ENHANCING NEUROMORPHIC COMPUTING WITH ADVANCED SPIKING NEURAL NETWORK ARCHITECTURES

MEJORA DE COMPUTACION NEUROMORFICA CON ARQUITECTURAS AVANZADAS DE REDES NEURONALES POR IMPULSOS

This dissertation proposal is submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

PAOLO G. CACHI

Bachelor of Science, University Nacional San Antonio Abad del Cusco, Peru, 2012

Master of Science, Pontifícia Universidade Católica do Rio de Janeiro, Brazil, 2015

Director: Krzysztof J. Cios, Professor,

Department of Computer Science, Virginia Commonwealth University

Co-Director: Sebastian Ventura, Professor,

Departmento Computación avanzada, energía y plasmas, Universidad de Cordoba

Virginia Commonwealth University

Richmond, Virginia

June, 2023

Acknowledgements

I would like to express my deepest gratitude to my advisor and chair of my committee, Dr. Krzysztof J. Cios, for his invaluable patience and feedback. I am also very grateful to my dissertation committee, Dr. Ventura, co-advisor, Dr. Arodz, Dr. Damevski, Dr. Luna, Dr. Zafra and Dr. García, for their time and provided knowledge and expertise. I also appreciate financial support by the Virginia Commonwealth University. I am grateful to my family and friends for their immense support during my academic journey.

TABLE OF CONTENTS

Chapter Page

Acknowledgements . . . iii

Table of Contents . . . iv

List of Tables . . . vi

List of Figures . . . viii

Resumen . . . xi

Abstract . . . xvi

1 Introduction . . . 1

1.1 Overview . . . 1
1.2 Objectives . . . 4
1.3 Organization . . . 5

2 Literature Review . . . 6

2.1 Artificial Neural Networks . . . 6
2.2 Spiking Neural Networks . . . 8
2.2.1 Spiking neuron models . . . 8
2.2.1.1 Hodgkin-Huxley Neuron Model . . . 10
2.2.1.2 Integrate and Fire Neuron Model . . . 12
2.2.1.3 Izhikevich Neuron Model . . . 13
2.2.2 Learning Methods . . . 15
2.2.2.1 Spike-based Backpropagation . . . 15
2.2.3 Supporting Hardware . . . 18
2.2.3.1 Loihi Neurmorphic Computer . . . 20
2.2.3.2 Loihi2 Neurmorphic Computer . . . 21
2.2.3.3 Loihi2 and Loihi 1 Comparison . . . 23
2.2.4 Supporting Software . . . 27
2.2.4.1 SpikingJelly . . . 28
2.2.4.2 Lava . . . 28

3 Performance Improvement Using Auxiliary Learning 30
3.1 Introduction 31
3.2 Background and Related Work 32
3.2.1 Auxiliary learning 32
3.2.2 Input data augmentation 33
3.3 Methods 34
3.3.1 Problem Definition 34
3.3.2 Architecture 34
3.3.3 Training and Testing 35
3.4 Experiments and results 36
3.4.1 Training with one auxiliary task 37
3.4.2 Training with more than one auxiliary task 40
3.4.3 Using implicit differentiation 41
3.4.4 Comparison with State-of-the-Art SNNs 42
3.5 Discussion and Future Work 44
4 Multi-task Learning with Firing Threshold Modulation 46
4.1 Introduction 47
4.2 Methods 49
4.2.1 Problem Definition 49
4.2.2 Architecture 49
4.2.3 Training and testing 50
4.3 Simulation Performance of TM-SNN 51
4.3.1 Varying threshold 52
4.3.2 Using the task classifier block in training 54
4.3.3 Use of a firing threshold vs using an external input current . 54
4.3.4 Learning several classification tasks at the same time 56
4.3.5 Comparison of TM-SNN with ANN 58
4.4 Profiling on Loihi2 60
4.4.1 Network Selection 60
4.4.2 Execution time 61
4.4.3 Power consumption 64
4.4.4 Spiking Activity 65
4.4.5 Memory usage 67
4.5 Discussion and Future Work 68
5 Conclusions and Future Work 70

Appendix A Abbreviations 71

Appendix B List of publications by the author 72

B.1 Journal Publications 72

B.2 Journal Publications 72

References 74

Vita 95

LIST OF TABLES

Table Page

1 Loihi and Loihi 2 comparison (reprinted from [114]). 24

2 Network architecture used for analyzing DVS-CIFAR10 and DVS128-Gesture neuromorphic data. 37

3 The main task (M) and auxiliary tasks (A1, A2, A3) configurations. . . . 38

4 Accuracy for DVS-CIFAR10 dataset using auxiliary learning for 250 training epochs. 39

5 Testing accuracy for DVS128-Gesture dataset using auxiliary learning (250 training epochs). 39

6 Accuracy for DVS-CIFAR10 and DVS128-Gesture datasets using multiple auxiliary tasks - 250 training epochs. 41

7 Accuracy for DVS128-Gesture dataset using implicit differentiation on validation set - 250 training epochs. 42

8 Accuracy, precision, recall and F1 score for best performing SNN with AL for CIFAR10-DVS and DVSGesture128 datasets. 44

9 Comparison with staet-of-the-art SNNs for CIFAR10-DVS and DVSGesture-128 datasets. 45

10 Testing results of TM-SNN using different firing threshold values. 53

11 Testing accuracy of TM-SNN when using task classifier block 56

12 Testing accuracy of TM-SNN when an external current I_{ext} is used to control the network operation. 57

13 Testing accuracy of TM-SNN trained on four tasks. 58

14 Events and SynOps comparison of TM-SNN vs ANN. 59

15 Testing accuracy for the top-three, on task 1, TM-SNN 61

16 Precision, recall and F1 score for the top-three, on task 1, TM-SNN. . . . 62

17 Testing accuracy of MT-SNN trained for four tasks. 66

LIST OF FIGURES

Figure Page

1 Comparison of spiking neuron models (reprinted from [86]) 9

2 Channel opening and closing probability functions for (A) m, (B) h, and (C) n gating variables in HH model. 11

3 Gating variables dynamics. (A) Voltage-dependent target values, and (B) voltage-dependent time-constant functions for m, h, and n gating variables in HH model. 12

4 Izhikevich neuron's hyperparameter values and functionalities (reprinted from [86]). 14

5 Computational graph for a SNNs with LIF neurons and direct input contribution (reprinted from [63]). 17

6 Heaviside and sigmoid function comparison for implementing surrogate gradient descent (reprinted from [63]). 18

7 multi-task and auxiliary learning. **Left:** In multi-task learning, the goal is to perform more than one learning task at the same time, with all tasks being equally importance. **Right:** In auxiliary learning, the goal is to learn one main task while using one or more auxiliary tasks. . . 32

8 Auxiliary learning architecture. The network uses a multitask architecture in which only one task, "the main task", is of importance. The other tasks, "the auxiliary tasks", are used as additional regularization losses for helping the main task performance. The auxiliary tasks are only used during training. 35

9 Confusion matrix for best performing SNN with AL for CIFAR10-DVS (a) and DVSGesture128 (b) datasets. 43

10 TM-SNN architecture. It consists of three processing blocks connected in a feed-forward fashion: a feature extraction block and two classifier blocks. The label classifier outputs the labels for task 1 or task 2 (or more). The task classifier is used as a regularization mechanism to aid the feature extraction block learn a set of independent features for each task. 50

11 Training accuracy of ST-SNN (base case) and of TM-SNN using different threshold values: φ_1 is set to 1.25 while φ_2 changes from 1.5 to 10. 52

12 Example spiking output when TM-SNN is presented with input representing digit 4 (a) with $\varphi_1 = 1.25$ threshold (b) and with $\varphi_2 = 5$ (c). 55

13 Confusion matrix of TM-SNN number 3 for three task classification on NMNIST data. 63

14 TM-SNN's execution time in seconds for one sample presentation using $\varphi = 1.25$, $\varphi = 5$, and $\varphi = 10$. 64

15 TM-SNN's power consumption in watts for TM-SNN using $\varphi = 1.25$. . . . 65

16 Spiking activity in terms of SynOps when running TM-SNN on Loihi2 for three tasks configuration with $\varphi = 1.25$, $\varphi = 5$, and $\varphi = 10$. 67

17 Relative per core memory usage of TM-SNN on the Loihi2 neuromoprhic chip. 68

Resumen

MEJORA DE COMPUTACION NEUROMORFICA CON ARQUITECTURAS AVANZADAS DE REDES NEURONALES POR IMPULSOS

Introducción

La computación neuromórfica (NC, del inglés neuromorphic computing) pretende revolucionar el campo de la inteligencia artificial. Implica diseñar e implementar sistemas electrónicos que simulen el comportamiento de las neuronas biológicas utilizando hardware especializado, como matrices de puertas programables en campo (FPGA, del inglés field-programmable gate array) o chips neuromórficos dedicados [1, 2]. NC está diseñado para ser altamente eficiente, optimizado para bajo consumo de energía y alto paralelismo [3]. Estos sistemas son adaptables a entornos cambiantes y pueden aprender durante la operación, lo que los hace muy adecuados para resolver problemas dinámicos e impredecibles [4].

Sin embargo, el uso de NC para resolver problemas de la vida real actualmente está limitado porque el rendimiento de las redes neuronales por impulsos (SNN), las redes neuronales empleadas en NC, no es tan alta como el de los sistemas de computación tradicionales, como los alcanzados en dispositivos de aprendizaje profundo especializado, en términos de precisión y velocidad de aprendizaje [5, 6]. Varias razones contribuyen a la brecha de rendimiento: los SNN son más difíciles de entrenar debido a que necesitan algoritmos de entrenamiento especializados [7, 8]; son más sensibles a hiperparámetros, ya que son sistemas dinámicos con interacciones complejas [9], requieren conjuntos de datos especializados (datos neuromórficos) que

actualmente son escasos y de tamaño limitado [10], y el rango de funciones que los SNN pueden aproximar es más limitado en comparación con las redes neuronales artificiales (ANN) tradicionales [11]. Antes de que NC pueda tener un impacto más significativo en la IA y la tecnología informática, es necesario abordar estos desafíos relacionados con los SNN.

Contenido

Esta tesis tiene como objetivo reducir la brecha de rendimiento entre los sistemas informáticos neuromórficos y los sistemas informáticos tradicionales, especialmente en la resolución de tareas de reconocimiento de patrones. Para ello, nos centramos en abordar las cuestiones descritas anteriormente con dos enfoques.

En primer lugar, mejoramos el rendimiento de los SNN mediante el aprendizaje auxiliar (AL) [12]. AL es una técnica utilizada en ANN en la que la red se entrena en una tarea principal y en una o más tareas auxiliares adicionales. Mediante el uso de tareas adicionales, la red se ve obligada a encontrar parámetros más generales y robustos. Sin embargo, el uso de AL requiere una cuidadosa selección de tareas auxiliares, así como el método de combinar múltiples tareas durante el entrenamiento [13]. Específicamente, la red consta de un bloque de extracción de características que alimenta a los bloques de prediccion de tarea principal y tarea(s) auxiliar(es). La señal de entrada de impulsos es procesada por el primer bloque, el bloque de extracción de características, que luego se alimenta a los bloques clasificador de tareas principal y auxiliar para encontrar las salidas. La idea detrás de esta arquitectura es permitir que el bloque de extracción de características reciba retroalimentación del todos los bloques de predicción al mismo tiempo. La implementación de la SNN con AL se llevó a cabo utilizando el framework SpikingJelly para la simulación de SNNs [14].

Los experimentos fueron validados en los datasets neuromórficos DVS-CIFAR10 [15] y DVS128-Gesture [16].

En segundo lugar, mejoramos el rendimiento de los SNN mediante el diseño de una nueva arquitectura que cambia su funcionamiento a partir del control del umbral de disparo. La red propuesta es capaz de aprender dos o mas tareas diferentes pero realizando solo una de ellas a la vez [17]. La tarea que realiza la red se selecciona modulando el umbral de disparo de la neurona de impulsos utilizada. Esta operación está inspirada en la propiedad de neuromodulación de las neuronas biológicas, que pueden regular (modificar) su dinámica interna en función de estímulos externos [18]. La red propuesta, red neuronal de picos multitarea (MT-SNN), consta de tres bloques. Cada bloque está formado por una o más capas de neuronas de impulsos conectadas en serie. El algoritmo SLAYER es usado para entrenar el sistema [19]. Los experimentos y los resultados de implementar MT-SNN en el software neuromórfico ”Lava” de Intel son presentados para resolver clasificación multitarea en el dataset neuromórfico NMNIST.

En términos de implementación, probamos las redes desarrolladas en el chip neuromórfico Loihi2 [20]. Tenemos acceso a Loihi2 a través de un acuerdo entre VCU e Intel. El software desarrollado se agregará a la biblioteca existente de Loihi2, llamada Lava. La precisión, los requisitos de memoria, el consumo de energía y la latencia se utilizan para medir el rendimiento de los SNN desarrollados para resolver una variedad de tareas en datasets neuromórficos/basados en eventos.

Conclusiones

Los sistemas NC y los SNN que utilizan tienen un gran potencial para desarrollar IA adaptable de bajo consumo. Sin embargo, desafíos como la complejidad del entre-

namiento, la selección de hiperparámetros, la flexibilidad computacional y la escasez de datos de entrenamiento dificultan su uso más amplio.

En esta disertación, nuestro objetivo es aumentar el uso de NC mejorando el rendimiento de los SNN. Para lograr este objetivo, propusimos dos arquitecturas SNN para abordar estas limitaciones. La primera arquitectura utiliza aprendizaje auxiliar para mejorar el rendimiento del entrenamiento y la eficiencia de los datos. La arquitectura de la red consta de un bloque de extracción de características conectado de forma realimentada a un bloque de clasificación principal y uno o más bloques de clasificación de tareas auxiliares. Al usar tareas auxiliares, usamos información adicional durante el entrenamiento que ayuda en la regularización del bloque de extracción de características. Como resultado, el bloque de extracción de funciones se ve obligado a aprender funciones más generales y sólidas que ayudan a mejorar el rendimiento de la red en la tarea principal. Nuestros experimentos confirman que el uso de AL durante el entrenamiento da como resultado un mejor rendimiento. Sin embargo, la mejora depende de una cuidadosa selección de la(s) tarea(s) auxiliar(es) y del ajuste de la constante de tasa de pérdida. Los experimentos presentados se obtuvieron solo mediante simulación, es decir, utilizando la biblioteca neuromórfica SpikingJelly.

La segunda arquitectura, a saber, Red neuronal de picos multitarea (MT-SNN), aprovecha las capacidades de neuromodulación de las neuronas de picos para mejorar el rendimiento multitarea. Específicamente, la modulación del umbral de activación se utiliza para modificar el funcionamiento de la red siguiendo un enfoque de tarea única de tareas múltiples. Los resultados de nuestros experimentos probados con la plataforma de simulación neuromórfica Lava de Intel muestran que MT-SNN predice ambas tareas con una precisión ligeramente menor que ST-SNN. Además, la comparación del uso del umbral de disparo frente al uso de la corriente de entrada externa

muestra que con el umbral de disparo la precisión es mayor que con la corriente de entrada externa.

Si bien nuestros experimentos demuestran la efectividad de las arquitecturas propuestas, también revelan algunas limitaciones que valdría la pena estudiar en trabajos futuros. Una de esas limitaciones es el uso exclusivo de neuronas LIF. El trabajo futuro podría analizar el uso de neuronas más complejas, como las neuronas de Izhikevich. Para utilizar neuronas IZ, se requeriría una versión compatible con retropropagación basada en picos.

Abstract

ENHANCING NEUROMORPHIC COMPUTING WITH ADVANCED SPIKING NEURAL NETWORK ARCHITECTURES

By Paolo G. Cachi

A dissertation proposal submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2023.

Director: Krzysztof J. Cios,

Professor, Department of Computer Science

This dissertation addresses current limitations of neuromorphic computing to create energy-efficient and adaptable artificial intelligence systems. It focuses on increasing utilization of neuromorphic computing by designing novel architectures that improve the performance of the spiking neural networks. Specifically, the architectures address the issues of training complexity, hyperparameter selection, computational flexibility, and scarcity of training data. The first proposed architecture utilizes auxiliary learning to improve training performance and data usage, while the second architecture leverages neuromodulation capability of spiking neurons to improve multitasking classification performance. The proposed architectures are tested on the Intel's Loihi2 neuromorphic computer using several neuromorphic data sets, such as NMIST, DVSCIFAR10, and DVS128-Gesture. Results presented in this dissertation demonstrate the potential of the proposed architectures, but also reveal some limitations that are proposed as future work.

CHAPTER 1

INTRODUCTION

1.1 Overview

Neuromorphic computing (NC) aims to revolutionize the field of artificial intelligence. NC, which is based on mimicking the structure and function of the human brain, aims to achieve more efficient and flexible computation than the one offered by traditional architectures based on CPU/GPU computing [21, 22, 23, 24, 25].

NC involves the design and implementation of electronic systems that use specialized hardware, such as field-programmable gate arrays (FPGAs) or dedicated neuromorphic chips, to simulate the behavior of biological neurons [1, 2, 26]. It is designed to be much more efficient than traditional computer systems as NC is optimized for low-power consumption and high parallelism [3, 27]. Neuromorphic computing systems are also adaptable to changing environment and can learn during their operation [4, 28]. This makes them well-suited for solving dynamic and unpredictable scenario problems. Examples of existing although not commercially available neuromorphic computers are Intel's Loihi [20, 29, 30] and IBM's TrueNorth [31, 32].

NC use Spiking Neural Networks (SNNs) as model for its computation. SNNs are artificial neural networks that use discrete events, called spikes, for communicating information between neurons [33, 34, 35]. In this type of communication, each neuron sends a spike to other neurons when it reaches a certain threshold of activation. This allows for precise and efficient communication as the timing of the spikes conveys information about the strength and frequency of the inputs. SNNs are more efficient and biologically realistic than traditional artificial neural networks (ANNs) [36, 37, 38], neural networks that do not use spiking neurons. SNNs have been used in a variety of tasks including image recognition [39, 40, 41, 42], audio signal processing [43, 44, 45, 46, 47], and robotics [48, 49, 50]. Implementations of SNNs on neuromorphic hardware (such as Loihi2) reported orders of magnitude less energy consumption than ANNs in solving the just mentioned tasks [51, 52, 53, 54, 55].

However, the usage of NC for solving real-life problems is still limited as performance of SNNs is not as good as that of ANNs, when measured by accuracy and speed of learning [5, 6, 56]. Some reasons why SNNs may not perform as well as ANNs are:

1. **SNNs are more difficult to train than ANNs:** One of the challenges of training SNNs is that they do not have a differentiable activation function, which means that standard backpropagation techniques cannot be used for its training [7, 8, 57, 58]. Instead, SNNs are trained using specialized algorithms such as Spike-Timing-Dependent Plasticity (STDP) learning rule [59, 60, 61] and Backpropagation Through Time (BPTT) [62, 63, 19]. These techniques are typically more complex, difficult to understand, and require more computational resources compared with ANNs' training methods. For example, although STDP learning rule is based on findings in the biological brain, it involves complex interactions between multiple neurons which makes difficult to disentangle contribution of each individual neuron to the overall learning process.

2. **SNNs are sensitive to hyperparameters:** Spiking neurons have more hyperparameters than non-spiking ones used in ANNs, such as the threshold for generating spikes, the time constant for the decay of the post-synaptic potential, and the refractory periods of the neuron [9, 64]. These hyperparameters are more difficult to optimize because they interact with each other in complex

ways and have a significant impact on the behavior of the SNN. Using typical hyperparameter optimization techniques, such as grid search [65] or Bayesian optimization [66], is also more challenging since training SNNs requires long times.

3. **Data sets available for training SNNs are scarce and limited in size:** SNNs low performance is also linked with the limited size of available data for their training [10, 67]. As dynamic systems, SNNs are better suited for processing temporal data. Such data are called neuromorphic or event-based data [15, 68]. Unfortunately, there is a small number of such datasets currently available, and even worse, they are often small in terms of number of instances. As a result, SNNs trained with these datasets exhibit over-fitting and unstable convergence.

4. **The type of functions SNNs can approximate are more limited than in traditional ANNs:** The latter are able to approximate any continuous function to any desired degree of accuracy [11, 69]. SNNs currently approximate a more restricted set of functions depending on the spiking neuron model used and also due to the way information is transmitted between neurons [56, 6]. This makes it more difficult for SNNs to learn some types of patterns in the data. An example of this problem is the leaky integrate-and-fire neurons (LIF) [70, 34], commonly used spiking neuron model, which cannot be used to approximate biological neuron functionality such as spike frequency adaptation or bursting behavior [71, 18]. The latter functions, however, are crucial for maintaining stability and efficiency of neuronal circuits and modeling of certain brain functions.

NC has great potential for building next generation of AI systems that are more energy efficient and capable of adapting in real-time to changing environments. However, before NC can play a bigger role in AI and computing technology, solving the discussed above issues related to SNNs is required.

1.2 Objectives

This dissertation aims to reduce the performance gap between neuromorphic computing systems and traditional computing systems, especially in solving pattern recognition tasks. To do so, we focus on addressing the described above issues with two approaches.

First, we improve SNNs performance using Auxiliary Learning (AL) [12, 72]. AL is a technique used in ANNs in which the network is trained on the main task and on one or more additional, auxiliary, tasks. By using additional tasks, the network is forced to find more general and robust parameters. Use of AL, however, requires careful selecting of auxiliary tasks as well as the method of combining multiple tasks during training [13]. We attempt to find the best AL setup for SNNs.

Second, we improve SNNs performance by designing new architecture that can be modified based on changing of the firing threshold. This is done as an attempt to exploit dynamic capabilities of SNNs. The proposed network is able to learn two different tasks but performing only one of them at the time [17]. The task the network performs is selected by modulating the firing threshold of the spiking neuron used. This operation is inspired by the neuromodulation property of biological neurons, which can regulate (modify) their internal dynamics based on external stimuli [18]. Training with different firing thresholds allows the network to create internal pathways for processing multiple tasks independently, which reduces problems such as the negative transfer problem inherent in multi-task ANNs.

We enhance development of the two proposed approaches by using neuromorphic data augmentation and and advanced spiking neuron model. Specifically, we focus on the parametric leaky integrate and fire (PLIF) neuron model [73]. PLIF neurons are modified leaky integrate and fire neurons that allow training of not only the weights but also the membrane time constants. Using this neuron allows for neuron variability which is an important property for achieving network robustness. Direct training of the membrane constant has the additional benefit of eliminating its hand tuning which alleviates issue number 3. We use data neuromorphic data augmentation to reduce the problems of overfitting and unstable convergence present during training of SNNs [10].

In terms of implementation, we test the developed networks on the Loihi2 neuromorphic chip [20, 29]. We have access to Loihi2 through an agreement between VCU and Intel. The developed software will be added to the existing Loihi2's library, called Lava. Accuracy, memory requirements, energy consumption, and latency are used to measure performance of the developed SNNs for solving a variety of tasks on the neuromorphic/event-based data.

1.3 Organization

Chapter 2 discusses the relevant background and related work. Chapter 3 presents the use of auxiliary learning for improving SNNs performance. Chapter 4 presents the new SNN architecture that uses modulation of the neuron's firing threshold for implementing multi-task learning. Chapter 5 concludes the dissertation proposal with conclusions and remaining work.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we discuss relevant literature. In Section 2.1, we briefly discuss operation and training of traditional artificial neural networks. Then, in second 2.2, we give in-depth description of spiking neural networks. The most relevant spiking neuron models as well as spike-based backpropagation learning are covered.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are machine learning algorithms that are inspired by the structure and function of the brain's circuits [74, 38, 75, 76]. It consists of a large number of interconnected processing units, called neurons, which are organized in interconnected layers. By adjusting the strengths of the connections between the neurons, ANNs can be trained to perform a wide variety of tasks, including recognition and classification.

Depending on a specific architecture, ANNs can be divided into different types. The feed-forward multi-layer perceptron is the most basic ANNs [77]. It processes a p-dimensional input vector x into a q-dimensional output vector $y = f(x)$ by passing it sequentially through one or more non-linear transformation layers, called hidden layers, according to

$$f(x) = f_L(f_{L-1}(f_{L-2}(...f_1(x)))) \tag{2.1}$$

where each hidden layer is defined by $f_i(v) = g(Wv+b)$ with g being a non-linear function, W is the matrix with connections weights and b a bias vector; both W and b are normally defined as training parameters.

Other ANNs architectures use specialized connections between layers to accommodate processing of different types of input data. For example, convolutional neural networks (CNN) [78] and recurrent neural networks (RNN) [79, 80] are designed to better handle image and sequential data, respectively. To process image data, CNN integrate the use of receptive fields, shared weights, and spatial sub-sampling. RNN, on the other hand, uses feedback connections that allow information to be processed within different time steps.

ANNs are trained using a variety of algorithms, including the most popular backpropagation [81] and stochastic gradient descent [82]. These algorithms adjust the strengths of the connections between the neurons in the network, W and b, to minimize a cost function, J, that measures how well the network's outputs, $\hat{y}$, match true values, y. That is

$$\underset{W,b}{\text{minimize}}\, J(W,b) \tag{2.2}$$

where the cost function J is typically chosen as the average prediction loss (error over all training samples), L, plus a regularization term (used for introducing soft constraints within the search space), R:

$$J(W,b) = \frac{1}{N}\sum_{1}^{N} L(y_i, f(x,W,b)) + \lambda R(W,b) \tag{2.3}$$

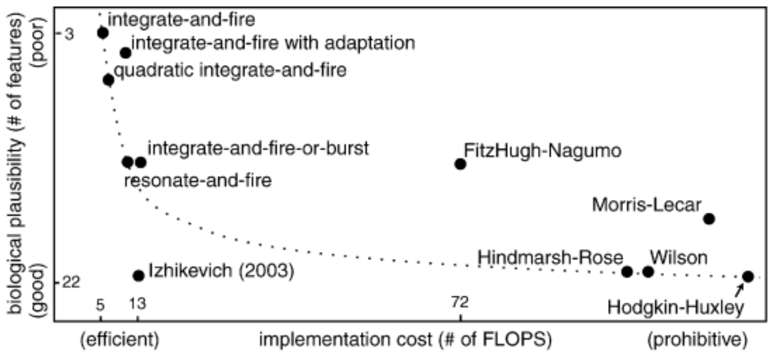
While ANNs have shown great promise in a variety of applications, they also have some limitations. One is their low energy usage efficiency. ANNs require a significant amount of computing power to train and operate, which is a problem where energy consumption is a concern, such as in the resource-constrained edge devices or in large-scale distributed systems. One approach to overcome this limitation is to use more efficient hardware and computation, such as a neuromorphic computer and spiking neural networks that run on it.

2.2 Spiking Neural Networks

SNNs are neural networks that more closely mimic the way the biological neurons work [33, 34, 35]. Unlike ANNs, which process data in a continuous manner, SNNs process data in a dynamic and event-driven fashion, with each neuron firing a spike (a brief impulse) in response to input over time. This type of operation allows SNNs to process information in a more temporally precise way than ANNs. This is because the spikes generated by the neurons can be timed very precisely, allowing for representation of temporal information. This is particularly useful for tasks such as speech recognition or video processing, where the order and timing of events is important. Additionally, SNNs are energy efficient when implemented on neuromorphic hardware; spiking communication consumes energy only when spikes are transmitted as opposed to continuous communication in ANNs. Using SNNs, however, requires developing new training methods and architecture designs in order to leverage their computational power. In the next subsections we will explain spiking neuron model as well as the training methods.

2.2.1 Spiking neuron models

In contrast to neuron models used in ANNs, which apply nonlinear transformations to continuous data $f : \mathbb{R}^P \rightarrow \mathbb{R}$, spiking neurons integrate time-dependent signals over time to generate a train of brief pulses called spikes $f(t) : X(t) \rightarrow y(t)$, where $X(t) = \{x_1(t), x_2(t), \ldots, x_p(t)\}$ and $y(t) = \sum_{t_f} \delta(t - t_f)$ [83, 84, 85]. Different spiking neuron models have been developed to account for different levels of biological similarity. The well-known spiking neuron models, their computational complexity, biological plausibility, and function capabilities are illustrated in Figure 1:



Models	biophysically meaningful	tonic spiking	phasic spiking	tonic bursting	phasic bursting	mixed mode	spike frequency adaptation	class 1 excitable	class 2 excitable	spike latency	subthreshold oscillations	resonator	integrator	rebound spike	rebound burst	threshold variability	bistability	DAP	accommodation	inhibition-induced spiking	inhibition-induced bursting	chaos	# of FLOPS
integrate-and-fire	-	+	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	5
integrate-and-fire with adapt.	-	+	-	-	-	-	+	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	10
integrate-and-fire-or-burst	-	+	+		+	-	+	+	-	-	-	-	+	+	+	-	+	+	-	-	-		13
resonate-and-fire	-	+	+	-	-	-	-	+	+	-	+	+	+	+	-	-	+	+	+	-	-	+	10
quadratic integrate-and-fire	-	+	-	-	-	-	-	+	-	+	-	-	+	-	-	+	+	-	-	-	-	-	7
Izhikevich (2003)	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	13
FitzHugh-Nagumo	-	+	+	-		-	-	+	-	+	+	+	-	+	-	+	+	-	+	+	-	-	72
Hindmarsh-Rose	-	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	120
Morris-Lecar	+	+	+	-		-	-	+	+	+	+	+	+	+		+	+	-	+	+	-	-	600
Wilson	-	+	+	+			+	+	+	+	+	+	+	+	+	+		+	+				180
Hodgkin-Huxley	+	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	1200

Fig. 1. Comparison of spiking neuron models (reprinted from [86])

The Hodgkin-Huxley (HH) [87] neuron model most closely simulates biological neurons. However, it is also the most computationally demanding which restricts its usage in SNNs. At the other extreme is the simple integrate and fire neuron model (IF) [83] which is thus widely used in SNNs. A model that achieves good balance between biological similarity and low computational cost is the Izhikevich (IZ) neuron model [86]. We discuss below specific mathematical models for these three types of spiking neuron models.

2.2.1.1 Hodgkin-Huxley Neuron Model

The Hodgkin-Huxley (HH) neuron model simulates membrane potential behavior of biological neurons [83, 85, 87]. It defines the neuron's membrane potential ($\frac{du}{dt}$) based on the total contribution of gate-dependent sodium, potassium and leak ion currents, as follows:

$$C_M \frac{du}{dt} = I_{Na}(u, m, h) + I_K(u, n) + I_L(u) + I_{app} \tag{2.4}$$

where C_M is membrane capacitance; I_{app} is an external current source; m, h and n are gating variables; and $I_{Na}(u, m, h)$, $I_K(u, n)$, and $I_L(u)$ are the sodium, potassium and leak ion currents, described by:

$$I_{Na}(u, m, h) = g_{Na} \cdot m^3 \cdot h \cdot (E_{Na} - u) \tag{2.5}$$

$$I_K(u, n) = g_K \cdot n^4 \cdot (E_K - u) \tag{2.6}$$

$$I_L(u) = g_l \cdot (E_L - u) \tag{2.7}$$

where g_{Na}, g_K, and g_L are maximum conductances; and E_{Na}, E_k, and E_L are reverse potentials. Each gating variable, m, h or n, is described in terms of channel opening and closing voltage-dependent functions as follows:

$$\frac{dx}{dt} = \alpha_x(u) \cdot (1 - x) - \beta_x(u) \cdot x \tag{2.8}$$

where x is used to represent the gating variable type, m, h or n; $\alpha_x(u)$ and $\beta_x(u)$ are the channel opening and closing voltage-dependent functions, respectively. Typical forms of the channel opening and closing functions for m, h and n gating variables [87, 83] are shown in Figure 2:

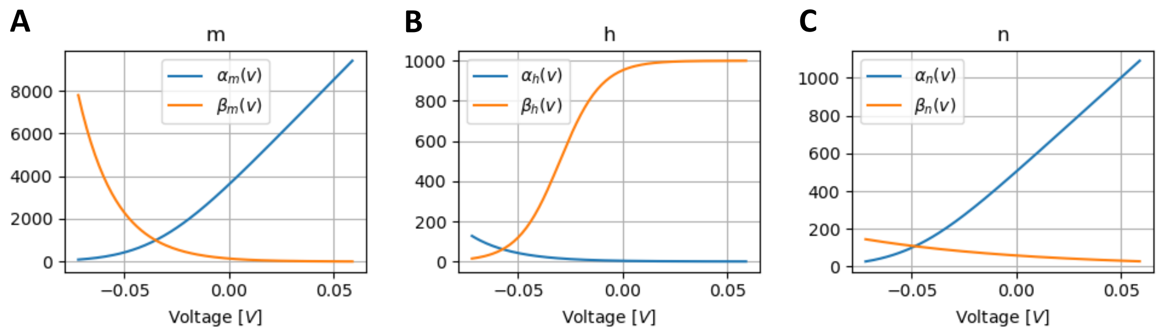


Fig. 2. Channel opening and closing probability functions for (A) m, (B) h, and (C) n gating variables in HH model.

Alternatively, to Equation 2.8, the gating variables can be described in terms of channel target and time-constant functions as follows:

$$\frac{dx}{dt} = \frac{x_\infty(u) - x}{\tau_x(u)} \tag{2.9}$$

where $x_\infty(u)$ is a target voltage-dependent function; and $\tau_x(u)$ is a time-constant voltage-dependent function. The relationships between the alpha, beta, target, and time-constant functions are given by:

$$x_\infty(u) = \frac{\alpha_x(u)}{\alpha_x(u) + \beta_x(u)} \tag{2.10}$$

$$\tau_x(u) = \frac{1}{\alpha_x(u) + \beta_x(u)} \tag{2.11}$$

Figure 3 shows typical opening and closing functions for the m, h and n gating variables [87, 83].

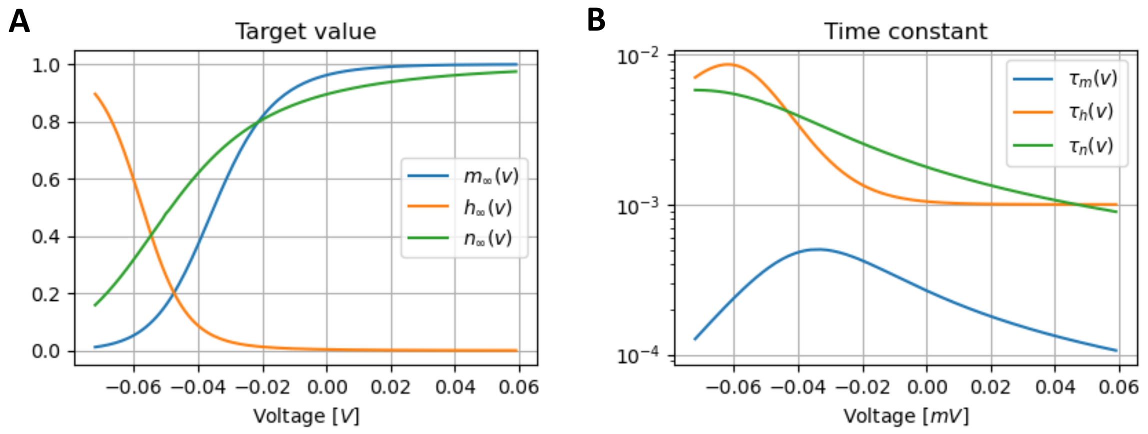


Fig. 3. Gating variables dynamics. (A) Voltage-dependent target values, and (B) voltage-dependent time-constant functions for m, h, and n gating variables in HH model.

2.2.1.2 Integrate and Fire Neuron Model

While HH models the membrane potential in terms of the interaction of sodium, potassium and leak ion currents, the integrate and fire neuron model (IF) simplifies it and implements it as the combination of an integrator mechanism that builds the membrane potential based on the input signals, and a firing mechanism, that emits an output spike every-time the membrane potential is greater than a threshold [83, 84, 85], as follows:

$$\frac{du}{dt} = \frac{f(u)}{\tau_f} + \frac{g(u)}{\tau_g} I(t) \tag{2.12}$$

where $f(u)$ and $g(u)$ are linear/nonlinear functions of the instant membrane potential value u; τ_f and τ_g are time decaying constants; and $I(t)$ is an input driving signal. The most typical choices for the function $f(u)$ can be a linear function $f_1(t)$, a quadratic $f_2(t)$, or an exponential $f_3(t)$:

$$f_1(u) = -(u - u_r) \tag{2.13}$$

$$f_2(u) = a_0(u - u_r)(u - u_c) \tag{2.14}$$

$$f_3(u) = -(u - u_r) + \Delta_T \exp \frac{(u - \vartheta)}{\Delta_T} \tag{2.15}$$

where u_r represents a resting membrane potential value, Δ_T the sharpness factor, ϑ the threshold variable/constant, and a_0 and u_c are constants with $a_0 > 0$ and $u_c > u_{rest}$. The term $g(u)$ is used to couple the driving signal $I(t)$ into the membrane potential function. It follows direct contribution $g(u) = 1$ or conductance-based contribution $g(u) = (u_i - u)$, where u_i is an input reverse potential constant.

The membrane potential $u(t)$ is pass-through the spike generation mechanism, where spikes are produced every time t^f the membrane potential value crosses, from below, a fixed or adaptive threshold ϑ. If the threshold is adaptive, its value follows an exponential decay process with constant increment α after each spike:

$$\frac{d\vartheta}{dt} = \frac{\vartheta_0 - \vartheta}{\tau_{\vartheta_0}} + \sum_{t^f} \alpha\delta(t - t_i^f) \tag{2.16}$$

where ϑ_0 is a threshold offset value, and τ_{ϑ_0} a decay time constant. After each spike is generated a reset signal is used to reset the membrane potential and halt its operation for time t_r.

2.2.1.3 Izhikevich Neuron Model

The Izhikevich (IZ) neuron model is a great simplification of HH. It captures many of the essential features of HH, but it does so with just two equations and parameters, making it more computationally efficient and easier to use [86]. IZ defines the neuron behavior in terms of a membrane potential variable, u, and a recovery variable, v. The recovery variable is used to define how fast the membrane potential goes to its resting state. The equations for both variables are given by:

$$\frac{du}{dt} = 0.04u^2 + 5u + 140 - v + I(t) \tag{2.17}$$

$$\frac{dv}{dt} = a(bu - v) \tag{2.18}$$

$$\text{if } u \geq 30mV \text{ then } u \leftarrow c,\ v \leftarrow v + d,\ \text{emit spike} \tag{2.19}$$

where a, b, c and d are hyperparameters that control the functionality of the neuron. Figure 4 shows eight different types of neuron functionality for different combinations of the hyperparameters a, b, c and d.

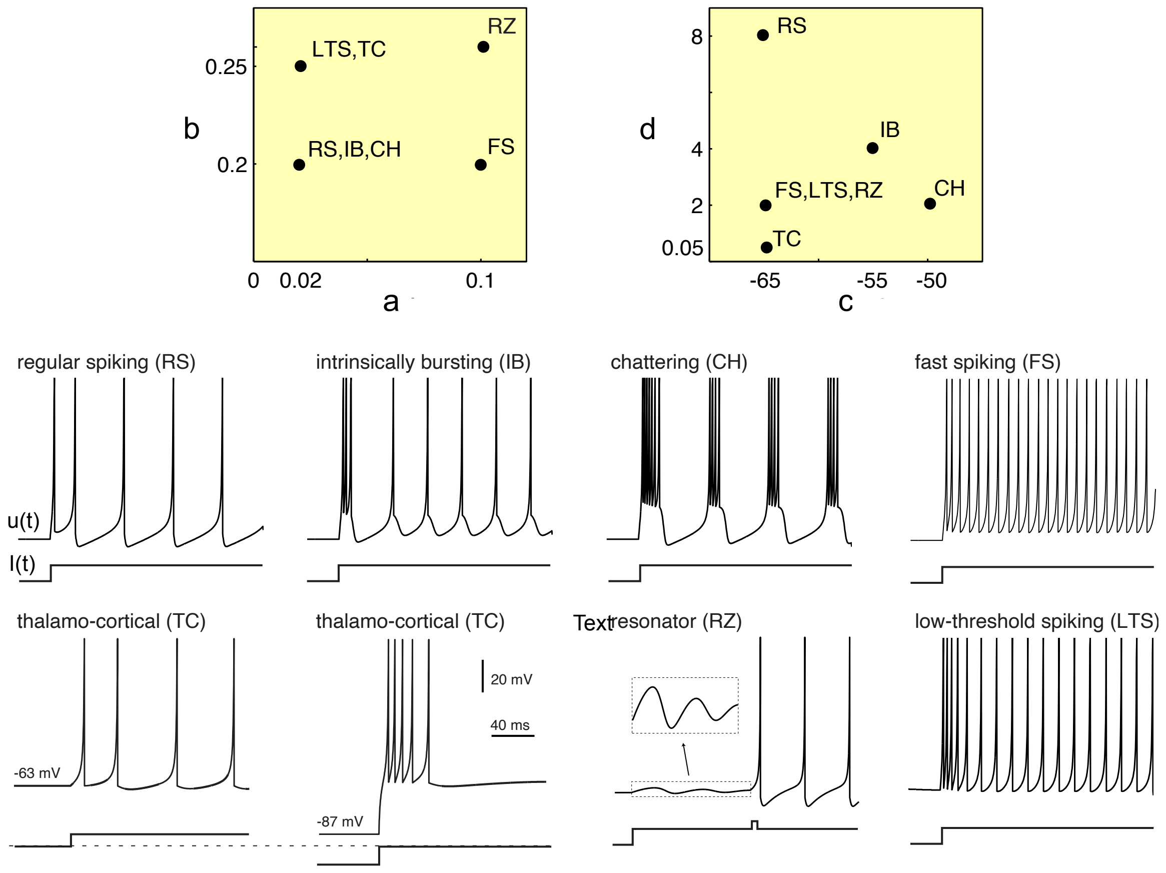


Fig. 4. Izhikevich neuron's hyperparameter values and functionalities (reprinted from [86]).

While IZ can reconstruct the membrane potential of several types of neurons with negligible error, it fails at bounding the contribution of the input current (which is inherently bounded by the gating mechanism in HH). This issue can cause the neuron to reach not biologically plausible firing frequencies. A later work [88, 89] corrected this problem by adding an upper bound restriction on the firing frequency. The equation 2.19 was replaced by:

$$\begin{aligned}&\text{if } u \geq 30mV \text{ and } t - t_{prev} \geq \tau_{min} \text{ then } u \leftarrow c,\ v \leftarrow v + d,\ \text{emit spike}\\&\text{else if } u \geq 30mV \text{ then } u \leftarrow 30mV, \text{no spike}\end{aligned} \tag{2.20}$$

2.2.2 Learning Methods

Learning in SNNs involves updating the between-neuron connection weights. However, different nature of communication between spiking neurons requires the development of specialized algorithms [7, 8, 57, 58]. There are several different approaches used for SNNs learning, which depend on the specific architecture and type of learning problem. Some of the most commonly used learning methods are Spike Timing Dependent Plasticity (STDP) [59, 60, 61] that performs unsupervised local learning, reward modulated STDP (R-STDP) [90, 91] for reinforcement learning through STDP, and spiking-based backpropagation (SBP) for supervised learning [62, 63, 19]. In this work we focus on SBP because of its advantages including bigger flexibility, SBP can be used in many network architectures, and ease of handling (SBP is well understood method as it derives of the widely used backpropagation learning).

2.2.2.1 Spike-based Backpropagation

Like the standard backpropagation algorithm, SBP uses gradient descent to update the weights in order to minimize the error between the network's predicted output and the desired output. However, unlike the standard backpropagation algorithm, SBP can handle dynamic operation of spiking neurons and the associated non-linearity of the communication via spikes. To enable incorporation of temporal dependencies in the training process, SBP uses backpropagation through time (BPTT) [19], and in order to overcome the non-linearity of the spiking mechanisms uses surrogate gradient functions (SGD) [63].

BPTT was originally developed for training recurrent neural networks, which can process sequences of inputs [92]. In BPTT, the error signal is propagated back through the network not just over a single time step, but over multiple time steps. This allows the network to take into account the past history of inputs and outputs while adjusting its weights. The trick that allows the use of BPTT in SNNs is that the spiking neuron model can be unfolded into a recurrent computation system [19]. Since the membrane potential is governed by differential equations, such as equation 2.12, its value intrinsically depends on the previous state, which is in fact the definition of a recurrent system, as follows

$$\frac{du}{dt} = F(u, I(t)) \quad \text{equivalent to} \quad u[t+1] = u[t] + F(u[t], I[t]) \tag{2.21}$$

The exact unfolded equation will depend on the specific spiking neuron model and architecture used. The unfolded equations for a SNNs with IF neurons and direct input contribution are given by:

$$u_i^{(l)}[t+1] = \beta u_i^{(l)}[t] + I_i^{(l)}[t] - s_i^{(l)}[t] \tag{2.22}$$

$$I_i^{(l)}[t+1] = \alpha I_i^{(l)}[t] \sum_j W_{ij} S_j^{(l-1)}[t] + \sum_j V_{ij} S_j^{(l)}[t] \tag{2.23}$$

$$s_i^{(l)}[t] = \theta(u_i^{(l)}[t] - \vartheta) \tag{2.24}$$

where $u_i^{(l)}$ is the membrane potential of neuron i at layer l; $I_i^{(l)}$ is the input driven signal resulted from the linear combination of the spiking signal output, $s_i^{(l)}$, from the previous layer and the weights W (feed-forward) and V (recurrent); and α and β are decay constants. The output from the spiking neuron is defined as the Heaviside function, θ, of the membrane potential minus the threshold ϑ. The unfolded equations of the SNNs are summarized in the computation graph shown in Figure 5.

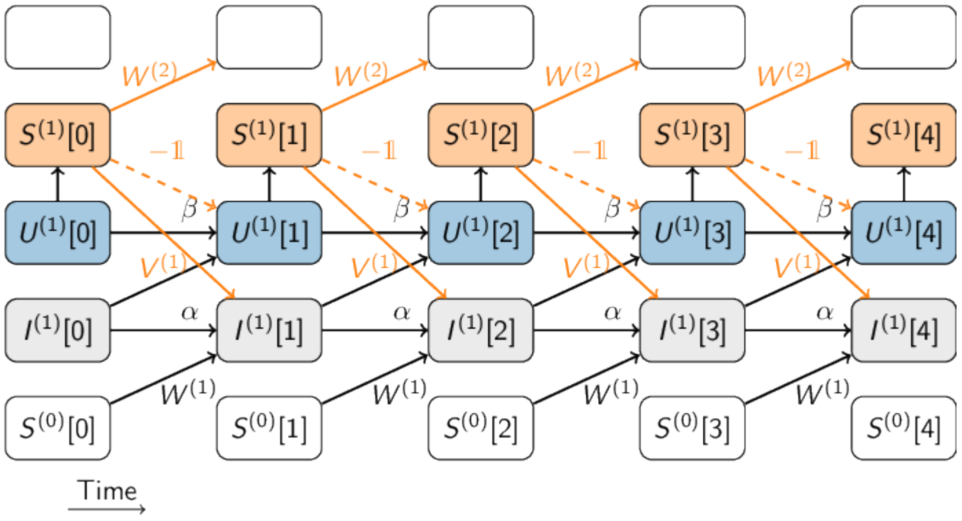


Fig. 5. Computational graph for a SNNs with LIF neurons and direct input contribution (reprinted from [63]).

Although implementing the computational graph of the spiking operation is feasible through tools such as PyTorch or TensorFlow, the non-differentiability of the Heaviside function impedes correct calculation of the error gradients. To overcome this issue, the SGD method is used [63]. The main idea behind SGD is to use a replacement function as the gradient for the non-differentiable function. This replacement is only used during the gradient calculation or backward pass training stage. The Heaviside function is still used during the forward pass to maintain the correct operation of the network. Different surrogate functions are used; however, efficient functions are preferred as BPTT is compute time intensive when used in SNNs. The typical function used as replacement of the gradient for the Heaviside function, $\Theta(x)$, is the sigmoid function, $\sigma(x)$, see Figure 6.

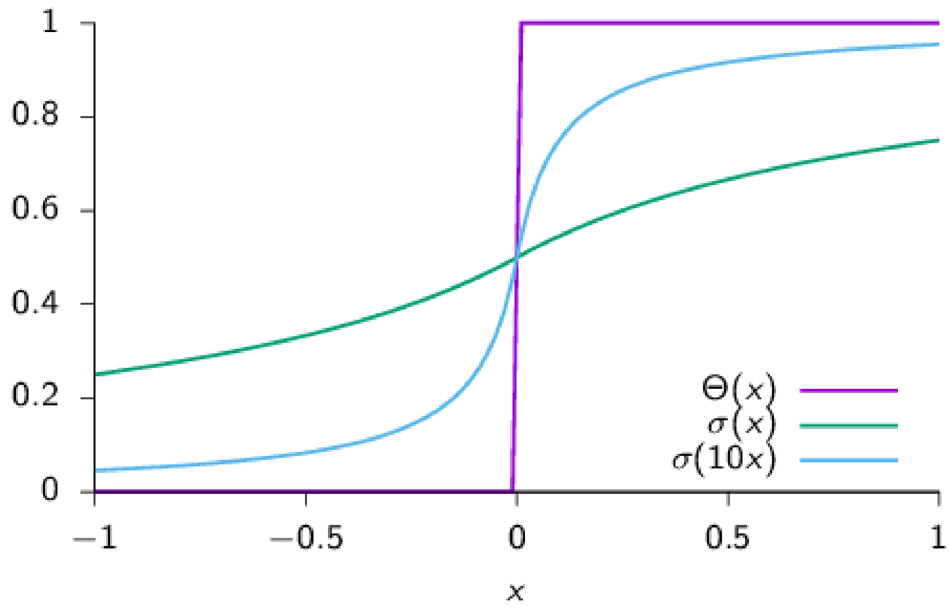


Fig. 6. Heaviside and sigmoid function comparison for implementing surrogate gradient descent (reprinted from [63]).

2.2.3 Supporting Hardware

Different specialized hardware and neuromorphic computing systems have been developed for running SNNs [93, 22, 27]. These systems can be categorized based on their architectural approach into digital, analog, and mixed-signal platforms. Each type offers distinct advantages and considerations.

Digital neuromorphic computing platforms utilize digital circuitry to implement SNNs. These platforms employ digital representations of neurons and synapses, providing precise control over network behavior and connectivity. They offer flexibility in terms of network size and complexity, making them well-suited for large-scale simulations. Digital platforms also facilitate on-chip learning and leverage parallelism for efficient computation. Such platforms can be designed using fully custom Application

Specific Integrated Circuit (ASIC) or Field Programable Gate Array (FPGA) based implementations.

Examples of ASIC-based implementations include IBM's TrueNorth [32, 94, 95], Intel's Loihi [30] and Loihi2 [20, 29], and SpiNNaker [96, 97, 98]. These platforms leverage custom-designed integrated circuits to achieve their neuromorphic functionality. On the other hand, FPGA-based implementations focus on exploring specific in-hardware training algorithms. For instance, some FPGA-based implementations utilize a modified STDP rule that replaces exponential operations with shift operations to reduce logic resource consumption [99]. Other implementations involve competitive Hebbian learning on chip, utilizing biologically plausible Izhikevich neurons implemented on FPGA [100]. Additionally, there are implementations that use a simplified STDP rule with 1-bit synaptic weights to reduce computing and communication overhead [101].

Analog neuromorphic computing platforms, such as Neurogrid [102, 103], utilize analog circuitry to closely mimic the behavior of neurons and synapses. These platforms take advantage of the continuous nature of electrical signals to emulate the dynamics of SNNs [1]. Analog implementations offer the potential for high-speed processing and more efficient energy consumption due to the utilization of continuous signals. They excel in applications requiring real-time processing or fine-grained control of neural dynamics. However, analog systems may encounter challenges related to noise, variability, and scalability.

In addition to Neurogrid, there is a wide range of other neuromorphic analog implementations [104, 105, 106]. These implementations include custom Field-Programmable Analog Arrays (FPAAs), such as the field programmable neural array (FPNA) [107] and the NeuroFPAA [108]. These platforms employ dedicated analog circuitry to simulate neural behavior, offering alternative solutions for implementing neuromorphic systems and exploring analog computing approaches.

Mixed-signal neuromorphic computing platforms combine digital and analog components to harness the advantages of both domains. These platforms integrate the flexibility and programmability of digital circuits with the efficiency and fine-grained analog processing. By combining digital and analog elements, mixed-signal platforms strike a balance between flexibility, power efficiency, and high-speed analog processing. Noteworthy members of the mixed analog/digital family include the prominent projects BrainScaleS [109, 110] and analog neuromorphic systems that store synaptic weights in digital memory to ensure dependability and longer lifetime [111, 112] or use digital communication within or across the neuromorphic chips [113].

Among the various options of neuromorphic computing hardware, in this dissertation we use Intel's Loihi neuromorphic computers to conduct experiments on multi-task learning with firing threshold modulation (Chapter 4). Specifically, we utilize the advanced iteration of Loihi, referred to as Loihi2, which offers substantial enhancements over its predecessor. In the following two subsections, we present a brief overview of the essential functionalities and primary characteristics of both Loihi and Loihi2.

2.2.3.1 Loihi Neurmorphic Computer

Loihi is a digital fully custom ASIC neuromorphic chip developed by Intel [30]. It is fabricated using Intel's 14-nm process and implements a total of 130,000 artificial current-based LIF neurons and 130 million synapses. Loihi allows customization of individual neurons, synapses, and network connectivity, allowing for the creation of highly tailored neural circuit designs. It also features a programmable microcode learning engine specifically designed for on-chip training.

Architecturally, Loihi employs a highly interconnected manycore mesh consisting of 128 neuromorphic cores and three embedded x86 Lakemont processor cores that are used to help support advanced learning rules and core management. To facilitate communication and coordination among these cores, Loihi incorporates an asynchronous network-on-chip (NoC) infrastructure. The NoC serves as the primary means of transporting packetized messages between cores.

The NoC supports various types of messages, including write, read request, and read response messages for core management and x86-to-x86 messaging. Spike messages, crucial for SNN computations, are also transmitted through the NoC, while barrier messages enable time synchronization among the cores. Messages can originate externally from a host CPU or on-chip from the x86 cores, and they can be directed to specific on-chip cores as needed. With its mesh protocol, Loihi achieves impressive scalability, allowing for up to 4,096 on-chip cores and 16,384 chips, thereby enabling large-scale neuromorphic simulations and complex computational tasks.

Synapses in Loihi are designed to have fully configurable and in-hardware adaptable weights, delays, and tags. Furthermore, each synapse associates with multiple presynaptic traces, incorporating different exponential smoothing parameters. These features contribute to the versatility of Loihi, allowing for the implementation of various neural network architectures and synaptic plasticity mechanisms.

2.2.3.2 Loihi2 Neurmorphic Computer

Loihi 2 chip [20, 29] is a significant advancement in terms of performance and efficiency over its predecessor Loihi [30] and serves as the platform for this research. The chip retains high programmability and flexibility of Loihi allowing to fully customize neuronal behavior, synaptic connections, and network topologies. Importantly, Loihi2 exhibits better energy efficiency, enabling running on it the computationally intensive tasks. The key features and innovations of Loihi 2 are as follows:

1. Generalized event-based messaging: Unlike its predecessor, Loihi 2 allows spikes to carry integer-valued payloads, enabling event-based messaging. This enhancement preserves the sparse and time-coded communication properties of spiking neural networks (SNNs) while providing greater numerical precision.

2. Greater neuron model programmability: Loihi 2 introduces a programmable pipeline in each neuromorphic core, expanding its range of neuron models without compromising performance or efficiency. This increased programmability supports common arithmetic, comparison, and program control flow instructions, enabling a richer space of use cases and applications.

3. Enhanced learning capabilities: While Loihi primarily supported two-factor learning rules, Loihi 2 introduces localized "third factors" mapped to specific synapses. This advancement allows for the implementation of various neuro-inspired learning algorithms, including approximations of the error backpropagation algorithm commonly used in deep learning.

4. Capacity optimizations for improved resource density: Loihi 2 incorporates numerous capacity optimizations to compress and maximize the efficiency of neural memory resources, resulting in improved overall resource density. Fabricated with Intel's pre-production version of the Intel 4 process, Loihi 2 achieves greater application scales within a single neuromorphic chip.

5. Faster circuit speeds: Loihi 2 features fully redesigned and optimized asynchronous circuits, resulting in significant processing speed improvements. With processing speed gains ranging from 2x to 10x, Loihi 2 can process neuromorphic networks up to 5000x faster than biological neurons, supporting highly efficient and high-speed computations.

6. Interface improvements: Loihi 2 offers more standard chip interfaces, including faster and higher-radix interfaces. It supports faster asynchronous chip-to-chip signaling bandwidths, destination spike broadcast features to reduce inter-chip bandwidth utilization, and three-dimensional mesh network topologies with improved scalability ports. Loihi 2 also supports seamless integration with a wider range of standard chips and emerging event-based vision and sensor devices.

2.2.3.3 Loihi2 and Loihi 1 Comparison

Table 1 compares the main features of Loihi 2 and Loihi 1.

Table 1.: Loihi and Loihi 2 comparison (reprinted from [114]).

Resources/Features	Loihi	Loihi 2
Process	Intel 14nm	Intel 4nm
Die Area	60 mm^2	31mm^2
Core Area	0.41 mm^2	0.21 mm^2
Transistors	2.1 billion	2.3 billion
Max # Neuron Cores/Chip	128	128
Max # Processors/Chip	3	6
Max # Neurons/Chip	128,000	1 million
Max # Synapses/Chip	128 million	120 million
Memory/Neuron Core	208 KB, fixed allocation	192 KB, flexible allocation
Neuron Models	Generalized LIF	Fully programmable
Neuron State Allocation	Fixed at 24 bytes per neuron	Variable from 0 to 4096 per neuron depending onneuron model requirements

Connectivity Features	Basic compression features: • Variety of sparse and dense synaptic compressionformats • Weight sharing of source neuron fanout lists	In addition to the Loihi 1 features: Shared synapses for convolution • Synapses generated from seed • Presynaptic weight-scaling factors • Core fan-out list compression and sharing • Broadcast of spikes at destination chip
Information Coding	Binary spike events	Graded spike events (up to 32-bit payload)
Neuron State Monitoring (for development/debug)	Requires remote pause and query of neuron memory	Neurons can transmit their state on-the-fly
Learning Architecture	Programmable rules applied to pre-, post-, and rewardtraces	Programmable rules applied to pre-, post-, and generalized “third-factor” traces
Spike Input	Handled by embedded processors	Hardware acceleration for spike encoding and synchronization of Loihi with external data stream
Spike Output	1,000 hardware-accelerated spike receivers perembedded processor	In addition to the Loihi 1 feature, hardware accelerated spike output per chip for reporting graded payload,timing, and source neuron

External Interfaces	Proprietary asynchronous interface	Support for standard synchronous (SPI) and asynchronous (AER) protocols, GPIO, and 1000BASE-KX,2500BASE-KX, and 10GBase-KR Ethernet
Multi-Chip Scaling	2D tile-able chip array Single inter-chip asynchronous protocol with fixedpin-count	3D tile-able chip array Range of inter-chip asynchronous protocols withvariable pipelining and pin-counts optimized fordifferent system configurations
Timestep Synchronization	Handled by cores	Accelerated by NoC routers

2.2.4 Supporting Software

Software plays a crucial role in development of SNNs and NC systems, as it provides the necessary tools and frameworks to program and train these hardware architectures effectively. The software stack for NC typically consists of multiple layers, each serving a specific purpose. At the lowest level, there are programming interfaces and compilers that enable developers to write and optimize code for neuromorphic hardware. These tools abstract the underlying hardware complexity, allowing users to focus on algorithm design and application development.

On top of the low-level interfaces, higher-level software frameworks and libraries provide specialized functions for building and training SNNs on neuromorphic architectures. These frameworks often include tools for defining network topologies, specifying learning rules, and configuring the hardware parameters. They also support efficient simulation and emulation of SNNs, enabling developers to test and validate their algorithms before deploying them on the actual hardware.

Open-source software initiatives, such as Lava [115], SpikingJelly [14], NEST [116], Brian [117], and SpiNNaker [97], have played a significant role in advancing the development of neuromorphic computing software. These projects provide comprehensive toolkits and libraries that facilitate the design, simulation, and optimization of neural networks on a range of neuromorphic hardware platforms. In the context of this dissertation, two specific software frameworks have been leveraged: SpikingJelly and Lava. We use the SpikingJelly framework for the implementation and testing of the auxiliary learning architecture presented in Chapter 3, and the Lava framework for the implementation, testing and deployment of the multi-task learning with threshold modulation architecture in Chapter 4.

2.2.4.1 SpikingJelly

SpikingJelly is an open-source software framework specifically designed for SNNs. It offers a comprehensive set of tools and functionalities that facilitate the implementation, simulation, and analysis of SNN models. Developed by the Institute of Neuroscience, Chinese Academy of Sciences, SpikingJelly aims to provide researchers and developers with a flexible and efficient platform for exploring the dynamics and learning mechanisms of spiking neurons.

SpikingJelly provides a range of built-in functions for constructing network architectures, neuron models, configuring synaptic connections, and defining learning rules. It also supports advanced training algorithms, such as Spike-based backpropagation training (learning method used in this dissertation), STDP and reward-modulated STDP.

Additionally, SpikingJelly offers efficient simulation capabilities optimized for large-scale networks. It employs parallel computing techniques to accelerate the simulation process, making it suitable for simulating complex neural models of neurons and synapses. The framework integrates seamlessly with Pytorch deep learning library.

2.2.4.2 Lava

In addition to its hardware, Loihi 2 is accompanied by Lava software environment [115], which provides a platform-agnostic framework for developing SNNs and neuromorphic applications. The framework is designed to be modular, composable, and extensible, allowing integration of algorithmic ideas from different sources and enabling contributions to a shared code base. Lava's hierarchical structure aims to make neuromorphic programming accessible to a wider developer community.

Lava includes the Magma low-level interface, which facilitates mapping and execution of neural network models and sequential processes on neuromorphic hardware. Magma supports cross-platform execution, enabling simulation on CPUs/GPUs before deployment on Loihi 2 chip or other neuromorphic platforms. The framework incorporates a profiler tool that allows developers to measure/ estimate performance and energy consumption across targeted back-end platforms. Lava also provides support for offline training using SLAYER [19] algorithm that enables efficient training of TM-SNNs.

Additional features of Lava include support for offline training using tools like SLAYER, integration with third-party frameworks such as Robotic Operating System (ROS), YARP, TensorFlow, PyTorch, Nengo, and other. Lava allows free use without legal agreements with Intel. However, the lowest-level components required for deploying applications on Loihi 2 hardware systems are accessible only to engaged Intel NRC members at no cost.

CHAPTER 3

PERFORMANCE IMPROVEMENT USING AUXILIARY LEARNING

In this chapter, we address the enhancement of SNNs through usage of an advanced ANN training method known as Auxiliary Learning (AL). In AL, the network is designed to support training of multiple tasks, however only one of all the tasks is consider of interest (i.e. the accuracy on the auxiliary tasks is not important). Our results indicate that training with AL improves their accuracy. Different scenarios, including manual and automatic combination loss using implicit differentiation, are explored to analyze usage of auxiliary tasks. The rest of the chapter is structured as follows: Section 3.1 introduces usage of AL for training SNN; Section 3.2 gives a brief description of background and relevant work. Section 3.3 describes the proposed framework for training SNNs using AL. Section 3.4 presents our experiments that confirms the viability of AL; the chapter ends with discussion and future work.

The following papers related to this chapter were published:

1. Paolo G. Cachi, Sebastian Ventura, and Krzysztof J. Cios, “Improving Spiking Neural Network Performance with Auxiliary Learning”, under review.

2. Paolo G. Cachi, Sebastian Ventura, and Krzysztof J. Cios, “CRBA: A Rate-Based Algorithm Based on Competitive Spiking Neural Networks,” in Frontiers in Computational Neuroscience, vol. 15, p. 32, 2021.

3. Paolo G. Cachi, Sebastian Ventura, and Krzysztof J. Cios, “Fast Convergence of Competitive Spiking Neural Networks with Sample-Based Weight Initialization,” in Information Processing and Management of Uncertainty in Knowledge-

Based Systems, pp. 773–786, 2020.

3.1 Introduction

One of the main difficulties while training SNNs is the limited size of available data [10, 67]. SNNs operate in the temporal domain and are well suited for processing temporal data, such as neuromorphic or event-based data [118]. Currently, there are few temporal datasets available for training and they frequently contain small number of samples. For example, the two most used for comparison datasets of SNNs performance are DVS-CIFAR10 [15] and DVS128-Gesture [16] datasets contain only 10K and 319 samples respectively. As a result, SNNs trained using these datasets exhibit overfitting and unstable convergence.

The problem of training when small-size data is available is not specific only to SNNs but to machine learning methods in general [119, 120]. Solutions proposed in the past involve two main approaches: data augmentation (the creation of new synthetic data by modification of input samples or latent feature vectors) and use of regularization methods: direct regularization by penalty loss or indirect regularization with AL. Only the first method has been studied in the framework of SNNs [73]. However, the use of data augmentation only does not allow to leverage the full potential of using more data. Therefore, implementing regularization methods such as AL on top of data augmentation is required for achieving better generalization of SNNs.

In this chapter we use AL as indirect regularization method for training SNNs. Auxiliary learning has been used in the past for improving the performance of ANNs, papers such as [121, 122, 12], have explore the use of one or multiple secondary tasks as a way of regularization. The attempts have proven to be helpful in the idea of increasing performance. Some limitations seen in the ANNs framework are still present for neuromorphic data [10, 13]. We present here the study of different ways for combination of the main and auxiliary losses as well as the selection of the auxiliary tasks and relation of the number of tasks to be used. The implementation of the network is carried out using the SpikingJelly framework for SNNs simulation [14]. The experiments are validated on DVS-CIFAR10 [15] and DVS128-Gesture [16] neuromorphic datasets.

3.2 Background and Related Work

3.2.1 Auxiliary learning

AL is a technique developed to improve the performance of ANNs when training data size is limited or expensive to collect [121, 12, 72]. In auxiliary learning a model is trained on multiple tasks at the same time in a similar setup as used in multi-task learning (ML) [123, 124], see Figure 7. The difference between ML and AL is that while ML strives for good performance on all tasks (treats all tasks as equal), AL focuses on performance of just one task (the main task) that the networks is to solve and treats all other tasks as auxiliary ones (used only to help improve the performance of the main task). The auxiliary tasks can be related, or not, to the main task.

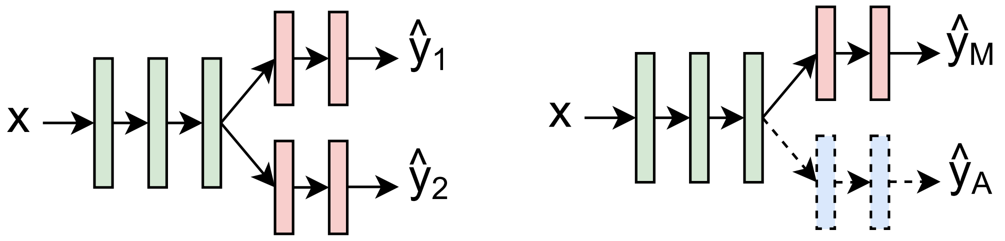


Fig. 7. multi-task and auxiliary learning. **Left:** In multi-task learning, the goal is to perform more than one learning task at the same time, with all tasks being equally importance. **Right:** In auxiliary learning, the goal is to learn one main task while using one or more auxiliary tasks.

AL approach has several advantages. By training a network on multiple tasks simultaneously, AL forces the network to learn more general transferable features, which can improve its performance on the main task. AL can also improve efficiency of training, as the network can learn from the auxiliary tasks without the need for additional training data or computation. This can make it more practical for training large complex neural networks. Finally, AL can serve as a regularization tool for the network, which can improve its generalization ability by reducing over-fitting.

Performing learning of multiple tasks, as done in AL, however, creates problems such as the negative transfer (when different tasks have conflicting goals, such as increasing performance for one task decreases performance of the other task(s)) [125]. Another challenge is how to efficiently combine multiple loss functions, i.e., how to weight the losses so the main task is preferred [13]. In this chapter, we tackle these questions by investigating various setups for combining loss errors. We explore linear loss error combination with manual tuning, as well as linear/non-linear error combination using implicit differentiation for automatic tuning. We also examine the impact of the number of auxiliary tasks employed in the training process.

3.2.2 Input data augmentation

Data augmentation is a technique used to increase the size and diversity of a dataset [120, 119]. In input data augmentation, additional data is generated by applying various transformations to the original input data, such as rotation, scaling, cropping, or adding noise. Doing this provides more examples to learn from and can help the trained model to generalize better on new data. Researchers studied the use of geometrical transformations for input data augmentation on neuromorphic data for training SNNs. Using this approach, allowed for about 4% accuracy increase [10]. This illustrates one of the problems of SNNs, namely, scarcity of event-based data for their training. In this paper, in addition to input data augmentation, we use AL as a method to increase accuracy on limited size training data.

3.3 Methods

3.3.1 Problem Definition

Consider an input space X, where $X \in \mathbb{R}^n$, and a main task T_{main} and one or more auxiliary tasks $T_{aux}^{(i)}$. The expected output for the main task is Y_{main} and for the auxiliary tasks Y_{aux}^{i}. We want to train a spiking neural network, $f(x)$, with weights W that minimize loss of T_{main} while using $T_{aux}^{(i)}$ as a regularization method during training. Note that $T_{aux}^{(i)}$ is used during training only.

3.3.2 Architecture

The auxiliary learning architecture for training SNNs is shown in Figure 8. It consists of a feature extraction block connected in a feed-forward fashion to the main task and auxiliary task(s) blocks. The spiking input signal is processed by the first block, the feature extraction block, into a latent p-dimensional spiking feature vector, which is then fed to the main and auxiliary task classifier blocks to find the outputs. The idea behind this architecture is to allow the feature extraction block receive feedback from the main classifier block (main task loss) and also from the auxiliary task classifier block(s) (auxiliary task losses) during training. In this way, the auxiliary task classifier blocks act as regularization blocks for the feature extraction block.

In this work as the spiking neuron model, we use the parametric leaky integrate and fire neuron model (PLIF) [73], which is a LIF neuron with learnable time constants 2.21.

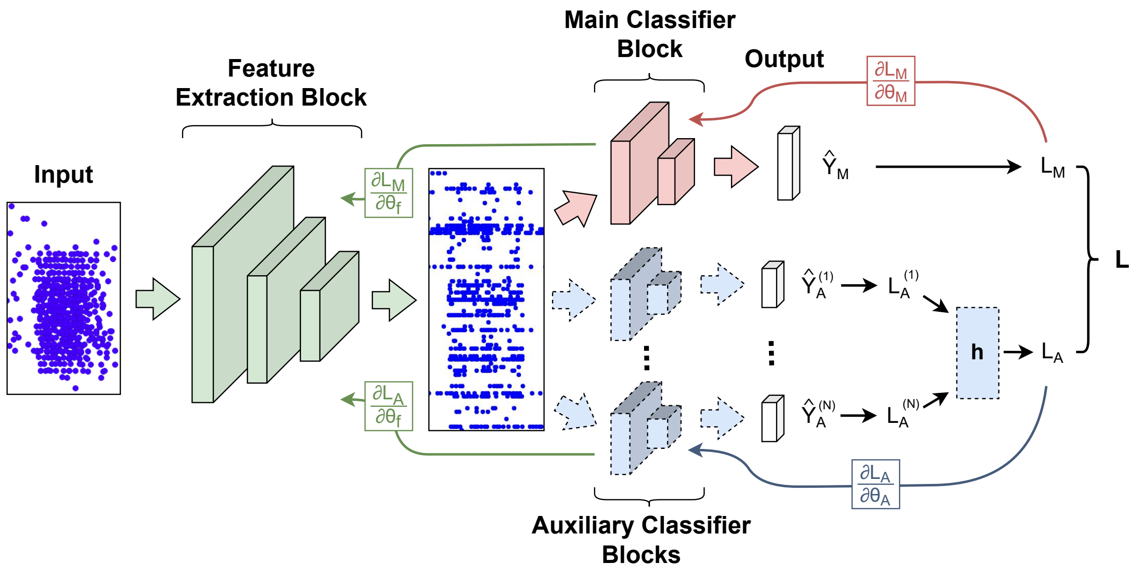

Main Classifier Block
Output
Feature Extraction Block
Input
$\frac{\partial L_M}{\partial \theta_M}$
$\frac{\partial L_M}{\partial \theta_f}$
$\hat{Y}_M$
L_M
L
$\hat{Y}_A^{(1)}$
$L_A^{(1)}$
h
L_A
$\hat{Y}_A^{(N)}$
$L_A^{(N)}$
$\frac{\partial L_A}{\partial \theta_f}$
$\frac{\partial L_A}{\partial \theta_A}$
Auxiliary Classifier Blocks

Fig. 8. Auxiliary learning architecture. The network uses a multitask architecture in which only one task, "the main task", is of importance. The other tasks, "the auxiliary tasks", are used as additional regularization losses for helping the main task performance. The auxiliary tasks are only used during training.

3.3.3 Training and Testing

The goal is to learn a set of weights, W^*, that minimizes the loss of the main task while utilizing the auxiliary losses as regularization parameters. This can be expressed as the following optimization problem:

$$W^* = \underset{W}{\operatorname{argmin}}\, L \tag{3.1}$$

where L represents the total loss, which is calculated from the main task loss, L_M, and the auxiliary task losses, $L_A^{(i)}$, as follows:

$$L = L_M + h(L_A^{(1)}, L_A^{(2)}, ..., L_A^{(i)}) \tag{3.2}$$

where L_M is the main task loss; L^i_{aux} are the auxiliary task losses; h is a linear/non-linear operation that processes the auxiliary losses. The simplest loss combination case is when h(.) is a linear combination of the auxiliary losses. In this scenario, the total loss, L, can be expressed as:

$$L = (1 - \alpha) * L_M + \alpha * \sum_{i=0}^{N} \gamma_i L_A^{(i)} \tag{3.3}$$

where α is a loss rate constant that controls the rate between the main and auxiliary losses; and γ_i denotes weights assigned to each auxiliary loss, and they can be determined through manual tuning methods like grid search, or automatic tuning methods like implicit differentiation [126]. The latter approach can also be used to train function $h(.)$ when a non-linear model is chosen. In this work, we compare the results obtained by all three methods: the manual tuning of a linear combination, automatic tuning of a linear model, and automatic tuning of a non-linear model for $h(.)$.

During testing, the samples are only fed into the feature extraction block and then to the main task classifier block. The auxiliary task blocks are not used since the focus is solely on evaluating performance of the main task.

3.4 Experiments and results

We evaluate effectiveness of AL in SNN for solving recognition tasks using CIFAR10-DVS [15], and DVS128-Gesture [16] neuromorphic datasets. All tests are performed using the architecture shown in Figure 8. For structuring the network we used the VGG like architecture [127]. The number of layers used for the feature extraction and classifier blocks for each dataset are shown in Table 2. Each layer of the feature extraction block is composed of PLIF neurons in a convolutional layer with batch normalization that is followed by max pooling with kernel 2x2. All convolution operations use kernel size of 3x3 with stride 1 and padding 1. The number of channels for all convolution layers is 128. The layers of the classifier blocks (the main and auxiliary) are composed of a fully connected layer of PLIF neurons with dropout 0.5. The number of features of the first fully connected layer is set to 1/4 of the number of input vector features. The number of features for the output layer (the last fully connected layer) is 10 times the number of classes as the average voting with stride 10 is used for computing the classification label. All results are presented as the average of ten runs.

Table 2. Network architecture used for analyzing DVS-CIFAR10 and DVS128-Gesture neuromorphic data.

Dataset	Number of layers per block	
	Feature extraction	Main/Auxiliary classifier
DVS-CIFAR10	4	2
DVS128-Gesture	5	2

3.4.1 Training with one auxiliary task

First, we test performance of training SNN with just one auxiliary task. For each dataset, we test three different auxiliary task configurations. The labels used for the main (M) and auxiliary (A) tasks are shown in Table 3.

For DVS-CIFAR10 data, A1 is selected as a duplicate of the main task label; A2 is categorization into living vs non-living class labels; and A3 is based on morphological properties of the classes. For example, deer and horse are put into the same group (group 4) because of morphological similarity. For DVS128-Gesture data, A1 is again a duplicate of the main task; while A2 and A3 are two different categorization tasks

Table 3. The main task (M) and auxiliary tasks (A1, A2, A3) configurations.

CIFAR10-DVS					DVS128-Gesture				
Class	M	A1	A2	A3	Class	M	A1	A2	A3
Airplane	0	0	0	0	Hand clapping	0	0	1	0
Automobile	1	1	0	1	Right hand wave	1	1	3	1
Bird	2	2	1	2	Left hand wave	2	2	2	2
Cat	3	3	1	3	Right arm clockwise	3	3	3	1
Deer	4	4	1	4	Right arm counter clock	4	4	3	1
Dog	5	5	1	3	Left arm clockwise	5	5	2	2
Frog	6	6	1	5	Left arm counter clock	6	6	2	2
Horse	7	7	1	4	Arm roll	7	7	0	0
Ship	8	8	0	0	Air drums	8	8	0	0
Truck	9	9	0	1	Air guitar	9	9	4	3
					Other gestures	10	10	5	4

based on morphological properties of the images.

For the above cases only a linear combination loss (Equation 3.3) is used. We test different values of the loss rate constant α. Specifically, we use α values 0.1, 0.2, 0.3, 0.4 and 0.5. Tables 4 and 5 show accuracy for the two datasets while using different auxiliary tasks and loss rate constants. Accuracy is recorded after 250 training epochs for a validation set randomly selected from the training set, with a size equal to 10% of the size of the training set.

Both data augmentation and auxiliary learning improve accuracy of the SNN. Data augmentation results in more significant increase of performance, while the utilization of auxiliary learning further improves the performance achieved through

Table 4. Accuracy for DVS-CIFAR10 dataset using auxiliary learning for 250 training epochs.

Model	Accuracy for CIFAR10-DVS (250 epochs) [%]		
	A1	A2	A3
ST-SNN	72.24 ± 0.35	72.24 ± 0.35	72.24 ± 0.35
ST-SNN + aug	80.83 ± 0.70	80.83 ± 0.70	80.83 ± 0.70
AL-SNN + aug + α=0.1	80.98 ± 0.38	**81.00 ± 0.40**	80.78 ± 0.31
AL-SNN + aug + α=0.2	81.60 ± 0.55	80.35 ± 0.71	81.02 ± 0.67
AL-SNN + aug + α=0.3	81.38 ± 0.47	79.45 ± 0.70	**81.13 ± 0.58**
AL-SNN + aug + α=0.4	81.00 ± 0.49	78.90 ± 0.39	81.05 ± 0.42
AL-SNN + aug + α=0.5	**81.75 ± 0.33**	78.72 ± 0.44	80.85 ± 0.81

Table 5. Testing accuracy for DVS128-Gesture dataset using auxiliary learning (250 training epochs).

Model	Accuracy for DVS128-Gesture (250 epochs) [%]		
	A1	A2	A3
ST-SNN	96.07 ± 0.27	96.07 ± 0.27	96.07 ± 0.27
ST-SNN + aug	98.32 ± 0.31	98.32 ± 0.31	98.32 ± 0.31
AL-SNN + aug + α=0.1	98.50 ± 0.33	98.55 ± 0.37	98.44 ± 0.28
AL-SNN + aug + α=0.2	98.50 ± 0.26	98.50 ± 0.33	98.61 ± 0.28
AL-SNN + aug + α=0.3	98.67 ± 0.37	**98.73 ± 0.26**	**98.61 ± 0.17**
AL-SNN + aug + α=0.4	98.44 ± 0.33	98.61 ± 0.20	98.32 ± 0.13
AL-SNN + aug + α=0.5	**98.67 ± 0.13**	98.38 ± 0.16	98.55 ± 0.31

data augmentation alone. It is worth noting that there is a decline in performance when using task A2 for CIFAR10-DVS data. This decrease can be attributed to the fact that auxiliary tasks should find useful information to facilitate learning. Apparently A2 does not provide such information for the network since living vs non-living categorization is based on very abstract concept that the network is not able to handle.

Regarding the choice of the loss rate constant, higher values (greater than 0.3) yield better results (except for case A2 for CIFAR10-DVS data). However, difference in performance is not clear-cut, making manual selection of this parameter quite challenging. Because of this, we use an automated method for selecting the loss rate constant; it is described in Subsection 3.4.3.

3.4.2 Training with more than one auxiliary task

Table 6 shows testing accuracies of AL with two (AL-SNN-2T), three (AL-SNN-3T), and four (AL-SNN-4T) auxiliary tasks. The first three auxiliary tasks are the same classification tasks as in Table 3. The fourth auxiliary task is randomly generated as a four-label classification. For convenience of comparisons, the results for ST-SNN and the best results for AL-SNN trained with one auxiliary task, repeated from Tables 4 and 5, are also shown.

Observe that training with more auxiliary tasks did not yield better results compared to using a single auxiliary task. The process of determining appropriate selection of auxiliary tasks, their respective weights, and choosing a proper combination of loss rate becomes highly challenging, rendering manual grid search infeasible. In our test, uniform combination of weights of 1 for all auxiliary losses and a combination loss rate of 0.5 which, as seen from the results, is not the optimal choice. Given the complexities involved in manual combination of multiple auxiliary tasks,

Table 6. Accuracy for DVS-CIFAR10 and DVS128-Gesture datasets using multiple auxiliary tasks - 250 training epochs.

Model	Validation accuracy - 250 epochs (%)	
	CIFAR10-DVS	DVSGesture128
ST-SNN	72.24 ± 0.35	96.07 ± 0.48
ST-SNN + aug	80.83 ± 0.70	98.32 ± 0.31
AL-SNN + aug	$\mathbf{81.75 \pm 0.33}$	$\mathbf{98.73 \pm 0.26}$
AL-SNN-2T + aug	80.22 ± 0.64	98.38 ± 0.31
AL-SNN-3T + aug	80.67 ± 0.24	98.67 ± 0.24
AL-SNN-4T + aug	80.77 ± 0.54	98.73 ± 0.33

an automated method for combining them becomes essential to effectively leverage its strength, which is described next.

3.4.3 Using implicit differentiation

Here we use implicit differentiation to train a loss combination function, h, such that L is minimized (Equation 3.2). Table 7 shows testing accuracies of training AL using all four auxiliary tasks and implicit differentiation. h is tested for both linear (AL-SNN-IDL-4T) and non-linear (AL-SNN-IDNL-4T) cases. Traditional ANN with three hidden layers is used for the non-linear case.

Observe that employing automatic differentiation with a non-linear function h yields the best overall result. When a linear function h is used, the obtained result is very close to the best outcome achieved through manual grid search. These findings show that automatic differentiation not only mitigates the challenges associated with manual grid search but also improves the SNN system performance. It is important to

Table 7. Accuracy for DVS128-Gesture dataset using implicit differentiation on validation set - 250 training epochs.

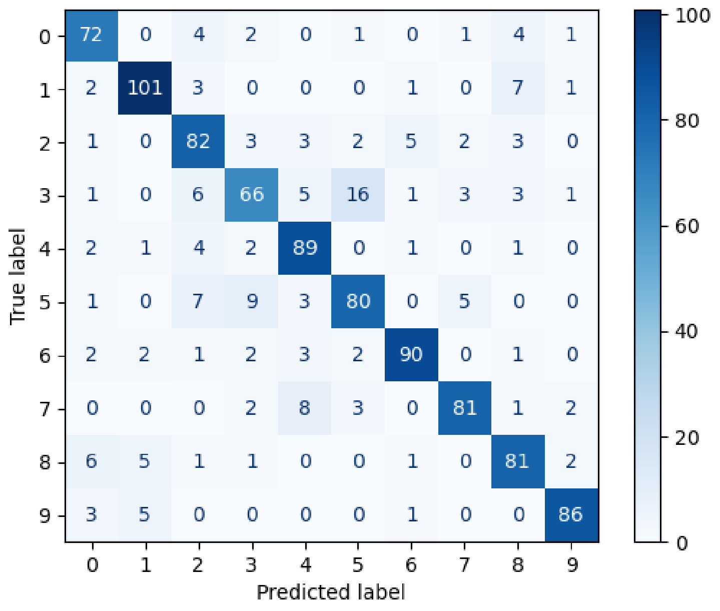
Model	Accuracy after 250 epochs (%)	
	CIFAR10-DVS	DVSGesture128
ST-SNN	72.24 ± 0.35	96.07 ± 0.48
ST-SNN + aug	80.83 ± 0.70	98.32 ± 0.31
AL-SNN + aug	**81.75 ± 0.33**	98.73 ± 0.26
AL-SNN-IDL-4T + aug	81.15 ± 0.27	98.67 ± 0.24
AL-SNN-IDNL-4T + aug	81.69 ± 0.34	**98.84 ± 0.39**

highlight that A4 is a random task that does not provide any useful information, yet automatic differentiation successfully handles this task. This underscores robustness and adaptability of automatic differentiation in effectively handling diverse tasks, even when they apparently do not provide additional information.

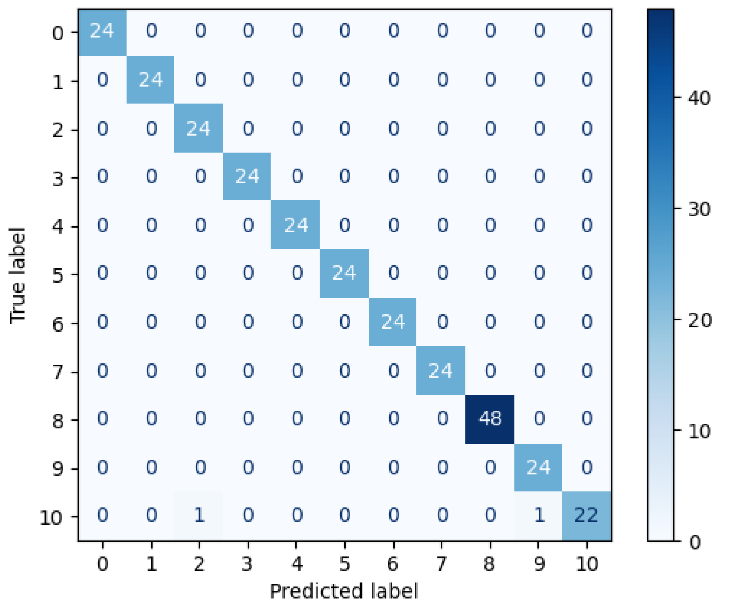
3.4.4 Comparison with State-of-the-Art SNNs

The proposed training approach using auxiliary learning with state-of- the-art methods, using SNN on the CIFAR10-DVS and DVSGesture128 neuromorphic datasets, is compared. To identify the best trained networks, we conduct an analysis using precision, recall, and F1-score. We then select the top-performing network for each dataset. Figure 9 shows the confusion matrix for the selected networks and Table 8 shows the above performance indicators. Results are shown for 1024 training epochs on the testing set.

Overall, SNN trained using auxiliary learning exhibits a well-balanced performance in predicting labels for each dataset. It is worth to highlight a particular case,



(a) CIFAR10-DVS



(b) DVSGesture-128

Fig. 9. Confusion matrix for best performing SNN with AL for CIFAR10-DVS (a) and DVSGesture128 (b) datasets.

Table 8. Accuracy, precision, recall and F1 score for best performing SNN with AL for CIFAR10-DVS and DVSGesture128 datasets.

Dataset	Model	Accuracy	Precision	Recall	F1-Score
CIFAR10-DVS	AL-SNN + aug + α=0.5	82.80	0.829	0.828	0.827
DVSGesture128	AL-SNN-IDNL-4T + aug	99.31	0.993	0.993	0.993

which is the prediction of class 3 (cat) for CIFAR10-DVS data. This specific class is the most challenging to predict in the CIFAR10-DVS dataset.

We compare the obtained results with state-of-the-art SNNs, which is shown in Table 9.

Notice that training with auxiliary learning achieves the highest accuracy for DVSGesture128 dataset and the second highest for CIFAR10-DVS. The highest accuracy for CIFAR10-DVS is achieved by AIA, which is a SNN that uses a more advanced neuron model than the PLIF neuron model used in this work. In fact. we see that, training with AL achieves higher accuracy when compared with SNN that uses PLIF neurons (PLIF and NDA). We expect that AL with AIA neuron model would achieve the best performance.

3.5 Discussion and Future Work

In this chapter we presented the usage of auxiliary learning, in addition to data augmentation, to improve performance of SNNs. The used network architecture consists of a feature extraction block connected in a feedforward fashion to a main classification block and one or more auxiliary task classification blocks. By using auxiliary tasks, we use additional information during training that helps in regularization of the feature extraction block. As a result, the feature extraction block is forced to

Table 9. Comparison with staet-of-the-art SNNs for CIFAR10-DVS and DVSGesture-128 datasets.

Model	Reference	CIFAR10-DVS	DVSGesture-128
STBP [128]	AAAI 2021	67.80	96.87
PLIF [73]	ICCV 2021	74.80	97.57
Dspike [129]	NeurIPS 2021	75.40	-
AutoSNN [130]	ICML 2022	72.50	96.53
RecDis [131]	CVPR 2022	72.42	-
DSR [132]	CVPR 2022	77.27	-
NDA [10]	ECCV 2022	81.70	-
SpikeFormer [133]	ICLR 2023	80.90	98.30
AIA [134]	ICASSP 2023	83.90	-
AL-SNN (ours)	-	82.80	99.31

learn more general and robust features which helps improving SNN network performance on the main task. The results confirm that using AL during training indeed results in improved performance. Moreover, the experiments demonstrate that the extent of improvement depends on careful tuning combination of loss rate parameters. To overcome this challenge, we used automatic differentiation [126] to automatically adjust the loss combination parameters. Note that all the experiments presented in this study were conducted through simulation using the SpikingJelly neuromorphic library. However, in the future we plan to leverage Intel's Lava framework, which enables to directly deploy the network on the Loihi2 neuromorphic chip.

CHAPTER 4

MULTI-TASK LEARNING WITH FIRING THRESHOLD MODULATION

"Neuromorphic approaches and conventional machine learning should not be considered simply two solutions to the same classes of problems, instead it is possible to identify and exploit their task-specific advantages" [6].

In this chapter we present a SNN that can learn multiple tasks in a way that is unique only to them, namely, their behavior can be changed based on modulation of its firing thresholds. Specifically, we train a network to solve multiple classification tasks performing only one at a time. The task to be performed is determined by changing the spiking neuron's firing threshold: with one threshold the network learns one task, with the second threshold another task, and so on. The proposed SNN was implemented on Intel's Lava platform and tested on the Loihi2 neuromorphic chip [20]. Results for multitask classification on neuromorphic NMNIST data [68] show that SNN can effectively learn different tasks through modulation of the neurons' firing thresholds. The proposed network constitutes to our best knowledge the first implementation of training threshold modulated SNN.

The rest of the chapter is structured as follows. Section 4.1 presents relevant work and introduces multitask learning using firing threshold modulation. Section 4.2 defines the problem to be solved and describes the network architecture. Section 4.3 presents simulation results of using threshold modulation for solving multitask classification for the NMNIST data. Section 4.4 presents results of running TM-SNN on the Loihi2 neuromorphic chip. The chapter finishes with discussion and future work.

The following papers related to this chapter were published:

1. Paolo G. Cachi, Sebastian Ventura, and Krzysztof J. Cios, “Implementing Threshold Modulated Spiking Neural Networks on Loihi2 Neuromorphic Chip”, under review.

2. Paolo G. Cachi, Sebastian Ventura, and Krzysztof J. Cios, “TM-SNN: Threshold Modulated Spiking Neural Network for Multi-task Learning,” in 17th International Work-Conference on Artificial Neural Networks (IWANN2023), 2023.

3. Paolo G. Cachi, Soumil Jain, Sebastian Ventura, Gert Cauwenberghs, and Krzysztof J. Cios, “Reproducing Aplysia R-15 Bursting Neurodynamics on a Neuromorphic Microchip,” in 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 1–4, 2022.

4.1 Introduction

Multi-task learning (ML) is a machine learning problem in which a model is trained to solve more than one task [123, 124]. The goal is to improve the model’s generalization ability by learning tasks in a shared feature space. This can be useful when there is a significant amount of shared information between the tasks, as it allows the model to learn shared features rather than learning them separately for each task. For example, a multi-task learning model trained to classify two different object datasets, such as CIFAR10 and ImageNet, might learn to recognize edges and basic shapes that are useful for both tasks. This can lead to a more efficient and effective model, as it reuses features learned from one task to improve performance on other tasks [122, 13]. Learning multiple tasks, however, encounters problems such as negative transfer, which happens when different tasks have conflicting goals like when increasing performance for one task decreases performance for the other(s) [125, 135].

Several solutions were proposed to deal with the negative learning problem [136, 137, 138, 17]. The one of interest here is [17], where the authors solved the multi-task learning problem using an approach called single tasking of multiple tasks. It consists of training a ANNs to solve more than one task but doing only one task at a time. To implement it, they used attention-like mechanisms with adversarial loss for training a feed-forward neural network that learns task-specific features. In other words, attention-like mechanism is used by the network to select different set of features. That is, the network makes use of different internal pathways for processing each task independently, which mitigates the negative transfer problem.

Inspired by the above described solution, we propose using SNNs to implement single tasking of multiple tasks. However, instead of controlling the behavior of the network by using attention mechanisms, we use neuromodulation. Neuromodulation is the property of spiking neurons to modify their intrinsic behavior based on the presence of external stimuli [18]. Specifically, we construct a network that can switch its behavior, namely the classification task to perform is based on the modulation of the spiking neurons' firing threshold. We refer to the proposed network as threshold modulated spiking neural network (TM-SNN). Its architecture, shown in Figure 10, consists of three blocks. Each block (described in detail later) is built of one or more spiking neuron layers connected in a feed-forward fashion. For training TM-SNN, we use the SLAYER backpropagation algorithm that was developed to work with SNN [19]. TM-SNN is implemented in Intel's Lava neuromorphic framework that allows for its direct deployment on the Loihi2 neuromorphic chip. Experiments for multi-task classification on NMNIST data [68] are performed.

4.2 Methods

4.2.1 Problem Definition

In the setting of single tasking of a multi-task problem, we assume an input space X, where $X \in \mathbb{R}^n$ and a set of two (or more) classification labels $Y^{(1)}$ and $Y^{(2)}$, where $Y^{(1)} = \{y_1^{(1)}, y_2^{(1)}, ..., y_m^{(1)}\}$ and $Y^{(2)} = \{y_1^{(2)}, y_2^{(2)}, ..., y_p^{(2)}\}$. We plan to construct a SNN, F, with weights W and an internal parameter (in our case the firing threshold) φ that learns the transformations: $y_i^{(1)} = F(x_i \mid W, \varphi = \varphi_1)$ and $y_i^{(2)} = F(x_i \mid W, \varphi = \varphi_2)$.

4.2.2 Architecture

TM-SNN architecture is shown in Figure 10. It consists of three spiking neuron blocks connected in a feed-forward fashion, similar to [139]. The spiking input signal is processed by the first block - the feature extraction block - into a latent p-dimensional spiking feature vector, which is then used to assign the multi-task labels using a label classifier block. A task classifier block is used for learning the specific task that is being performed. The idea behind this three-block architecture is to allow the feature extraction block receive training feedback not only from the label classifier block (classification loss) but also from the additional task classifier block (task loss). The task classifier block is used as an auxiliary block in a similar way as in Chapter 3. The task classifier block is not used during testing. Note that in contrast to the architecture proposed in [139], TM-SNN does not use a gradient reversal layer before the task classifier block. This is because we want the feature extraction block to learn specific feature vectors for each classification task rather than a common feature vector as done in [139].

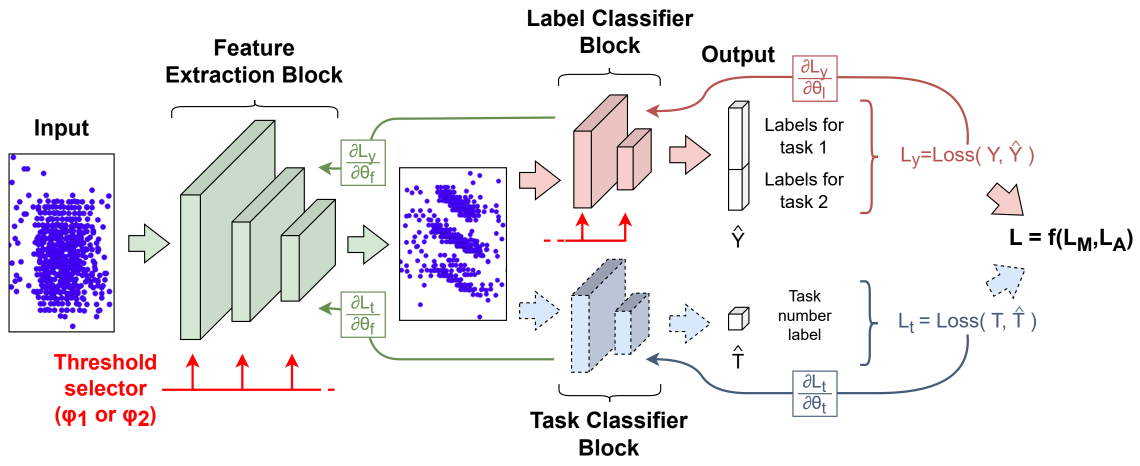

Label Classifier Block
Feature Extraction Block
Output
Input
$\frac{\partial L_y}{\partial \theta_l}$
$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_t}{\partial \theta_f}$
$\frac{\partial L_t}{\partial \theta_t}$
Labels for task 1
Labels for task 2
L_y=Loss(Y, $\hat{Y}$)
$\hat{Y}$
L = f(L_M,L_A)
Task number label
$\hat{T}$
L_t = Loss(T, $\hat{T}$)
Threshold selector (φ_1 or φ_2)
Task Classifier Block

Fig. 10. TM-SNN architecture. It consists of three processing blocks connected in a feed-forward fashion: a feature extraction block and two classifier blocks. The label classifier outputs the labels for task 1 or task 2 (or more). The task classifier is used as a regularization mechanism to aid the feature extraction block learn a set of independent features for each task.

4.2.3 Training and testing

The goal of training is to learn weights, W, that predicts task 1 with firing threshold $\varphi = \varphi_1$ and task 2 (or more tasks) when $\varphi = \varphi_2$. To achieve this, TM-SNN is trained for both tasks concurrently using a per-batch task selection process. Before each batch sample presentation, a task to train TM-SNN for is selected randomly. If task 1 is selected, then the firing threshold of the feature extraction block and the label classifier block is set to $\varphi = \varphi_1$ and for task 2 it is $\varphi = \varphi_2$. After setting the firing threshold, the training process is done using the spike-based backpropagation SLAYER algorithm [19]. The backward process is set to minimize both the label classifier and the task classifier loss functions. The combined loss, L, is calculated as a simple linear combination:

$$L = (1-\gamma) * L_y + \gamma * L_t \tag{4.1}$$

where L_y is the loss for the label classifier block given by $L_y = Loss(Y, \hat{Y})$; L_t is the loss for the task classifier block given by $L_t = Loss(T, \hat{T})$; and γ is a loss rate constant that controls the rate between the label and task classifier losses. The true labels for the label classifier block, Y, are constructed as a concatenation of Y_1 and $Y_2 = 0$ or $Y_1 = 0$ and Y_2, depending on whether task 1 or task 2 was selected. The task classifier block predicts 0 when trained for task 1 or 1 when trained on task 2 data. Note that the firing threshold is not changed for the task classifier block. This is because the goal is for the task classifier block to backpropagate the same information to the feature extraction block regardless of which task is being learned.

For testing, first, the firing threshold φ_1 or φ_2 is set depending on which task is tested. After that, the samples are input only to the feature extraction block and to the label classifier block. The task classifier block is not used as it is already determined by the chosen firing threshold.

4.3 Simulation Performance of TM-SNN

Performance of TM-SNN is analyzed on the neuromorphic NMNIST data (60K training and 10K testing samples) [68] using Intel's Lava Framework. Five types of experiments are performed. First, training and testing performance of TM-SNN using different threshold values is reported, see Figure 11 and Table 10. Here we don't use the task classifier block since our aim is to assess the effects of selecting different threshold values. Second, the influence of including the task classifier block in training is analyzed, see Table 11. Third, the results of TM-SNN operating as described above are compared with TM-SNN that uses the external input current (not the threshold) to control its behavior, see Table 12. Fourth, the ability of TM-SNN to learn more than two tasks is assessed, see Table 13. Fifth, we estimate and compare computational efficiency of TM-SNN with ANNs.

4.3.1 Varying threshold

Figure 11 shows accuracy of TM-SNN for two-task classification problem on the NMNIST data using different thresholds. Task 1 is the digit classification with 10 labels, and task 2 is the odd/even digit classification with 2 labels. Figure 11 also shows the results for a single-task SNN, called ST-SNN, which was separately trained only on task 1 or only on task 2, to establish a base case. The network architecture for both TM-SNN and ST-SNN is essentially the same. It consists of two layers of 512 spiking neurons in the feature extraction block and two layers of 128 and 12 spiking neurons in the label classifier block. Note that φ_1 is set to 1.25 in all tests while φ_2 varies from 1.5 to 10. The constant φ_1 value is used to tune spiking neurons to operate in a normal operation mode (single tasking).

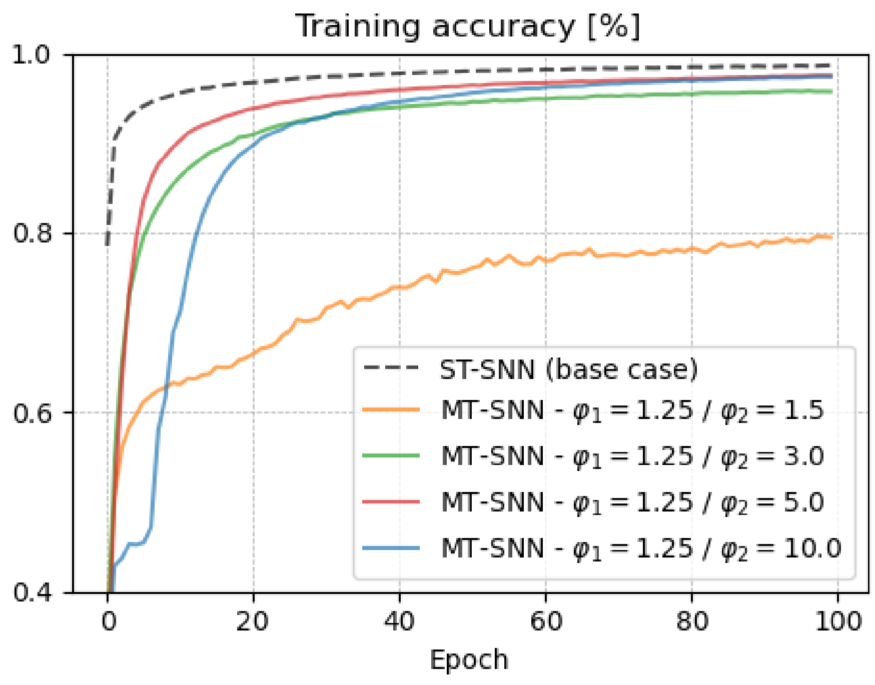


Fig. 11. Training accuracy of ST-SNN (base case) and of TM-SNN using different threshold values: φ_1 is set to 1.25 while φ_2 changes from 1.5 to 10.

Notice in Figure 11 that using $\varphi_1 = 1.25$ and $\varphi_2 = 5.0$ results in performance close to the base case scenario (when ST-SNN is trained on task 1 only). Using values for φ_1 and φ_2 close to each other ($\varphi_1 = 1.25$ and $\varphi_2 = 1.5$) achieves results in lower accuracies than the base case. On the other hand, using values that are too far apart (like $\varphi_1 = 1.25$ and $\varphi_2 = 10$) causes longer training times for TM-SNN (see blue line in Figure 11).

Table 10 compares testing accuracy for both tasks for different firing threshold pairs, after 100 epochs. It also shows accuracy of the ST-SNN (base case).

Table 10. Testing results of TM-SNN using different firing threshold values.

Model	Test accuracy (%)	
	Task 1	Task 2
ST-SNN (base case)	***98.93***	***99.34***
TM-SNN - $\varphi_1 = 1.25$, $\varphi_2 = 1.5$	91.98	96.09
TM-SNN - $\varphi_1 = 1.25$, $\varphi_2 = 2.0$	95.50	98.51
TM-SNN - $\varphi_1 = 1.25$, $\varphi_2 = 3.0$	96.59	98.90
TM-SNN - $\varphi_1 = 1.25$, $\varphi_2 = 5.0$	97.80	**99.11**
TM-SNN - $\varphi_1 = 1.25$, $\varphi_2 = 10.0$	**97.85**	99.01

Two conclusions can be drawn from Table 10 results. First, similar to the training performance (shown in Figure 11) TM-SNN performs better when the difference between φ_1 and φ_2 increases. Second, the best accuracies on both tasks are lower than the accuracies of the base case, which is typical when solving multi-task problems. However, in order to improve this performance, we use the task classifier block as well as more tasks, which results are described in the following subsection.

It is also informational to compare TM-SNN spiking outputs for each firing threshold, which is shown in Figure 12 for thresholds $\varphi_1 = 1.25$ and $\varphi_2 = 5$ values.

Observe a drastic change in the output when the firing threshold is changed. Specifically, when $\varphi_1 = 1.25$, neuron 3 (corresponding to digit class 3) exhibits the highest activity, while when $\varphi_1 = 5$, neuron 11 (corresponding to odd-numbered class) shows the highest activity. Also notice that the overall firing rate of the output neurons for $\varphi_1 = 1.25$ is higher than for $\varphi_2 = 5$.

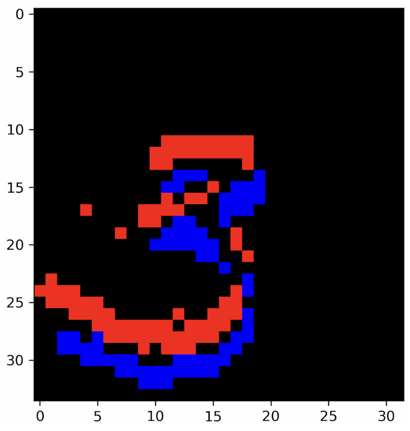
4.3.2 Using the task classifier block in training

The task classifier block is used to decrease the loss function value, which is inherent in multi-task problems. Table 11 compares accuracy when using task classifier block during training. Results are shown for the loss constant γ (Equation 4.1) values equal to 0.5, 0.4, 0.3, 0.2 and 0.1. All tests are done using $\varphi_1 = 1.25$ and $\varphi_2 = 5$ values. For convenience of the reader, the results for ST-SNN and TM-SNN (repeated from Table 10) are also shown (two first rows).

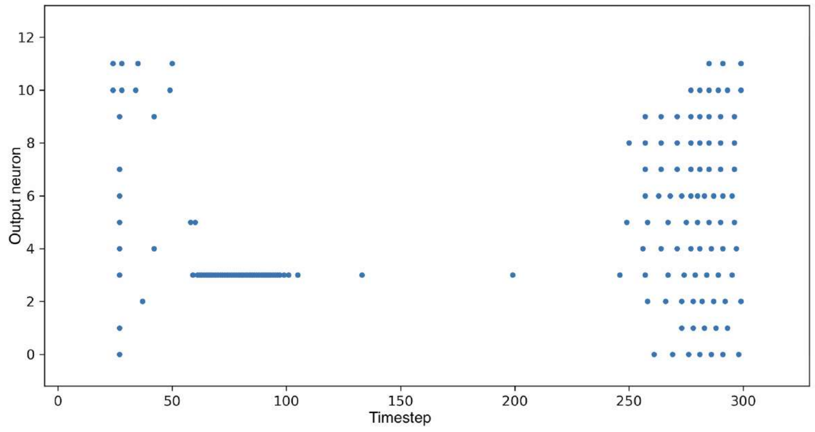
We see that the addition of the task classifier block slightly increased accuracy of task 1 by 0.18% and by 0.20% on task 2, both for $\gamma = 0.3$. This small increase can be attributed to the fact that the task classifier is very simple (only two labels). Note that the task classifier block reaches a plateau very close to 100% accuracy after training for only 20 epochs.

4.3.3 Use of a firing threshold vs using an external input current

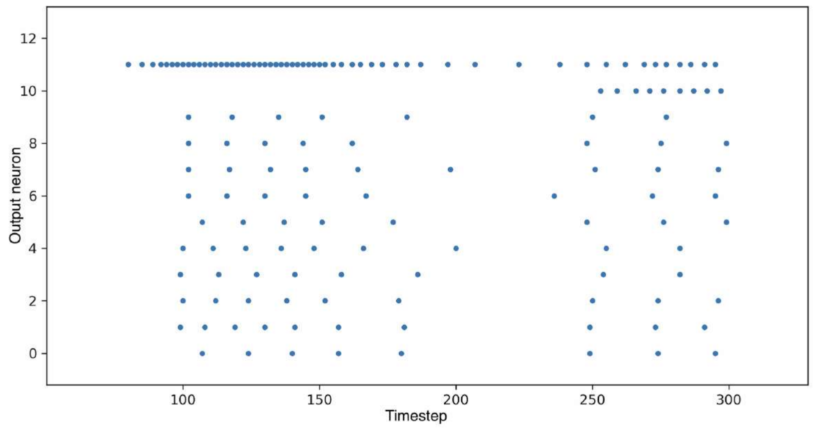
Table 12, shows accuracy of a SNN that uses modulation via changing the external input current, called EC-SNN, instead of modulating firing threshold. The architecture of EC-SNN is essentially the same as TM-SNN. The training was done



(a) Input of class/digit 3



(b) Output with $\varphi_1 = 1.25$



(c) Output with $\varphi_1 = 5$

Fig. 12. Example spiking output when TM-SNN is presented with input representing digit 4 (a) with $\varphi_1 = 1.25$ threshold (b) and with $\varphi_2 = 5$ (c).

Table 11. Testing accuracy of TM-SNN when using task classifier block

Model	Test accuracy (%)	
	Task 1	Task 2
ST-SNN - Base case	***98.93***	***99.34***
TM-SNN (without task classifier)	97.80	99.11
TM-SNN / $\gamma = 0.1$	97.90	99.29
TM-SNN / $\gamma = 0.2$	97.86	99.23
TM-SNN / $\gamma = 0.3$	**97.98**	**99.31**
TM-SNN / $\gamma = 0.4$	97.70	99.18
TM-SNN / $\gamma = 0.5$	97.77	99.20

for 100 epochs using $I_{ext1} = 0$ for task 1 and I_{ext2} equal to 0.05, 0.1, 0.5, 1 and 5 for task 2.

Notice that while controlling I_{ext} the results are lower than when modifying the firing threshold of neurons. This finding suggests that firing threshold modulation outperforms external current modulation in the context of multitask learning. Furthermore, this outcome is consistent with the behavior of biological circuits during neuromodulation.

4.3.4 Learning several classification tasks at the same time

We test the ability of TM-SNN to learn more than two tasks at the same time. Table 13 shows testing accuracies of TM-SNN trained with two, three, and four tasks. The firing threshold for the first tasks is set at 1.25 and for the other tasks are 5, 10 and 15. Task 1 and task 2 are the same classification tasks from the previous

Table 12. Testing accuracy of TM-SNN when an external current I_{ext} is used to control the network operation.

Model	Testing accuracy (%)	
	Task 1	Task 2
ST-SNN (base case)	***98.93***	***99.34***
TM-SNN / $\gamma = 0.3$	**97.98**	**99.31**
EC-SNN / $I_{ext2} = 0.05$	95.63	98.06
EC-SNN / $I_{ext2} = 0.1$	96.05	97.86
EC-SNN / $I_{ext2} = 0.5$	96.07	97.66
EC-SNN / $I_{ext2} = 1.0$	95.78	97.62
EC-SNN / $I_{ext2} = 5.0$	92.20	97.95

experiments (10 digit label classification and odd/even digit classification). Task 3 is greater/less than 5 classification (2 labels), and task 4 is the modulo operation of 3 classification (3 labels). The network architecture is the same as in the previous case with the exception that the number of output neurons are changed accordingly. Table 13 also includes the single task SNN (ST-SNN) trained with each task independently for reference.

Results show that threshold modulation also works for cases involving more than two classification tasks. Interesting is the result of training TM-SNN for three tasks that resulted in higher accuracy than training for two tasks. However, the accuracy decreased when number of tasks is four.

Table 13. Testing accuracy of TM-SNN trained on four tasks.

Model	Testing accuracy (%)			
	Task 1	Task 2	Task 3	Task 4
ST-SNN (base case)	***98.93***	***99.34***	***99.01***	***98.97***
TM-SNN (2 tasks)	97.98	**99.34**	-	-
TM-SNN (3 tasks)	**98.24**	99.17	**98.84**	-
TM-SNN (4 tasks)	97.05	98.83	98.33	98.11

4.3.5 Comparison of TM-SNN with ANN

We compare computational efficiency of TM-SNN vs ANN in terms of neuron activity (number of events) and synaptic operations (SynOps). These indicators are directly proportional to the network's energy consumption. Table 14 compares results of TM-SNN and ANN with the similar architecture (4 fully connected layers of 512, 512, 128 and 14 neurons, respectively). For TM-SNN, we select the best trained network from the previous experiment, which involved training it for three different tasks. Table 14 shows the number of events and SynOps at each layer under different threshold values: $\varphi = 1.25$, $\varphi = 5$, and $\varphi = 10$. Number of activations and multiply-accumulate operations (MACs) are the corresponding indicators for computational efficiency in the ANN.

We observe that TM-SNN exhibits a higher degree event efficiency (TM-SNNs' events vs ANNs' activations) and operations sparsity (SynOps vs MACs) than ANN. This is expected since SNN rely on temporal sparse computations rather than continuous activation functions. Importantly, the degree of computational efficiency achieved by TM-SNN depends on the threshold value. Higher threshold values, such

Table 14. Events and SynOps comparison of TM-SNN vs ANN.

		TM-SNN ($\varphi = 1.25$)		TM-SNN ($\varphi = 5$)		TM-SNN ($\varphi = 10$)		ANN	
Layer	Shape	Events	SynOps	Events	SynOps	Events	SynOps	Activations	MACs
FullyCon-1	(-1, 512)	90.48	46,325.60	48.09	24,623.84	30.93	15,833.69	512	1'183,744
FullyCon-2	(-1, 512)	207.39	106,182.58	90.93	46,555.27	30.27	15,497.32	512	262,144
FullyCon-3	(-1, 128)	1.45	185.98	2.48	317.70	3.36	430.37	128	655,336
FullyCon-4	(-1, 14)	0.51	7.15	0.58	8.14	0.55	7.70	14	1,792
TOTAL		299.83	152,701.30	142.08	71,504.96	65.11	31,769.08	3,478	1'513,216
GAIN		11.60x	9.91x	24.48x	21.16x	53.42x	47.63x	1x	1x

as $\varphi = 1.25$, lead to a substantial reduction in the number of events and SynOps. This indicates that by modifying the threshold value, we control both the computational load and energy consumption. The ability to modulate the threshold provides great flexibility to regulate the network's energy requirements to specific operating conditions.

4.4 Profiling on Loihi2

The Lava framework provides the NetX functionality that enables deployment of SNN on the Loihi2 chip. In this section, we present a comparative analysis of performance of MT-SNN deployed on Loihi2 using the Lava framework. The Loihi2 chip is accessible through Intel's Oheogulch board, a platform for experimentation and evaluation of the Loihi2. By leveraging the capabilities of the Lava framework and utilizing the computational power of the Loihi2, we investigate the efficiency and effectiveness of the MT-SNN model in multitask classification.

4.4.1 Network Selection

To identify a suitable network for deployment, we analyze the top three best-performing TM-SNN (see the previous subsection where TM-SNN was trained with 3 tasks). The accuracy comparison on the validation set of these top three networks are presented in Tables 15. In addition to accuracy, we also calculate precision, recall, and F1 score which are shown in Table 16.

The precision, recall, and F1 scores of the three TM-SNN are consistently very high (close to 1). This indicates that TM-SNN demonstrates a well-balanced performance in predicting all three tasks. Among the three models, we select TM-SNN number 3 for deployment due to its overall superior performance across all tasks. It

Table 15. Testing accuracy for the top-three, on task 1, TM-SNN

Model	Testing Accuracy (%)		
	Task 1	Task 2	Task 3
1	**98.32**	98.92	98.63
2	98.29	98.99	98.70
3	98.26	**99.01**	**98.80**

achieves the highest performance on tasks 2 and 3 while maintaining only a slightly lower performance on task 1. This selection ensures the best overall performance while still considering the specific challenges posed by task 1. The confusion matrix for TM-SNN number 3 on the validation set is shown in Figure 13.

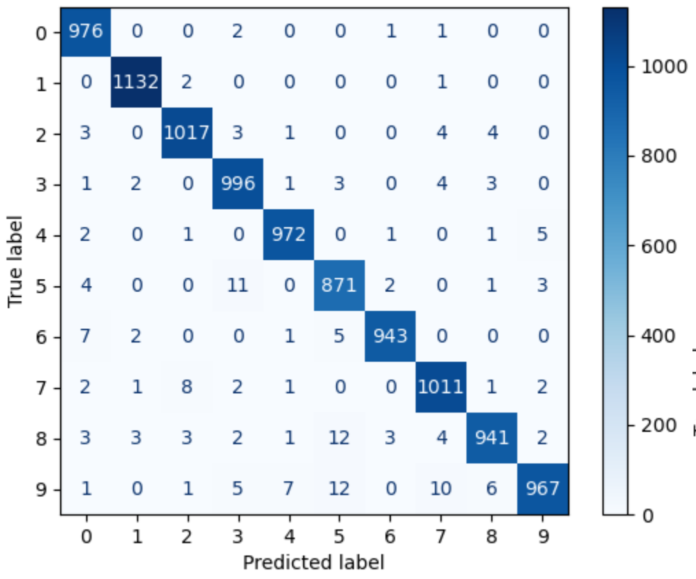
4.4.2 Execution time

Execution time in Loihi2 is measured using the execution time profiler, which measures the total and average time per step. Figure 14 shows the plots of execution time for one sample presentation using TM-SNN for the three tasks classification. While NMNIST sample consists of 300 timesteps, we extended the duration to 350 timesteps to allow the network to return to its resting state before the next sample presentation. The network used in this analysis is the top-performing network when training with three tasks using firing threshold values of 1.25, 5 and 10 (see Subsection 4.3.4).

Contrary to our expectations, the execution time per time step in Loihi2 is not as fast as anticipated. On average, each time step takes approximately 0.075 seconds, resulting in a throughput of 0.44. This relatively slow execution rate is primarily

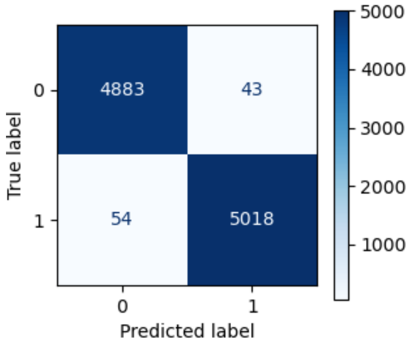
Table 16. Precision, recall and F1 score for the top-three, on task 1, TM-SNN.

Model	Precision			Recall			F1 score		
	Task 1	Task 2	Task 3	Task 1	Task 2	Task 3	Task 1	Task 2	Task 3
1	0.983	0.989	0.986	0.983	0.989	0.986	0.983	0.989	0.986
2	0.983	0.990	0.987	0.983	0.990	0.987	0.983	0.990	0.987
3	0.983	0.990	0.988	0.983	0.990	0.988	0.983	0.990	0.988



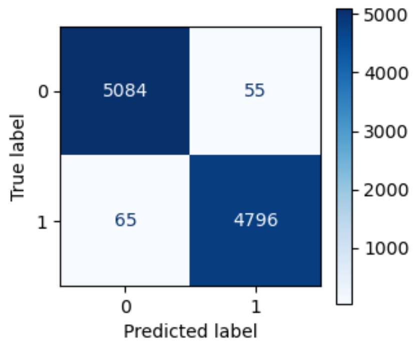
True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	976	0	0	2	0	0	1	1	0	0
1	0	1132	2	0	0	0	0	1	0	0
2	3	0	1017	3	1	0	0	4	4	0
3	1	2	0	996	1	3	0	4	3	0
4	2	0	1	0	972	0	1	0	1	5
5	4	0	0	11	0	871	2	0	1	3
6	7	2	0	0	1	5	943	0	0	0
7	2	1	8	2	1	0	0	1011	1	2
8	3	3	3	2	1	12	3	4	941	2
9	1	0	1	5	7	12	0	10	6	967

(a) Task 1



True label \ Predicted label	0	1
0	4883	43
1	54	5018

(b) Task 2



True label \ Predicted label	0	1
0	5084	55
1	65	4796

(c) Task 3

Fig. 13. Confusion matrix of TM-SNN number 3 for three task classification on NM-NIST data.

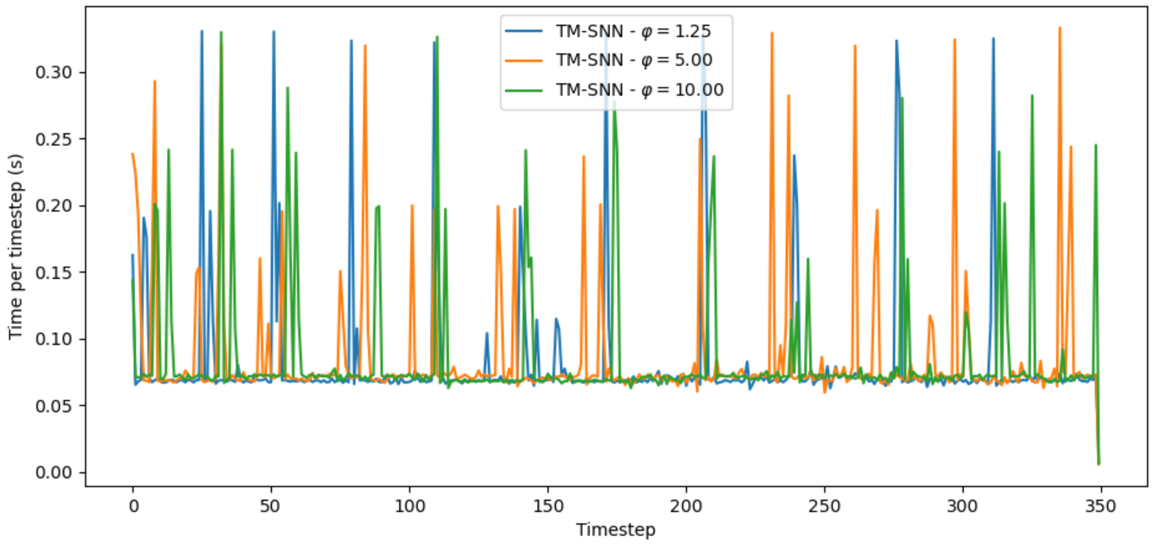


Fig. 14. TM-SNN's execution time in seconds for one sample presentation using $\varphi = 1.25$, $\varphi = 5$, and $\varphi = 10$.

attributed to a bottleneck caused by communication time between the host computer and the Loihi 2 chip when submitting input samples. However, we anticipate a significant improvement once the dedicated spike input feature of Loihi2 is integrated into Lava. Furthermore, the results do not show significant difference in execution time when varying firing threshold values. These results shed light on the current performance limitations and highlight the ongoing need to develop better solutions addressing these challenges.

4.4.3 Power consumption

Power consumption is measured using Lava's Loihi2Power profiling module. It measures power consumption across the whole Oheogulch board. Figure 15 shows power consumption during 2000 micro secs. The plot includes total power, total static power consumed when idle (static power), supply power (VDD power), memory circuit power (VDD-M power) and input output peripheral circuits power (VDD-IO).

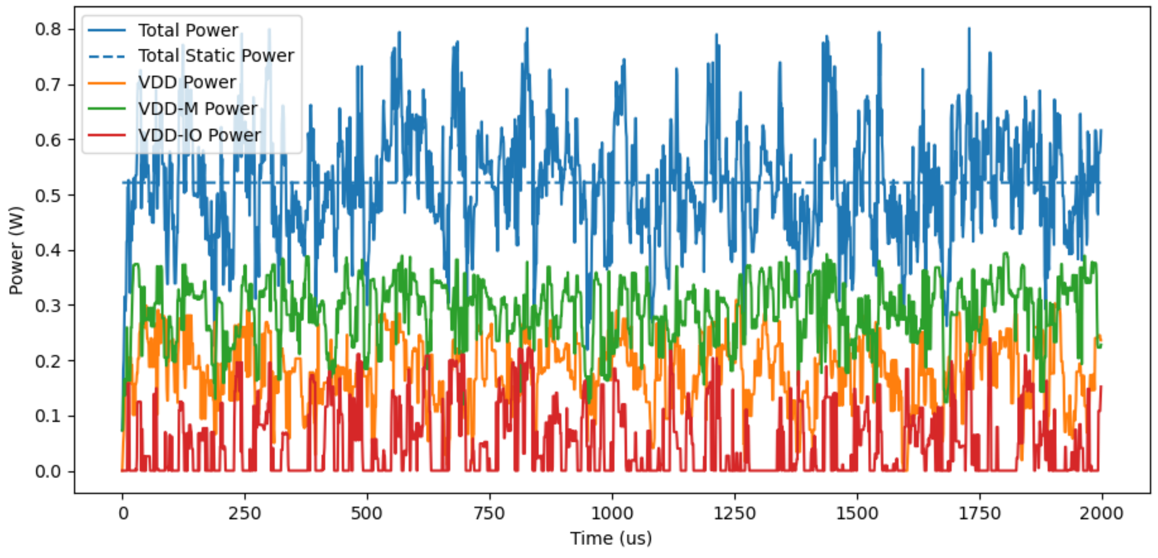


Fig. 15. TM-SNN's power consumption in watts for TM-SNN using $\varphi = 1.25$.

To evaluate the power consumption of TM-SNN for multi-task classification, we show a summary of the consumed power in Table 17. It should be noted that due to the current Loihi2 bottleneck, the differences in power consumption depending on the task being solved cannot be spotted. This is because the implementation spends more time on the communication phase between the host computer and the Loihi2 chip, which dominates the power consumption across all tasks.

4.4.4 Spiking Activity

Spiking activity in terms of synaptic operations (SynOps) in Loihi2 is measured using the Loihi2Activity profiler in Lava. Figure 16 shows the SynOps per sample for TM-SNN run on Loihi2. Similar to previous experiments, the results are shown for the three-task configuration using 200 samples.

The results obtained on Loihi2 align with the estimated computational efficiency discussed in Subsection 4.3.5. Specifically, the number of synaptic operations (Syn-

Table 17. Testing accuracy of MT-SNN trained for four tasks.

	TM-SNN Power Consumption		
	$\varphi = 1.25$	$\varphi = 5$	$\varphi = 10$
Total Power [mW]	531.86	522.34	532.93
Static Power [mW]	549.52	517.17	547.68
VDD Power [mW]	182.62	184.06	181.88
VDD-M Power [mW]	287.52	286.33	293.34
W VDD-IO Power [mW]	61.72	51.94	57.71
Total Energy [mJ/sample]	16,245.60	14,361.59	14,329.74
Dynamic Energy [mJ/sample]	539.52	141.75	396.67

Ops) used varies depending on the firing threshold. Using a low firing threshold, like $\varphi = 1.25$, the number of SynOps is relatively higher, indicating higher frequency of spikes and greater energy consumption. On the other hand, increasing the firing threshold to $\varphi = 5$ and $\varphi = 10$ leads to the decrease in the number of SynOps used. This reduction demonstrates the ability of threshold modulation to control the spiking activity, directly impacting the energy consumption requirements.

By effectively modulating the firing threshold, TM-SNN can dynamically adjust the level of spiking activity. The results obtained on Loihi2 highlight the practical application of threshold modulation in controlling spiking activity and energy requirements. This capability can be very important in resource-constrained scenarios where energy efficiency is critical.

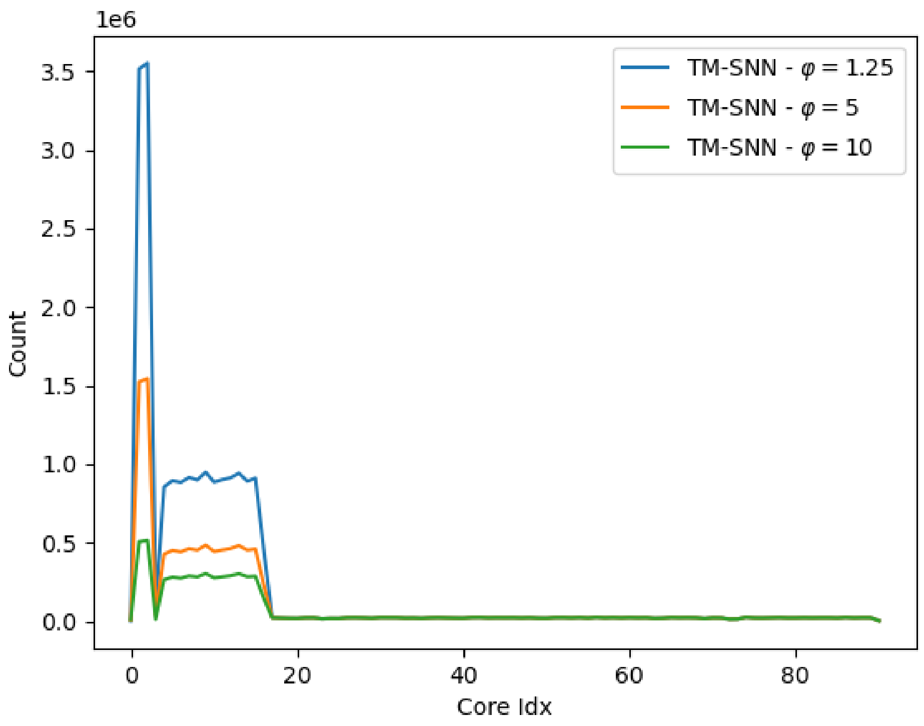


Fig. 16. Spiking activity in terms of SynOps when running TM-SNN on Loihi2 for three tasks configuration with $\varphi = 1.25$, $\varphi = 5$, and $\varphi = 10$.

4.4.5 Memory usage

Figure 17 shows the relative, per core, memory usage of TM-SNN. It is important to mention that the memory usage does not vary depending on the tested task, as the only parameter that is being changed is the firing threshold and not the network configuration (weights).

TM-SNN consumes less than half of the total memory on a Loihi2 chip. Furthermore, the memory usage plot provides insights into the efficient mapping of memory across the network cores, shows compact nature of memory allocation in Loihi2.

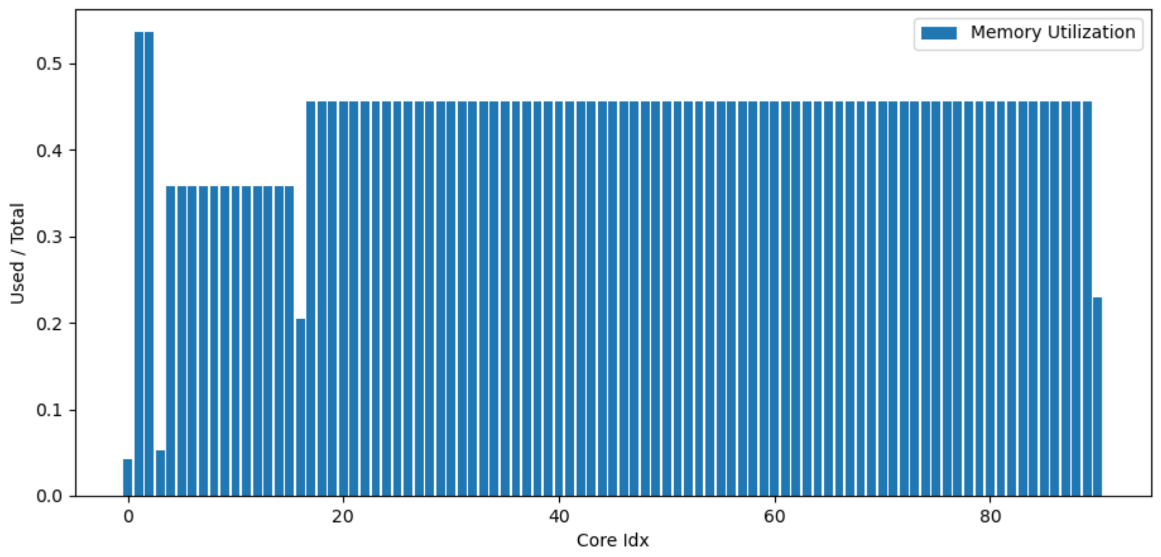


Fig. 17. Relative per core memory usage of TM-SNN on the Loihi2 neuromoprhic chip.

4.5 Discussion and Future Work

In this chapter, we introduced a novel spiking neural network architecture called Threshold Modulated Spiking Neural Networks (TM-SNN) for addressing multi-task classification problems. TM-SNN utilizes firing threshold modulation to adapt its behavior during operation. The architecture consists of three processing blocks: feature extraction, label classification, and task classification. The task classification block serves as an additional source for regularizing the feature extraction block. Through training, the inclusion of the task classifier block resulted in a slight improvement in testing accuracy.

Our experiments were conducted using Intel's Lava neuromorphic platform, and we performed tests in both simulation and on the Loihi2 neuromorphic chip. The results demonstrate that TM-SNN is capable of learning multiple tasks with only a marginal reduction in accuracy compared to spiking neural networks trained for single task classification (ST-SNN). Specifically, TM-SNN achieved accuracy rates of 98.24%, 99.17%, and 98.84% for the three task classifications, respectively, while ST-SNN achieved 98.93%, 99.34%, and 99.01% accuracy.

Comparing the use of a firing threshold modulation to the use of external input current, we observe that modulation of the firing threshold leads to better accuracy performance. In terms of computational efficiency, TM-SNN exhibits a lower energy consumption compared to artificial neural networks. It utilizes 9.91x, 21.16x, and 47.63x fewer synaptic operations (SynOps) depending on the task and firing threshold employed. The observed variation in energy consumption based on the firing threshold modulation is particularly noteworthy as it highlights the flexibility of TM-SNN in optimizing energy usage.

While the results presented in this chapter pertain specifically to multitask classification on the NMNIST dataset, future work can expand the application of threshold modulation to more complex problems. For instance, employing firing threshold modulation for wheel angle control in autonomous driving based on images. We acknowledge that the manual modulation of the firing threshold in TM-SNN limits its flexibility. Therefore, future efforts will focus on exploring dynamic modulation of the firing threshold to enhance the adaptability and versatility of the system. These proposed improvements are part of our ongoing work.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

This chapter summarizes the overall conclusions of the dissertation and specifies some ideas as future work.

NC systems and the SNNs they use have great potential for developing low-power adaptable AI. However, challenges such as training complexity, hyperparameter selection, computational flexibility and scarcity of training data hinder their wider use.

In this dissertation, we aim to increase usage of NC by enhancing performance of SNNs. To achieve this goal, we proposed two SNNs architectures to address these limitations. The first architecture, presented in Chapter 3, utilizes auxiliary learning to improve training performance and data efficiency. The second architecture, presented in Chapter 4, leverages the neuromodulation capabilities of spiking neurons to enhance multitask performance. We validate the proposed architectures through experiments using the SpikingJelly and Lava neuromorphic libraries.

While our experiments demonstrate effectiveness of the proposed architectures, they also reveal some limitations that would be worth studying in future work. One of such limitation is the ussage of LIF neurons only. Future work could analyze use IZ neurons as well as its deployment on the Loihi2 neuromorphic computer hardware. To utilize IZ neurons, a compatible with spike-based backpropagation version of it is required. To use Loihi2, extensions to the Lava software package is needed.

Appendix A

ABBREVIATIONS

AL	Auxiliary Learning
ANNs	Artificial Neural Networks
BPTT	Backpropagation Through Time
CNN	Convolutional Neural Network
FPGAs	field-programmable gate arrays
HH	Hodgkin-Huxley
IF	Integrate and Fire
IZ	Izhikevich
LIF	Leaky Integrate and Fire
ML	multi-task Learning
MT-SNN	multi-task Spiking Neural Network
MT-SNN-EC	multi-task Spiking Neural Network with External Current
NC	Neuromorphic Computing
PLIF	Parametric Leaky Integrate and Fire
R-STDP	Reward Modulated STDP
RNN	Recurrent Neural Network
SBP	Spike-based Backpropagation
SGD	Surrogate Gradient Descent
SNNs	Spiking Neural Networks
ST-SNN	Single-task Spiking Neural Network
STDP	Spike-Timing-Dependent Plasticity

Appendix B

LIST OF PUBLICATIONS BY THE AUTHOR

This appendix presents a list of the author's published journal and peer-reviewed conference publications.

B.1 Journal Publications

1. Paolo G. Cachi, Sebastian Ventura, and Krzysztof J. Cios, "Improving Spiking Neural Network Performance with Auxiliary Learning", under review.
2. Paolo G. Cachi, Sebastian Ventura, and Krzysztof J. Cios, "Implementing Threshold Modulated Spiking Neural Networks on Loihi2 Neuromorphic Chip", under review.
3. Paolo G. Cachi, Sebastian Ventura, and Krzysztof J. Cios, "CRBA: A Rate-Based Algorithm Based on Competitive Spiking Neural Networks," in Frontiers in Computational Neuroscience, vol. 15, p. 32, 2021.

B.2 Journal Publications

1. Paolo G. Cachi, Sebastian Ventura, and Krzysztof J. Cios, "TM-SNN: Threshold Modulated Spiking Neural Network for Multi-task Learning," in 17th International Work-Conference on Artificial Neural Networks (IWANN2023), in press 2023.
2. Paolo G. Cachi, Soumil Jain, Sebastian Ventura, Gert Cauwenberghs, and Krzysztof J. Cios, "Reproducing Aplysia R-15 Bursting Neurodynamics on a

Neuromorphic Microchip," in 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 1–4, 2022.

3. Paolo G. Cachi, Sebastian Ventura, and Krzysztof J. Cios, "Improving Spiking Neural Network Performance with Auxiliary Learning", in development 2023.

4. Paolo G. Cachi, Sebastian Ventura, and Krzysztof J. Cios, "Fast Convergence of Competitive Spiking Neural Networks with Sample-Based Weight Initialization," in Information Processing and Management of Uncertainty in Knowledge-Based Systems, pp. 773–786, 2020.

REFERENCES

[1] C. Mead. "Neuromorphic electronic systems". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1629–1636. DOI: 10.1109/5.58356.

[2] Steve Furber. "Large-scale neuromorphic computing systems". In: *Journal of Neural Engineering* 13.5 (Aug. 2016), p. 051001. DOI: 10.1088/1741-2560/13/5/051001. URL: https://dx.doi.org/10.1088/1741-2560/13/5/051001.

[3] Geoffrey W. Burr et al. "Neuromorphic computing using non-volatile memory". In: *Advances in Physics: X* 2.1 (2017), pp. 89–124. DOI: 10.1080/23746149.2016.1259585. eprint: https://doi.org/10.1080/23746149.2016.1259585. URL: https://doi.org/10.1080/23746149.2016.1259585.

[4] Shufang Zhao et al. "Neuromorphic-computing-based adaptive learning using ion dynamics in flexible energy storage devices". In: *National Science Review* 9.11 (Aug. 2022). nwac158. ISSN: 2095-5138. DOI: 10.1093/nsr/nwac158. eprint: https://academic.oup.com/nsr/article-pdf/9/11/nwac158/47840181/nwac158_supplemental_file.pdf. URL: https://doi.org/10.1093/nsr/nwac158.

[5] Aboozar Taherkhani et al. "A Review of Learning in Biologically Plausible Spiking Neural Networks". In: *Neural Netw.* 122.C (Feb. 2020), pp. 253–272. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2019.09.036. URL: https://doi.org/10.1016/j.neunet.2019.09.036.

[6] Michael Pfeiffer and Thomas Pfeil. "Deep Learning With Spiking Neurons: Opportunities and Challenges". In: *Frontiers in Neuroscience* 12 (2018). ISSN:

1662-453X. DOI: 10.3389/fnins.2018.00774. URL: https://www.frontiersin.org/articles/10.3389/fnins.2018.00774.

[7] Jason K. Eshraghian et al. *Training Spiking Neural Networks Using Lessons From Deep Learning*. 2021. DOI: 10.48550/ARXIV.2109.12894. URL: https://arxiv.org/abs/2109.12894.

[8] Amirhossein Tavanaei et al. "Deep learning in spiking neural networks". In: *Neural Networks* 111 (2019), pp. 47–63. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2018.12.002. URL: https://www.sciencedirect.com/science/article/pii/S0893608018303332.

[9] C Koch, M Rapp, and I Segev. "A brief history of time (constants)". en. In: *Cereb Cortex* 6.2 (Mar. 1996), pp. 93–101.

[10] Yuhang Li et al. "Neuromorphic Data Augmentation for Training Spiking Neural Networks". In: *Computer Vision – ECCV 2022*. Ed. by Shai Avidan et al. Cham: Springer Nature Switzerland, 2022, pp. 631–649. ISBN: 978-3-031-20071-7.

[11] Andrew R. Barron. "Approximation and estimation bounds for artificial neural networks". In: *Machine Learning* 14.1 (Jan. 1994), pp. 115–133. ISSN: 1573-0565. DOI: 10.1007/BF00993164. URL: https://doi.org/10.1007/BF00993164.

[12] Shikun Liu, Andrew Davison, and Edward Johns. "Self-Supervised Generalisation with Meta Auxiliary Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/92262bf907af914b95a0fc33c3f33bf6-Paper.pdf.

[13] Trevor Standley et al. *Which Tasks Should Be Learned Together in Multi-task Learning?* 2019. DOI: 10.48550/ARXIV.1905.07553. URL: https://arxiv.org/abs/1905.07553.

[14] Wei Fang et al. *SpikingJelly*. https://github.com/fangwei123456/spikingjelly. Accessed: 2023-07-08. 2020.

[15] Hongmin Li et al. "CIFAR10-DVS: An Event-Stream Dataset for Object Classification". In: *Frontiers in Neuroscience* 11 (2017). ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00309. URL: https://www.frontiersin.org/articles/10.3389/fnins.2017.00309.

[16] Arnon Amir et al. "A Low Power, Fully Event-Based Gesture Recognition System". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 7388–7397. DOI: 10.1109/CVPR.2017.781.

[17] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. "Attentive Single-Tasking of Multiple Tasks". In: *CoRR* abs/1904.08918 (2019). arXiv: 1904.08918. URL: http://arxiv.org/abs/1904.08918.

[18] Eve Marder. "Neuromodulation of neuronal circuits: back to the future". In: *Neuron* 76.1 (Oct. 2012), pp. 1–11. DOI: 10.1016/j.neuron.2012.09.010.

[19] Sumit Bam Shrestha and Garrick Orchard. *SLAYER: Spike Layer Error Reassignment in Time*. 2018. DOI: 10.48550/ARXIV.1810.08646. URL: https://arxiv.org/abs/1810.08646.

[20] Garrick Orchard et al. "Efficient Neuromorphic Signal Processing with Loihi 2". In: *CoRR* abs/2111.03746 (2021). arXiv: 2111.03746. URL: https://arxiv.org/abs/2111.03746.

[21] Dennis V Christensen et al. “2022 roadmap on neuromorphic computing and engineering”. In: *Neuromorphic Computing and Engineering* 2.2 (May 2022), p. 022501. DOI: 10.1088/2634-4386/ac4a83. URL: https://dx.doi.org/10.1088/2634-4386/ac4a83.

[22] Catherine D. Schuman et al. “Opportunities for neuromorphic computing algorithms and applications”. In: *Nature Computational Science* 2.1 (Jan. 2022), pp. 10–19. ISSN: 2662-8457. DOI: 10.1038/s43588-021-00184-y. URL: https://doi.org/10.1038/s43588-021-00184-y.

[23] Danijela Marković et al. “Physics for neuromorphic computing”. In: *Nature Reviews Physics* 2.9 (Sept. 2020), pp. 499–510. ISSN: 2522-5820. DOI: 10.1038/s42254-020-0208-2. URL: https://doi.org/10.1038/s42254-020-0208-2.

[24] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. “Towards spike-based machine intelligence with neuromorphic computing”. In: *Nature* 575.7784 (Nov. 2019), pp. 607–617. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1677-2. URL: https://doi.org/10.1038/s41586-019-1677-2.

[25] Shih-Chii Liu et al. *Event-based neuromorphic systems*. John Wiley & Sons, 2014.

[26] Yoeri van de Burgt et al. “Organic electronics for neuromorphic computing”. In: *Nature Electronics* 1.7 (July 2018), pp. 386–397. ISSN: 2520-1131. DOI: 10.1038/s41928-018-0103-3. URL: https://doi.org/10.1038/s41928-018-0103-3.

[27] Catherine D. Schuman et al. “A Survey of Neuromorphic Computing and Neural Networks in Hardware”. In: *CoRR* abs/1705.06963 (2017). arXiv: 1705.06963. URL: http://arxiv.org/abs/1705.06963.

[28] Aaron R. Voelker and Chris Eliasmith. "Improving Spiking Dynamical Networks: Accurate Delays, Higher-Order Synapses, and Time Cells". In: *Neural Computation* 30.3 (2018), pp. 569–609. DOI: 10.1162/neco_a_01046.

[29] Mike Davies et al. "Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook". In: *Proceedings of the IEEE* 109.5 (2021), pp. 911–934. DOI: 10.1109/JPROC.2021.3067593.

[30] Mike Davies et al. "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning". In: *IEEE Micro* 38.1 (2018), pp. 82–99. DOI: 10.1109/MM.2018.112130359.

[31] Michael V. DeBole et al. "TrueNorth: Accelerating From Zero to 64 Million Neurons in 10 Years". In: *Computer* 52.5 (2019), pp. 20–29. DOI: 10.1109/MC.2019.2903009.

[32] Filipp Akopyan et al. "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (2015), pp. 1537–1557. DOI: 10.1109/TCAD.2015.2474396.

[33] SAMANWOY GHOSH-DASTIDAR and HOJJAT ADELI. "SPIKING NEURAL NETWORKS". In: *International Journal of Neural Systems* 19.04 (2009). PMID: 19731402, pp. 295–308. DOI: 10.1142/S0129065709002002. eprint: https://doi.org/10.1142/S0129065709002002. URL: https://doi.org/10.1142/S0129065709002002.

[34] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511815706.

[35] Wolfgang Maass. "Networks of spiking neurons: The third generation of neural network models". In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 0893-6080. DOI: `https://doi.org/10.1016/S0893-6080(97)00011-7`. URL: `https://www.sciencedirect.com/science/article/pii/S0893608097000117`.

[36] Laith Alzubaidi et al. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". In: *Journal of Big Data* 8.1 (Mar. 2021), p. 53. ISSN: 2196-1115. DOI: `10.1186/s40537-021-00444-8`. URL: `https://doi.org/10.1186/s40537-021-00444-8`.

[37] Frank Emmert-Streib et al. "An Introductory Review of Deep Learning for Prediction Models With Big Data". In: *Frontiers in Artificial Intelligence* 3 (2020). ISSN: 2624-8212. DOI: `10.3389/frai.2020.00004`. URL: `https://www.frontiersin.org/articles/10.3389/frai.2020.00004`.

[38] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: `10.1038/nature14539`. URL: `https://doi.org/10.1038/nature14539`.

[39] Guillermo Gallego et al. "Event-Based Vision: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.1 (Jan. 2022), pp. 154–180. DOI: `10.1109/tpami.2020.3008413`. URL: `https://doi.org/10.1109%5C%2Ftpami.2020.3008413`.

[40] Boudjelal Meftah et al. "Image Processing with Spiking Neuron Networks". In: *Artificial Intelligence, Evolutionary Computing and Metaheuristics: In the Footsteps of Alan Turing*. Ed. by Xin-She Yang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 525–544. ISBN: 978-3-642-29694-9. DOI: `10.1007/978-3-642-29694-9_20`. URL: `https://doi.org/10.1007/978-3-642-29694-9_20`.

[41] J. Shin et al. "Recognition of Partially Occluded and Rotated Images With a Network of Spiking Neurons". In: *IEEE Transactions on Neural Networks* 21.11 (Nov. 2010), pp. 1697–1709. ISSN: 1941-0093. DOI: 10.1109/TNN.2010.2050600.

[42] Krzysztof J. Cios and Inho Shin. "Image recognition neural network: IRNN". In: *Neurocomputing* 7.2 (1995), pp. 159–185. ISSN: 0925-2312. DOI: https://doi.org/10.1016/0925-2312(93)E0062-I.

[43] Kaushalya Kumarasinghe, Nikola Kasabov, and Denise Taylor. "Brain-inspired spiking neural networks for decoding and understanding muscle activity and kinematics from electroencephalography signals during hand movements". In: *Scientific Reports* 11.1 (Jan. 2021), p. 2486. ISSN: 2045-2322. DOI: 10.1038/s41598-021-81805-4. URL: https://doi.org/10.1038/s41598-021-81805-4.

[44] Soufiyan Bahadi, Jean Rouat, and Éric Plourde. "Adaptive Approach for Sparse Representations Using the Locally Competitive Algorithm for Audio". In: *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*. 2021, pp. 1–6. DOI: 10.1109/MLSP52302.2021.9596348.

[45] Juan P. Dominguez-Morales et al. "Deep Spiking Neural Network model for time-variant signals classification: a real-time speech recognition approach". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–8. DOI: 10.1109/IJCNN.2018.8489381.

[46] Jens Kremkow, Ad Aertsen, and Arvind Kumar. "Gating of Signal Propagation in Spiking Neural Networks by Balanced and Correlated Excitation and Inhibition". In: *Journal of Neuroscience* 30.47 (2010), pp. 15760–15768. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.3874-10.2010. eprint: https:

//www.jneurosci.org/content/30/47/15760.full.pdf. URL: https://www.jneurosci.org/content/30/47/15760.

[47] R.F. Lyon and C. Mead. “An analog electronic cochlea”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 36.7 (1988), pp. 1119–1134. DOI: 10.1109/29.1639.

[48] Michael Ehrlich et al. “Adaptive control of a wheelchair mounted robotic arm with neuromorphically integrated velocity readings and online-learning”. In: *Frontiers in Neuroscience* 16 (2022). ISSN: 1662-453X. DOI: 10.3389/fnins.2022.1007736. URL: https://www.frontiersin.org/articles/10.3389/fnins.2022.1007736.

[49] Marco Monforte et al. “Where and When: Event-Based Spatiotemporal Trajectory Prediction from the iCub’s Point-Of-View”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 9521–9527. DOI: 10.1109/ICRA40945.2020.9197373.

[50] Daniel Gutierrez-Galan et al. “Neuropod: A real-time neuromorphic spiking CPG applied to robotics”. In: *Neurocomputing* 381 (Mar. 2020), pp. 10–19. DOI: 10.1016/j.neucom.2019.11.007. URL: https://doi.org/10.1016%5C%2Fj.neucom.2019.11.007.

[51] Mohammadreza Mohammadi et al. “Static hand gesture recognition for American sign language using neuromorphic hardware”. In: *Neuromorphic Computing and Engineering* 2.4 (Oct. 2022), p. 044005. DOI: 10.1088/2634-4386/ac94f3. URL: https://dx.doi.org/10.1088/2634-4386/ac94f3.

[52] Elvin Hajizada et al. “Interactive Continual Learning for Robots: A Neuromorphic Approach”. In: *Proceedings of the International Conference on Neuromorphic Systems 2022*. ICONS ’22. Knoxville, TN, USA: Association for

Computing Machinery, 2022. ISBN: 9781450397896. DOI: 10.1145/3546790.3546791. URL: https://doi.org/10.1145/3546790.3546791.

[53] Kyle Buettner and Alan D. George. "Heartbeat Classification with Spiking Neural Networks on the Loihi Neuromorphic Processor". In: *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2021, pp. 138–143. DOI: 10.1109/ISVLSI51109.2021.00035.

[54] Enea Ceolini et al. "Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing". In: *Frontiers in Neuroscience* 14 (2020). ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00637. URL: https://www.frontiersin.org/articles/10.3389/fnins.2020.00637.

[55] J. Darby Smith et al. "Solving a Steady-State PDE Using Spiking Networks and Neuromorphic Hardware". In: *International Conference on Neuromorphic Systems 2020*. ICONS 2020. Oak Ridge, TN, USA: Association for Computing Machinery, 2020. ISBN: 9781450388511. DOI: 10.1145/3407197.3407202. URL: https://doi.org/10.1145/3407197.3407202.

[56] Maxence Bouvier et al. "Spiking Neural Networks Hardware Implementations and Challenges: A Survey". In: *J. Emerg. Technol. Comput. Syst.* 15.2 (Apr. 2019). ISSN: 1550-4832. DOI: 10.1145/3304103. URL: https://doi.org/10.1145/3304103.

[57] Yujie Wu et al. "Direct Training for Spiking Neural Networks: Faster, Larger, Better". In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI

Press, 2019. ISBN: 978-1-57735-809-1. DOI: 10.1609/aaai.v33i01.33011311. URL: https://doi.org/10.1609/aaai.v33i01.33011311.

[58] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. "Training Deep Spiking Neural Networks Using Backpropagation". In: *Frontiers in Neuroscience* 10 (2016). ISSN: 1662-453X. DOI: 10.3389/fnins.2016.00508. URL: https://www.frontiersin.org/articles/10.3389/fnins.2016.00508.

[59] Wulfram Gerstner and Werner M. Kistler. "Mathematical formulations of Hebbian learning". In: *Biological Cybernetics* 87.5 (2002), pp. 404–415. ISSN: 1432-0770. DOI: 10.1007/s00422-002-0353-y. URL: https://doi.org/10.1007/s00422-002-0353-y.

[60] Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[61] Jerzy Konorski. *Conditioned reflexes and neuron organization*. Cambridge University Press, 1948.

[62] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. "Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)". In: *Frontiers in Neuroscience* 14 (2020). ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00424. URL: https://www.frontiersin.org/articles/10.3389/fnins.2020.00424.

[63] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks". In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63. DOI: 10.1109/MSP.2019.2931595.

[64] Maurizio Mattia and Paolo Del Giudice. "Population dynamics of interacting spiking neurons". en. In: *Phys Rev E Stat Nonlin Soft Matter Phys* 66.5 Pt 1 (Nov. 2002), p. 051917.

[65] Petro Liashchynskyi and Pavlo Liashchynskyi. *Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS*. 2019. DOI: 10.48550/ARXIV.1912.06059. URL: https://arxiv.org/abs/1912.06059.

[66] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. 2018. DOI: 10.48550/ARXIV.1807.02811. URL: https://arxiv.org/abs/1807.02811.

[67] Bojian Yin, Federico Corradi, and Sander M. Bohté. *Effective and Efficient Computation with Multiple-timescale Spiking Recurrent Neural Networks*. 2020. DOI: 10.48550/ARXIV.2005.11633. URL: https://arxiv.org/abs/2005.11633.

[68] Garrick Orchard et al. "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades". In: *Frontiers in Neuroscience* 9 (2015). ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00437.

[69] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[70] Eilen Nordlie, Tom Tetzlaff, and Gaute Einevoll. "Rate Dynamics of Leaky Integrate-and-Fire Neurons with Strong Synapses". In: *Frontiers in Computational Neuroscience* 4 (2010). ISSN: 1662-5188. DOI: 10.3389/fncom.2010.00149. URL: https://www.frontiersin.org/articles/10.3389/fncom.2010.00149.

[71] Gabrielle J. Gutierrez, Timothy O'Leary, and Eve Marder. "Multiple Mechanisms Switch an Electrically Coupled, Synaptically Inhibited Neuron between Competing Rhythmic Oscillators". In: *Neuron* 77.5 (2013), pp. 845–858. ISSN: 0896-6273.

[72] Yunshu Du et al. *Adapting Auxiliary Losses Using Gradient Similarity.* 2018. DOI: 10.48550/ARXIV.1812.02224. URL: https://arxiv.org/abs/1812.02224.

[73] Wei Fang et al. *Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks.* 2020. DOI: 10.48550/ARXIV.2007.05785. URL: https://arxiv.org/abs/2007.05785.

[74] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[75] F Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain". en. In: *Psychol Rev* 65.6 (Nov. 1958), pp. 386–408.

[76] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[77] David E. Rumelhart and James L. McClelland. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations.* 1987, pp. 318–362.

[78] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[79] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. *Recurrent Neural Network Regularization*. 2014. DOI: 10.48550/ARXIV.1409.2329. URL: https://arxiv.org/abs/1409.2329.

[80] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[81] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

[82] Léon Bottou. “Online Algorithms and Stochastic Approximations”. In: *Online Learning and Neural Networks*. Ed. by David Saad. revised, oct 2012. Cambridge, UK: Cambridge University Press, 1998. URL: http://leon.bottou.org/papers/bottou-98x.

[83] Wulfram Gerstner et al. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. New York, NY, USA: Cambridge University Press, 2014.

[84] Eric R Kandel et al. *Principles of neural science*. Vol. 5. McGraw-hill New York, 2013.

[85] Christof Koch and Idan Segev. *Methods in neuronal modeling: from ions to networks*. MIT Press, 1998.

[86] E.M. Izhikevich. “Simple model of spiking neurons”. In: *IEEE Transactions on Neural Networks* 14.6 (2003), pp. 1569–1572. DOI: 10.1109/TNN.2003.820440.

[87] A L Hodgkin and A F Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *The Journal of physiology* 117.4 (Aug. 1952), pp. 500–544.

[88] Beata Strack, Kimberle M Jacobs, and Krzysztof J Cios. “Simulating vertical and horizontal inhibition with short-term dynamics in a multi-column multi-layer model of neocortex”. en. In: *Int. J. Neural Syst.* 24.5 (Aug. 2014), p. 1440002.

[89] Beata Strack, Kimberle M. Jacobs, and Krzysztof J. Cios. “Biological restraint on the Izhikevich neuron model essential for seizure modeling”. In: *2013 6th International IEEE/EMBS Conference on Neural Engineering (NER)*. 2013, pp. 395–398. DOI: 10.1109/NER.2013.6695955.

[90] Nicolas Frémaux, Henning Sprekeler, and Wulfram Gerstner. “Functional requirements for reward-modulated spike-timing-dependent plasticity”. en. In: *J Neurosci* 30.40 (Oct. 2010), pp. 13326–13337.

[91] Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. “A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback”. In: *PLOS Computational Biology* 4.10 (Oct. 2008), pp. 1–27. DOI: 10.1371/journal.pcbi.1000180. URL: https://doi.org/10.1371/journal.pcbi.1000180.

[92] Michael C. Mozer. “A Focused Backpropagation Algorithm for Temporal Pattern Recognition”. In: *Complex Syst.* 3 (1989).

[93] Amar Shrestha et al. “A Survey on Neuromorphic Computing: Models and Hardware”. In: *IEEE Circuits and Systems Magazine* 22.2 (2022), pp. 6–35. DOI: 10.1109/MCAS.2022.3166331.

[94] Paul A. Merolla et al. “A million spiking-neuron integrated circuit with a scalable communication network and interface”. In: *Science* 345.6197 (2014), pp. 668–673. DOI: 10.1126/science.1254642. eprint: https://www.science.org/doi/pdf/10.1126/science.1254642. URL: https://www.science.org/doi/abs/10.1126/science.1254642.

[95] Andrew S. Cassidy et al. “Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores”. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. 2013, pp. 1–10. DOI: 10.1109/IJCNN.2013.6707077.

[96] Christian Mayr, Sebastian Höppner, and Steve B. Furber. “SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning”. In: *CoRR* abs/1911.02385 (2019). arXiv: 1911.02385. URL: http://arxiv.org/abs/1911.02385.

[97] Steve B. Furber et al. “The SpiNNaker Project”. In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665. DOI: 10.1109/JPROC.2014.2304638.

[98] Eustace Painkras et al. “SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation”. In: *IEEE Journal of Solid-State Circuits* 48.8 (2013), pp. 1943–1953. DOI: 10.1109/JSSC.2013.2259038.

[99] Haowen Fang et al. "Scalable NoC-based Neuromorphic Hardware Learning and Inference". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–8. DOI: 10.1109/IJCNN.2018.8489619.

[100] Moslem Heidarpur et al. "CORDIC-SNN: On-FPGA STDP Learning With Izhikevich Neurons". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 66.7 (2019), pp. 2651–2661. DOI: 10.1109/TCSI.2019.2899356.

[101] Amirreza Yousefzadeh et al. "On Practical Issues for Stochastic STDP Hardware With 1-bit Synaptic Weights". In: *Frontiers in Neuroscience* 12 (2018). ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00665. URL: https://www.frontiersin.org/articles/10.3389/fnins.2018.00665.

[102] Dion Khodagholy et al. "NeuroGrid: recording action potentials from the surface of the brain". In: *Nature Neuroscience* 18.2 (Feb. 2015), pp. 310–315. ISSN: 1546-1726. DOI: 10.1038/nn.3905. URL: https://doi.org/10.1038/nn.3905.

[103] Ben Varkey Benjamin et al. "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations". In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716. DOI: 10.1109/JPROC.2014.2313565.

[104] Konstantinos I. Papadimitriou et al. "Neuromorphic log-domain silicon synapse circuits obey bernoulli dynamics: a unifying tutorial analysis". In: *Frontiers in Neuroscience* 8 (2015). ISSN: 1662-453X. DOI: 10.3389/fnins.2014.00428. URL: https://www.frontiersin.org/articles/10.3389/fnins.2014.00428.

[105] Massimiliano Giulioni et al. "Robust Working Memory in an Asynchronously Spiking Neural Network Realized with Neuromorphic VLSI". In: *Frontiers in Neuroscience* 5 (2012). ISSN: 1662-453X. DOI: 10.3389/fnins.2011.00149.

URL: https://www.frontiersin.org/articles/10.3389/fnins.2011.00149.

[106] Theodore Yu and Gert Cauwenberghs. "Analog VLSI Biophysical Neurons and Synapses With Programmable Membrane Channel Kinetics". In: *IEEE Transactions on Biomedical Circuits and Systems* 4.3 (2010), pp. 139–148. DOI: 10.1109/TBCAS.2010.2048566.

[107] E. Farquhar, C. Gordon, and P. Hasler. "A field programmable neural array". In: *2006 IEEE International Symposium on Circuits and Systems*. 2006, 4 pp.–4117. DOI: 10.1109/ISCAS.2006.1693534.

[108] Ming Liu, Hua Yu, and Wei Wang. "FPAA Based on Integration of CMOS and Nanojunction Devices for Neuromorphic Applications". In: *Nano-Net*. Ed. by Maggie Cheng. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 44–48. ISBN: 978-3-642-02427-6.

[109] Christian Pehle et al. "The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity". In: *Frontiers in Neuroscience* 16 (2022). ISSN: 1662-453X. DOI: 10.3389/fnins.2022.795876. URL: https://www.frontiersin.org/articles/10.3389/fnins.2022.795876.

[110] Eric Muller et al. "The Operating System of the Neuromorphic BrainScaleS-1 System". In: *CoRR* abs/2003.13749 (2020). arXiv: 2003.13749. URL: https://arxiv.org/abs/2003.13749.

[111] Federico Corradi et al. "Decision making and perceptual bistability in spike-based neuromorphic VLSI systems". In: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2015, pp. 2708–2711. DOI: 10.1109/ISCAS.2015.7169245.

[112] Mihai A Petrovici et al. “Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms”. en. In: *PLoS One* 9.10 (Oct. 2014), e108590.

[113] Srinjoy Mitra, Giacomo Indiveri, and Stefano Fusi. “Learning to classify complex patterns using a VLSI network of spiking neurons”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Platt et al. Vol. 20. Curran Associates, Inc., 2007. URL: `https://proceedings.neurips.cc/paper_files/paper/2007/file/c3992e9a68c5ae12bd18488bc579b30d-Paper.pdf`.

[114] *Taking Neuromorphic Computing to the Next Level with Loihi 2*. `https://download.intel.com/newsroom/2021/new-technologies/neuromorphic-computing-loihi-2-brief.pdf`. Accessed: 07-08-2023.

[115] Intel Labs. *Lava Software*. `https://github.com/lava-nc/lava-dl`. Accessed: 2023-07-08. 2023.

[116] Marc-Oliver Gewaltig and Markus Diesmann. “NEST (NEural Simulation Tool)”. In: *Scholarpedia* 2.4 (2007), p. 1430.

[117] Marcel Stimberg, Romain Brette, and Dan FM Goodman. “Brian 2, an intuitive and efficient neural simulator”. In: *eLife* 8 (Aug. 2019). Ed. by Frances K Skinner, e47314. ISSN: 2050-084X. DOI: `10.7554/eLife.47314`.

[118] Alexander Kugele et al. “Efficient Processing of Spatio-Temporal Data Streams With Spiking Neural Networks”. In: *Frontiers in Neuroscience* 14 (2020). ISSN: 1662-453X. DOI: `10.3389/fnins.2020.00439`. URL: `https://www.frontiersin.org/articles/10.3389/fnins.2020.00439`.

[119] Nour Eldeen Khalifa, Mohamed Loey, and Seyedali Mirjalili. “A comprehensive survey of recent trends in deep learning for digital images augmentation”.

In: *Artificial Intelligence Review* 55.3 (Mar. 2022), pp. 2351–2377. ISSN: 1573-7462. DOI: 10.1007/s10462-021-10066-4. URL: https://doi.org/10.1007/s10462-021-10066-4.

[120] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (July 2019), p. 60. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0.

[121] Baifeng Shi et al. *Auxiliary Task Reweighting for Minimum-data Learning.* 2020. DOI: 10.48550/ARXIV.2010.08244. URL: https://arxiv.org/abs/2010.08244.

[122] Fynn Schröder and Chris Biemann. "Estimating the influence of auxiliary tasks for multi-task learning of sequence tagging tasks". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.* Online: Association for Computational Linguistics, July 2020, pp. 2971–2985. DOI: 10.18653/v1/2020.acl-main.268. URL: https://aclanthology.org/2020.acl-main.268.

[123] Michael Crawshaw. "Multi-Task Learning with Deep Neural Networks: A Survey". In: *CoRR* abs/2009.09796 (2020). arXiv: 2009.09796. URL: https://arxiv.org/abs/2009.09796.

[124] Sebastian Ruder. *An Overview of Multi-Task Learning in Deep Neural Networks.* 2017. DOI: 10.48550/ARXIV.1706.05098. URL: https://arxiv.org/abs/1706.05098.

[125] Zirui Wang et al. *Characterizing and Avoiding Negative Transfer.* 2018. DOI: 10.48550/ARXIV.1811.09751. URL: https://arxiv.org/abs/1811.09751.

[126] Aviv Navon et al. "Auxiliary Learning by Implicit Differentiation". In: *CoRR* abs/2007.02693 (2020). arXiv: 2007.02693. URL: https://arxiv.org/abs/2007.02693.

[127] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556.

[128] Hanle Zheng et al. *Going Deeper With Directly-Trained Larger Spiking Neural Networks*. 2020. arXiv: 2011.05280 [cs.NE].

[129] Yuhang Li et al. "Differentiable Spike: Rethinking Gradient-Descent for Training Spiking Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 23426–23439.

[130] Byunggook Na et al. "AutoSNN: Towards Energy-Efficient Spiking Neural Networks". In: *CoRR* abs/2201.12738 (2022). arXiv: 2201.12738. URL: https://arxiv.org/abs/2201.12738.

[131] Yufei Guo et al. "RecDis-SNN: Rectifying Membrane Potential Distribution for Directly Training Spiking Neural Networks". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 326–335. DOI: 10.1109/CVPR52688.2022.00042.

[132] Qingyan Meng et al. *Training High-Performance Low-Latency Spiking Neural Networks by Differentiation on Spike Representation*. 2023. arXiv: 2205.00459 [cs.NE].

[133] Zhaokun Zhou et al. "Spikformer: When Spiking Neural Network Meets Transformer". In: *The Eleventh International Conference on Learning Representations.* 2023.

[134] Haibo Shen et al. "Training Stronger Spiking Neural Networks with Biomimetic Adaptive Internal Association Neurons". In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2023, pp. 1–5. DOI: `10.1109/ICASSP49357.2023.10096958`.

[135] Michael T Rosenstein et al. "To transfer or not to transfer". In: *NIPS 2005 workshop on transfer learning.* Vol. 898. 2005.

[136] Hakan Bilen and Andrea Vedaldi. "Universal representations: The missing link between faces, text, planktons, and cat breeds". In: *CoRR* abs/1701.07275 (2017). arXiv: `1701.07275`. URL: `http://arxiv.org/abs/1701.07275`.

[137] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. "Adversarial Multi-task Learning for Text Classification". In: *CoRR* abs/1704.05742 (2017). arXiv: `1704.05742`. URL: `http://arxiv.org/abs/1704.05742`.

[138] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. "Efficient parametrization of multi-domain deep neural networks". In: *CoRR* abs/1803.10082 (2018). arXiv: `1803.10082`. URL: `http://arxiv.org/abs/1803.10082`.

[139] Yaroslav Ganin and Victor Lempitsky. "Unsupervised Domain Adaptation by Backpropagation". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37.* ICML'15. Lille, France: JMLR.org, 2015, pp. 1180–1189.

VITA

Paolo G. Cachi received his Bachelor's degree in Electronic Engineering from the San Antonio Abad del Cusco University in Peru in 2012 and his Master's degree in Electrical Engineering from the Pontifícia Universidade Católica do Rio de Janeiro in Brazil in 2015. He began the Doctor of Philosophy program at Virginia Commonwealth University in Richmond, Virginia in 2018 and is currently working as a research assistant under the supervision of Professor Krzysztof Cios. His research interests include neuromorphic computing, spiking neural networks, machine learning, and artificial general intelligence.

REASONED SUPERVISOR'S REPORT

This document is to be presented together with the filing of the thesis at https://moodle.uco.es/ctp3/

PhD STUDENT

Paolo Gabriel Alejandro Cachi Delgado

THESIS TITLE:

Enhancing Neuromorphic Computing with Advanced Spiking Neural Network Architectures

REASONED SUPERVISOR'S REPORT
(Ratifying the advisor's favorable report. Only when the advisor does not belong to the University of Cordoba)

Mr. Paolo Cachi's research has been in the area of spiking neural networks. In particular he used auxiliary learning for their improving performance. The network architecture he proposed consists of a feature extraction block connected in a feed-forward fashion to a main classification block and one or more auxiliary classification blocks. By using auxiliary tasks, additional information during training is used to help regularize the feature extraction block. As a result, the feature extraction block learns more general and robust features which improves performance on the main task. The experiments were performed using the SpikingJelly neuromorphic library as well as using Intel's Lava framework. The other area of his research has been in multi-task (MT) spiking neural networks (SNN) learning using modification of the firing threshold to modify its operation. The MT-SNN architecture consists of three processing blocks for feature extraction, label classification and task classification. Results of extensive experiments using Intel's Lava neuromorphic simulation platform show that MTSNN predicts both tasks with only slightly lower accuracy than single task SNN. Comparing using modification of the firing threshold of neurons with changing the external input current to neurons showed that the networks with the former achieved better accuracy than using the latter. The network was tested on Loihi2 neuromorphic computer on several neuromorphic event-based datasets.

Hence, the presentation of the doctoral thesis is authorized.

Cordoba, on the 25 de junio de 2023

The supervisor(s)

[illegible]

Signed: K J C
Krzysztof Cios

Firmado digitalmente por VENTURA SOTO SEBASTIAN EMILIO - 30510000V
Fecha: 2023.07.03 19:14:36 +02'00'

Signed:______________
Sebastian Ventura Soto