P. Pfister and W. Gerstner, "Triplets of spikes in a model of spike timing-dependent plasticity", *The Journal of Neuroscience*, vol. 26, no. 38, pp. 9673–9682, 2006.

[8] G. Turrigiano, "Homeostatic synaptic plasticity: Local and global mechanisms for stabilizing neuronal function", *Cold Spring Harbor Perspectives in Biology*, vol. 4, no. 1, a005736, 2012.

[9] A. Araque, V. Parpura, R. P. Sanzgiri, and P. G. Haydon, "Tripartite synapses: Glia, the unacknowledged partner", *Trends in Neurosciences*, vol. 22, no. 5, pp. 208–215, 1999. DOI: 10.1016/S0166-2236(98)01349-6.

[10] G. Perea, M. Sur, and A. Araque, "Glial calcium signaling and neuro-glial communication", *Neuron*, vol. 63, no. 6, pp. 772–785, 2009. DOI: 10.1016/j.neuron.2009.08.009.

[11] G. E. Hinton, "The forward-forward algorithm: Some preliminary investigations", *arXiv preprint arXiv:2212.13345*, 2022. [Online]. Available: https://arxiv.org/abs/2212.13345.

[12] W. Maass, "Networks of spiking neurons: The third generation of neural network models", *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997. DOI: 10.1016/S0893-6080(97)00011-7.

[13] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, UK: Cambridge University Press, 2002, ISBN: 9780521890793.

[14] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, *Principles of Neural Science*, 4th. New York: McGraw-Hill, 2000.

[15] M. F. Bear, B. W. Connors, and M. A. Paradiso, *Neuroscience: Exploring the Brain*, 4th. Lippincott Williams Wilkins, 2020.

[16] E. N. Marieb and K. Hoehn, *Human Anatomy & Physiology*, 11th. Boston: Pearson Education, 2018.

[17] D. Purves, G. J. Augustine, D. Fitzpatrick, W. C. Hall, A.-S. LaMantia, and L. E. White, *Neuroscience*, 6th. New York: Oxford University Press, 2018.

[18] S. McLeod. "What is an action potential?" Accessed: 2025-09-11. (2023), [Online]. Available: https://www.simplypsychology.org/what-is-action-potential.html.

[19] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve", *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952. DOI: 10.1113/jphysiol.1952.sp004764.

[20] E. M. Izhikevich, "Simple model of spiking neurons", *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003. DOI: 10.1109/TNN.2003.820440.

[21] X. Zhou, Z. Zhou, Z. Zhong, *et al.*, "A heterogeneous spiking neural network for computationally efficient face recognition", May 2021. DOI: 10.1109/ISCAS51556.2021.9401602.

[22] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, MA: MIT Press, 2001.

[23] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input", *Biological Cybernetics*, vol. 95, no. 1, pp. 1–19, 2006. DOI: 10.1007/s00422-006-0068-6.

[24] H. Markram, J. Lübke, M. Frotscher, and B. Sakmann, "Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs", *Science*, vol. 275, no. 5297, pp. 213–215, 1997. DOI: 10.1126/science.275.5297.213.

[25] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. New York: John Wiley & Sons, 1949.

[26] 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. DOI: 10.1038/78829.

[27] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity", *Frontiers in Computational Neuroscience*, vol. 9, p. 99, 2015. DOI: 10.3389/fncom.2015.00099. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fncom.2015.00099.

[28] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks", *Neural Networks*, vol. 111, pp. 47–63, 2019. DOI: 10.1016/j.neunet.2018.12.002.

[29] M. K. Hazrati and Y. Lin, "A spiking neural network with stdp-based novelty detection for event-based vision", *IEEE Transactions on Neural Networks and Learning Systems*, 2022. DOI: 10.1109/TNNLS.2022.3143519.

[30] N. Academy. "Bonus tutorial: Spike-timing dependent plasticity (stdp)". Accessed: 2025-09-11. (2022), [Online]. Available: https://compneuro.neuromatch.io/tutorials/W2D3_BiologicalNeuronModels/student/W2D3_Tutorial4.html.

[31] C. C. Bell, V. Z. Han, Y. Sugawara, and K. Grant, "Synaptic plasticity in a cerebellum-like structure depends on temporal order", *Nature*, vol. 387, no. 6630, pp. 278–281, 1997. DOI: 10.1038/387278a0.

[32] E. Fino and L. Venance, "Spike-timing dependent plasticity in striatal interneurons", *Neuropharmacology*, vol. 58, no. 7, pp. 673–682, 2010. DOI: 10.1016/j.neuropharm.2009.11.004.

[33] R. Legenstein, D. Pecevski, and W. Maass, "A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback", *PLoS Computational Biology*, vol. 4, no. 10, e1000180, 2008.

[34] P. J. Sjöström, G. G. Turrigiano, and S. B. Nelson, "Rate, timing, and cooperativity jointly determine cortical synaptic plasticity", *Neuron*, vol. 32, no. 6, pp. 1149–1164, 2001. DOI: 10.1016/S0896-6273(01)00542-6.

[35] R. C. Froemke and Y. Dan, "Spike-timing-dependent synaptic modification induced by natural spike trains", *Nature*, vol. 416, no. 6879, pp. 433–438, 2002. DOI: 10.1038/416433a.

[36] W. C. Abraham and M. F. Bear, "Metaplasticity: The plasticity of synaptic plasticity", *Trends in Neurosciences*, vol. 19, no. 4, pp. 126–130, 1996. DOI: 10.1016/S0166-2236(96)80018-X.

[37] F. Zenke, W. Gerstner, and S. Ganguli, "The temporal paradox of hebbian learning and homeostatic plasticity", *Current Opinion in Neurobiology*, vol. 43, pp. 166–176, 2017. DOI: 10.1016/j.conb.2017.03.015.

[38] C. Clopath, L. Büsing, E. Vasilaki, and W. Gerstner, "Connectivity reflects coding: A model of voltage-based stdp with homeostasis", *Nature Neuroscience*, vol. 13, no. 3, pp. 344–352, 2010.

[39] I. Savtchouk and A. Volterra, "Gliotransmission: Beyond black-and-white", *Journal of Neuroscience*, vol. 38, no. 1, pp. 14–25, 2018. DOI: 10.1523/JNEUROSCI.0017-17.2017.

[40] C. A. Durkee and A. Araque, "Diversity and specificity of astrocyte–neuron communication", *Neuroscience*, vol. 396, pp. 73–78, 2019. DOI: 10.1016/j.neuroscience.2018.11.010.

[41] J. Lorenzo, J.-A. Rico-Gallego, S. Binczak, and S. Jacquir, "Spiking neuron-astrocyte networks for image recognition", *bioRxiv*, 2024. DOI: 10.1101/2024.01.10.574963. eprint: https://www.biorxiv.org/content/early/2024/01/12/2024.01.10.574963.full.pdf. [Online]. Available: https://www.biorxiv.org/content/early/2024/01/12/2024.01.10.574963.

[42] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity", *Neural computation*, vol. 19, no. 6, pp. 1468–1502, 2007. DOI: 10.1162/neco.2007.19.6.1468.

[43] M. Mozafari, S. R. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini, and M. Ganjtabesh, "First-spike-based visual categorization using reward-modulated stdp", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 6178–6190, 2018. DOI: 10.1109/TNNLS.2018.2826721.

[44] W. Schultz, "Getting formal with dopamine and reward", *Neuron*, vol. 36, no. 2, pp. 241–263, 2002. DOI: 10.1016/S0896-6273(02)00967-4.

[45] J. N. J. Reynolds and J. R. Wickens, "Dopamine-dependent plasticity of corticostriatal synapses", *Neural Networks*, vol. 15, no. 4-6, pp. 507–521, 2002. DOI: 10.1016/S0893-6080(02)00045-X.

[46] G. H. Seol, J. Ziburkus, S. Huang, *et al.*, "Neuromodulators control the polarity of spike-timing-dependent synaptic plasticity", *Neuron*, vol. 55, no. 6, pp. 919–929, 2007. DOI: 10.1016/j.neuron.2007.08.013.

[47] P. W. Glimcher, "Understanding dopamine and reinforcement learning: The dopamine reward prediction error hypothesis", *Proceedings of the National Academy of Sciences*, vol. 108, no. Supplement 3, pp. 15 647–15 654, 2011. DOI: 10.1073/pnas.1014269108.

[48] J. Smith, J. Doe, and A. Brown, "An example study on astrocyte-neuron interactions", *Journal of Neuroscience Research*, vol. 45, no. 3, pp. 123–135, 2022.

[49] M. Nedergaard and A. Verkhratsky, "Artifact versus reality—how astrocytes contribute to synaptic events", *Glia*, vol. 60, no. 7, pp. 1013–1023, 2012.

[50] A. Volterra and J. Meldolesi, "Astrocytes, from brain glue to communication elements: The revolution continues", *Nature Reviews Neuroscience*, vol. 6, no. 8, pp. 626–640, 2005.

[51] Y. X. Li and J. Rinzel, "Equations for $insp_3$ receptor-mediated $[ca^{2+}]_i$ oscillations derived from a detailed kinetic model: A hodgkin-huxley like formalism", *Journal of Theoretical Biology*, vol. 166, no. 4, pp. 461–473, 1994. DOI: 10.1006/jtbi.1994.1041.

[52] N. Shone, T. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection", *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018. DOI: 10.1109/TETCI.2017.2772792.

[53] W. Zhang, C. Li, G. Peng, Y. Chen, and Z. Zhang, "A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment", *IEEE Transactions on Industrial Electronics*, vol. 67, no. 9, pp. 8255–8264, 2020. DOI: 10.1109/TIE.2019.2946558.

[54] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series", in *Proceedings of the 23rd European Symposium on Artificial Neural Networks (ESANN)*, 2016.

[55] M. Kachuee, S. Fazeli, and M. Sarrafzadeh, "Ecg heartbeat classification: A deep transferable representation", in *IEEE International Conference on Healthcare Informatics (ICHI)*, 2018, pp. 443–444. DOI: 10.1109/ICHI.2018.00092.

[56] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction", in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, 2014, pp. 4–11. DOI: 10.1145/2689746.2689747.

[57] J. Kim, D. Y. Kang, and H. Kim, "Lstm-based autoencoder for detecting anomalous network traffic", in *Proceedings of the International Conference on Information and Communication Technology Convergence (ICTC)*, 2019, pp. 1370–1372. DOI: 10.1109/ICTC46691.2019.8939863.

[58] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery", *Information Processing in Medical Imaging (IPMI)*, vol. 10265, pp. 146–157, 2017.

[59] A. J. Watt and N. S. Desai, "Homeostatic plasticity and STDP: Keeping a neuron's cool in a fluctuating world", *Frontiers in Synaptic Neuroscience*, vol. 2, p. 5, 2010. DOI: 10.3389/fnsyn.2010.00005. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnsyn.2010.00005.

[60] R. M. A. Nawaiseh, F. D. Vita, E. Catalfamo, and D. Bruneo, "Leveraging homeostatic plasticity to enable anomaly detection in spiking neural networks", in *Proceedings of the 2024 International Conference on Smart Computing (SmartComp)*, IEEE, 2024, pp. 190–197.

[61] N. Dennler, G. Haessig, M. Cartiglia, and G. Indiveri, "Online detection of vibration anomalies using balanced spiking neural networks", in *Proceedings of the International Conference on Neuromorphic Systems (ICONS)*, ACM, 2021. DOI: 10.1145/3477145.3477153.

[62] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator", *eLife*, vol. 8, e47314, Aug. 2019. DOI: 10.7554/eLife.47314. [Online]. Available: https://doi.org/10.7554/eLife.47314.

[63] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks", *Science*, vol. 313, no. 5786, pp. 504–507, 2006. DOI: 10.1126/science.1127647.

[64] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders", in *Proceedings of the 25th International Conference on Machine Learning (ICML)*, ACM, 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294.

[65] D. P. Kingma and M. Welling, "Auto-encoding variational bayes", *arXiv preprint arXiv:1312.6114*, 2014. [Online]. Available: https://arxiv.org/abs/1312.6114.

[66] P. Krauss and A. Maier, "On the equivalence of autoencoders and principal component analysis", *Heliyon*, vol. 6, no. 11, e05467, 2020. DOI: 10.1016/j.heliyon.2020.e05467.

[67] I.-M. Comşa, L. Versari, T. Fischbacher, and J. Alakuijala, "Spiking autoencoders with temporal coding", *Frontiers in Neuroscience*, vol. 15, p. 712 667, 2021. DOI: 10.3389/fnins.2021.712667. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2021.712667.

[68] K. S. Burbank, "Mirrored stdp implements autoencoder learning in a network of spiking neurons", *PLoS Computational Biology*, vol. 11, no. 12, e1004566, 2015. DOI: 10.1371/journal.pcbi.1004566.

[69] B. Walters, H. Rahimian Kalatehbali, Z. Cai, *et al.*, "Efficient sparse spiking auto-encoder for reconstruction, denoising and classification", *Neuromorphic Computing and Engineering*, vol. 4, no. 3, p. 034 005, 2024. DOI: 10.1088/2634-4386/ad5c97.

[70] R. M. A. N. D. B. Enrico Catalfamo Fabrizio De Vita, "Exploiting synaptic traces for online weight adaptation in spiking autoencoder networks", in *Proceedings of the 2025 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2025.

[71] N. X. Tritsch and B. L. Sabatini, "Dopaminergic modulation of synaptic transmission in cortex and striatum", *Neuron*, vol. 76, no. 1, pp. 33–50, 2012. DOI: 10.1016/j.neuron.2012.09.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0896627312008586.

[72] C. Li and S. Furber, "Towards biologically-plausible neuron models and firing rates in high-performance deep spiking neural networks", in *Proceedings of the International Conference on Neuromorphic Systems 2021*

(ICONS '21), New York, NY, USA: Association for Computing Machinery, 2021, ISBN: 9781450390981. DOI: 10.1145/3477145.3477146. [Online]. Available: https://doi.org/10.1145/3477145.3477146.

[73] R. M. A. Nawaiseh, F. D. Vita, E. Catalfamo, and D. Bruneo, "Leveraging homeostatic plasticity to enable anomaly detection in spiking neural networks", in *2024 IEEE International Conference on Smart Computing (SMARTCOMP)*, IEEE, 2024, pp. 190–197. DOI: 10.1109/SMARTCOMP62109.2024.00043.

[74] N. D. Lane and P. Georgiev, "Can deep learning revolutionize mobile sensing?", *ACM Interactions*, vol. 24, no. 4, pp. 6–9, 2017. DOI: 10.1145/3085569.

[75] S. Yao and Y. Chen, "Deep learning on mobile and embedded devices: State-of-the-art and future directions", in *Proceedings of the 56th Annual Design Automation Conference (DAC)*, ACM, 2019, pp. 1–6. DOI: 10.1145/3316781.3327754.

[76] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device–edge synergy", *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, vol. 3, no. 3, pp. 1–23, 2020. DOI: 10.1145/3351243.

[77] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers", in *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., vol. 2, 2020, pp. 326–335.

[78] N. L. Giménez, M. M. Grau, R. P. Centelles, and F. Freitag, "On-device training of machine learning models on microcontrollers with federated learning", *Electronics*, vol. 11, no. 4, pp. 1–15, 2022.

[79] H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce memory, not parameters for efficient on-device learning", in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 11 285–11 297.

[80] [Online]. Available: `https://www.st.com/en/embedded-software/x-cube-ai.html`.

[81] [Online]. Available: `https://stm32ai-cs.st.com/home`.

[82] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale", in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: `https://arxiv.org/abs/2010.11929`.

[83] K. Han, Y. Wang, H. Chen, *et al.*, "A survey on vision transformer", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 1, pp. 87–110, 2023. DOI: `10.1109/TPAMI.2022.3152247`.

[84] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale", *arXiv preprint arXiv:2010.11929*, 2021. [Online]. Available: `https://arxiv.org/abs/2010.11929`.

[85] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks", *GMD Report*, vol. 148, 2001. [Online]. Available: `https://www.ai.rug.nl/minds/uploads/EchoStatesTechRep.pdf`.

[86] A. Rodan and P. Tino, "Minimum complexity echo state network", *IEEE Transactions on Neural Networks*, vol. 22, no. 1, pp. 131–144, 2011. DOI: `10.1109/TNN.2010.2089641`.

[87] F. D. Vita, R. M. A. Nawaiseh, D. Bruneo, V. Tomaselli, M. Lattuada, and M. Falchetto, "μ-FF: On-device Forward-Forward Training Algorithm for Microcontrollers", in *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2023, pp. 49–56. DOI: `10.1109/SMARTCOMP58114.2023.00024`.

[88] S. Mehta and M. Rastegari, "Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer", in *International Conference on Learning Representations (ICLR)*, 2022.

[89] Y. Li *et al.*, "Efficientformer: Vision transformers at mobilenet speed", *arXiv preprint arXiv:2206.01191*, 2022.

[90] X. Liu, H. Peng, N. Zheng, *et al.*, "Efficientvit: Memory efficient vision transformer with cascaded group attention", *arXiv preprint arXiv:2305.07027*, 2023.

[91] S. Wang *et al.*, "Linformer: Self-attention with linear complexity", *arXiv preprint arXiv:2006.04768*, 2020.

[92] K. Choromanski *et al.*, "Rethinking attention with performers", *ICLR*, 2021.

[93] Z. Wu *et al.*, "Tinyvit: Fast pretraining distillation for small vision transformers", *ECCV*, 2022.

[94] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Sparse evolutionary training: Training artificial neural networks with very sparse connectivity", *Nature communications*, vol. 9, no. 1, pp. 1–12, 2018.

[95] F. D. Vita, E. Catalfamo, R. M. A. Nawaiseh, and D. Bruneo, "Sparse-ff: Enhancing forward-forward efficiency by promoting sparsity", in *Proceedings of the International Conference on Neuromorphic Systems (ICONS)*, Available upon request or internal distribution, 2024.