UNIVERSITY OF CALIFORNIA,
IRVINE

EMULATING BRAIN-LIKE RAPID ONLINE LEARNING WITH NEUROMORPHIC EDGE COMPUTING

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Kenneth Michael Stewart

Dissertation Committee:
Distinguished Professor Nikil Dutt, Chair
Professor Emre Neftci
Professor Jeffrey Krichmar
Assistant Professor Mohsen Imani

2023

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank Nikil Dutt, Jeffrey Krichmar, Emre Neftci, Andreea Danielescu, Eric Gallo, Timothy Shea, Noah Pacick-Nelson, Sumit Bam Shrestha, Garrick Orchard, Danielle Rager, Mike Davies, Massimiliano Iacono, Jason Eshraigan, Jason Yik, my family, and all of the friends I have made during my PhD. I also thank the University of California Irvine, Intel Labs, Accenture Labs, the Instituto di Technologia Italy, Forschungszentrum Jülich, and the NSF. Without the institutions and people mentioned this thesis and my PhD would not be possible. I feel extraordinarily lucky to have developed such a wonderful network and group of friends.

The research presented in this thesis was funded by the following grants and institutions. The Telluride Neuromorphic Cognition Engineering workshop, years 2018, 2019, and 2020 (NSF OISE 2020624), The Intel Corporation, the National Science Foundation under grant 1652159, partially supported by Programmatic grant no. A1687b0033 from the Singapore government's Research, Innovation and Enterprise 2020 plan (Advanced Manufacturing and Engineering domain), Accenture labs, the National Science Foundation (NSF) under grant 1652159, NeuroSys as part of the initiative "Clusters4Future" is funded by the Federal Ministry of Education and Research BMBF (03ZU1106XX), the National Science Foundation under grant 1640081, and the Nanoelectronics Research Corporation (NERC), a wholly-owned subsidiary of the Semiconductor Research Corporation (SRC), through Extremely Energy Efficient Collective Electronics (EXCEL), an SRC-NRI Nanoelectronics Research Initiative under Research Task ID 2698.003.

You also need to acknowledge any publishers of your previous work who have given you permission to incorporate that work into your dissertation. See Section 3.2 of the UCI Thesis and Dissertation Manual.

Chapter 3 is a modified version of the papers: [161] and [162] published in and the IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS) © 2020 IEEE and the IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS) © 2020 IEEE respectively. Chapter 4 is a modified version of the paper: [165] published in the Institute of Physics Journal of Neuromorphic Computing and Engineering (NCE) © 2022 Kenneth Michael Stewart and Emre Ozgur Neftci. Chapter 5 is a modified version of the paper: [159] published in the ACM Conference on Neural Information Computing Elements (NICE) © 2023 ACM, Inc..

Dr. Emre Neftci, Dr. Nikil Dutt, Dr. Jeffrey Krichmar, and Dr. Mohsen Imani contributed to the text and or work of chapters 1, 2, 3, 4, 5, and 6. Dr. Sumit Shrestha, and Dr. Garrick Orchard contributed to the text and work of chapters 3, 4, and 5. Dr. Andreea Danielescu, Dr. Timothy Shea, Dr. Eric Gallo, and Noah Pacick-Nelson contributed to the text and work of chapter 5.

The use of "we" in the thesis refers to the above mentioned people in the relevant sections.

VITA

Kenneth Michael Stewart

EDUCATION

Doctor of Philosophy in Computer Science **2023**
University of California, Irvine *Irvine, California*

Master of Science in Computer Science **2020**
University of California, Irvine *Irvine, California*

Bachelor of Science in Computer Science and Engineering **2017**
Michigan State University *East Lansing, Michigan*

RESEARCH EXPERIENCE

Associate Scientist **2022–2023**
Peter Grünberg Institute 15, Forschungszentrum Jülich, *Aachen, Germany*

Technology Associate Research Principle **2022–2022**
Accenture Labs, *San Francisco, California*

Research Contractor **2020–2020**
Accenture Labs, *San Francisco, California*

Graduate Research Assistant **2019–2022**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **2018–2019**
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

Meta-learning spiking neural networks with surrogate gradient descent **2022**
Institute of Physics Journal of Neuromorphic Computing and Engineering (NCE)

Online Few-Shot Gesture Learning on a Neuromorphic Processor **2020**
IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)

REFEREED CONFERENCE PUBLICATIONS

Speech2Spikes: Efficient Audio Encoding Pipeline for Real-time Neuromorphic Systems **April 2023**
2023 10th ACM Conference on Neural Information Computing Elements (NICE)

Encoding Event-Based Data With a Hybrid SNN Guided Variational Auto-encoder in Neuromorphic Hardware **April 2022**
2022 9th ACM Conference on Neural Information Computing Elements (NICE)

HyperSpike: hyperdimensional computing for more efficient and robust spiking neural networks **March 2022**
2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)

On-chip Few-shot Learning with Surrogate Gradient Descent on a Neuromorphic Processor **March 2020**
2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)

Live Demonstration: On-chip Few-shot Learning with Surrogate Gradient Descent on a Neuromorphic Processor **March 2020**
2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)

SOFTWARE

SNN MAML `https://github.com/nmi-lab/snn_maml/tree/main/snn_maml`
Pytorch SNN MAML algorithm.

Torchneuromorphic `https://github.com/nmi-lab/torchneuromorphic/tree/master`
Efficiently download and load event-based datasets.

Hybrid Guided VAE `https://github.com/kennetms/Accenture_Hybrid_Guided_VAE`
Pytorch SNN encoder ANN decoder for semi-supervised learning.

Online Gesture Learning with Loihi `https://gitlab.com/kennetms/error-triggered-gesture-`
SOEL learning rule live demo with Loihi 1 and DAVIS

ABSTRACT OF EMULATING BRAIN-LIKE RAPID ONLINE LEARNING WITH NEUROMORPHIC EDGE COMPUTING

By

Kenneth Michael Stewart

Doctor of Philosophy in Computer Science

University of California, Irvine, 2023

Distinguished Professor Nikil Dutt, Chair

The increasing demand for real-time, low-power, and intelligent systems has led to the development of edge computing, which involves processing data at the source directly from sensors rather than in the cloud or a centralized data center. Neuromorphic hardware, inspired by the structure and function of the biological nervous system, offers unique advantages for edge computing such as parallel processing, event-driven computation, and low power consumption with in memory computing. In memory computing enables the ability to perform online learning where learning can happen simultaneously with data input in real-time rather than the typical offline learning approach of presenting data in large batches for processing and then learning with gradient descent. With online learning, rapid adaptation to the unfamiliar becomes possible which is necessary for many applications such as user customization, and mobile robot navigation in unknown environments. Here we will demonstrate the machine learning algorithms we developed for use with neuromorphic hardware for online learning and rapid adaptation to unfamiliar data streamed from event-based sensors.

The thesis begins by providing an overview of the state-of-the-art in edge computing and neuromorphic hardware, highlighting their strengths and limitations for machine learning. Next, we present our original contributions, including the development of new algorithms for learning on edge neuromorphic hardware, the implementation of these algorithms on

state-of-the-art hardware platforms, and the evaluation of their performance in comparison to traditional machine learning algorithms. Our results demonstrate that edge neuromorphic hardware is a promising platform for machine learning, offering high performance, low latency, and low power consumption. Our algorithms are shown to be effective in real-world applications such as image classification, and mid-air gesture recognition. In conclusion, this thesis makes a significant contribution to the field of edge computing and neuromorphic hardware by demonstrating the feasibility of using neuromorphic hardware for learning at the edge. Our work lays the foundation for the development of efficient, low-power, and real-time intelligent systems for edge applications.

Chapter 1

Introduction

What is intelligence? As defined by the Google Oxford Languages dictionary [157], intelligence is the ability to acquire and apply knowledge and skills. The human species' most powerful feature is intelligence, enabling us to do amazing things such as communicate, form societies, engineer the world around us, and engage in reasoning and scientific discovery. This has led to interest in trying to discover how and why we are intelligent and able to do the things we do. Over the last century a multi-disciplinary effort to unlock the mysteries of intelligence has been unfolding. Scientists across fields such as neuroscience, cognitive science, computer science, philosophy, and others have been studying the brains and behaviors of animals to be able to apply the knowledge to areas such as artificial intelligence and machine learning.

Where does intelligence come from? First we can look to the building blocks of life, genetics. Genetics are the code, or programming language, that makes up all life in the known universe from single cell organisms to the multi-trillion cell human [76]. Genetic material consists of nucleic acids that consist of code in the form of RNA and DNA. Such code provides instructions for how life functions, and it is possible to manipulate that code with tools such

as CRISPR CAS9 [143]. Genetic material and what it creates is not immortal, and thus genetics have mechanisms by which to survive and reproduce to continue the existence of the genetic code and life created with the instructions [29]. However the universe is a harsh place that is difficult to survive and reproduce in to sustain and create life as we know it. Varying and changing environments and resources, limited resources, competition, these are some of the many obstacles life faces in the endless cycle of survival and reproduction. To overcome the obstacles genetic code needs to be altered in attempts to increase the chance the life it creates can survive and reproduce. Such alterations result in what we know as evolution, adaptation, or learning to optimize the chances of survival.

The universe is an ever changing place and so mechanisms evolved to give the ability to learn during the lifetime quickly in a flexible manner leading to the development of the brain [71]. The brain acts as the executive controller of an organism and is an information processing system. As an information processing system, the brain takes input from internal or external stimuli and executes functions in response. Additionally the brain is plastic meaning that its biology can change to better perform necessary functions, i.e., the brain learns to optimize functionality in response to its environment. Therefore genetic code programs the brain to effectively program itself to optimize the chances of an organism's survival in response to dynamic environments. Computers are also information processing systems, and computers have long taken inspiration from the brain, with the von Neumann architecture which took inspiration from the brain being the basis for computers used today [181]. Ada Lovelace was an English mathematician and writer who worked with Charles Babbage on the design of the first general purpose computer, the Analytical Engine. In trying to explain how the Analytical Engine worked, Ada Lovelace invented the first programming language to write a program to calculate a sequence of Bernoulli numbers, becoming the worlds first programmer. While only theoretically realized, the Analytical Engine was designed to function similarly to the von Neumann computer architecture realized a century later [183]. Building upon the work of Lovelace, Babbage, and others, the breakthroughs of von Neumann [182] and

Alan Turing [174] led to computers that could be programmed with what people today call Artificial Intelligence (AI), with the goal of AI being to program computers that can optimally program themselves in response to their environments.

Alan Turing is considered the father of the fields of Artificial Intelligence and Computer Science. Turing was one of the early adopters of the idea that general purpose computers could be created after the functions of the brain, something he discusses in his famous paper regarding what is now known as the Turing test [175]. Turing discusses how computers could eventually be made to compute in the likeness of the brain with neurons and synapses [175]. Despite being a prolific scholar, practitioner, and hero of World War II, Turing met a tragic fate at the mercy of the politics of his time but his accomplishments and legacy lives on. Other great minds of the time took upon themselves the challenge of mimicking the intelligence of the brain through modeling it and started creating mathematical models of synapse and neuron functions, chief among them the seminal work by McCulloch and Pitts. The work of Turing, McCulloch and Pitts was built upon by Geoffery Hinton [59] [58] and Yann LeCun [92] creating the study and application of Deep Learning that has defined modern Artificial Intelligence.

Learning is a fundamental process in artificial intelligence that enables machines to improve their performance over time in tasks such as pattern recognition, decision making, and problem solving. The first work on developing learning algorithms for computers based on biology is the work of McCulloch and Pitts in their highly influential paper on developing a logical calculus for simulating and applying nervous system activity [109]. McCulloch and Pitts connected the fact that both computers and the nervous system compute with binary signals, often referred to as "all-or-none" events or spikes in the nervous systems case. Therefore the same ingredients used in computer algorithms and mathematics could apply to the nervous system and vice versa. This led to the Perceptron, also known as the McCulloch-Pitts neuron, and later the Perceptron Learning Algorithm (PLA) by Rosenblatt

[109] [146]. A perceptron is a fundamental building block of an artificial neural network (ANN), representing a single neuron that takes multiple input values, applies weights to them, sums them up, and then passes the result through an activation function to produce an output. The PLA adjusts the weights of a perceptron based on the difference between the predicted output and the desired output, aiming to minimize this difference iteratively during training. Perceptrons that have multiple layers are called Multi-layer Perceptrons (MLPs) or Deep Neural Networks (DNNs) [150]. Eventually the MLP and PLA led to the now popular DNNs and algorithms that are based on them which are defining modern AI such as the now famous GPT models [17]. While DNN algorithms are so wide spread in use today they are currently almost synonymous with the areas of machine learning and artificial intelligence it took many decades since McCulloch and Pitts for researchers and practitioners to adopt them. This was primarily due to three things. The first was the misconception that Perceptron based algorithms could not solve complex problems such as an XOR function [115], which is true if the network does not have any hidden layers but it was incorrectly assumed that even with hidden layers the problem could still not be solved. The second being traditional computing hardware designed for general-purpose computation is not well-suited for implementing these algorithms, as they are energy-intensive and limited in terms of parallelism so it took a long time for hardware to be able to practically implement DNN algorithms at the necessary scale to be more useful. The third was that effective techniques and algorithms for training and applying DNNs had not yet been developed making them difficult to use and requiring human assistance.

Learning in neural networks is performed fundamentally with gradients. Quite simply, a gradient or derivative, is the change of something at a point in time. Mculloch and Pitts understood learning as changes across points of time which applies to neurons and neural networks as well [109]. Building upon their work Geoffery Hinton and his colleagues developed the backpropagation algorithm for learning with deep neural networks [150]. Backpropagation, or gradient descent, is the method by which the chain rule is applied to a loss at the

output to determine how much to change a synaptic weight, and is fundamentally how learning is performed in DNNs. Hinton demonstrated the importance of learning from data and the need of large labeled datasets which were not readily available. Because of this, Hinton's work also paved the way for techniques in unsupervised and self-supervised learning and demonstrated the importance of pre-training [59] [58] [151]. Yann LeCun is another deep learning pioneer, who developed the MNIST handwritten digit recognition dataset alongside a type of neural network architecture inspired by the human visual system for feature extraction and learning spatial hierarchies called the Convolutional Neural Network (CNN) [92] which propelled the field of machine learning with DNNs forward by demonstrating high performance on tasks such as image classification [84] and object recognition [144]. LeCun additionally contributed to the advancement of backpropagation algorithms by introducing techniques such as weight initialization and gradient clipping [92]. Since then LeCun has made other prominent contributions to the field particularly in unsupervised learning and information representation with Generative Adversarial Networks (GANs) [51] and sparse coding [53].

Because of the amazing work of Hinton, LeCun, and others, the field of artificial intelligence has seen tremendous growth in recent years, with deep neural network learning algorithms being applied to solve complex problems in a wide range of domains such as object detection, user recommendation, natural language generation, and autonomous vehicles. However, traditional computing hardware designed for general-purpose computation based on the von Neumann architecture is not well-suited for implementing these algorithms, as they require tight integration between memory and compute, something that biological neurons implement naturally but von Neumann architectures are not designed for causing them to be energy-intensive and limited in terms of parallelism [120, 77]. Parallel to the growth of Artificial Intelligence, visionary researchers began to question if a digital von Neumann architecture is the most effective and efficient way of computing leading to new fields exploring novel ways to compute such as quantum computing [101], analog computing [110], and

neuromorphic computing [111]. Neuromorphic computing has been proposed as a potential solution to the problems seen in von Neumann architectures, as neuromorphic architectures are designed to take greater biological inspiration by mimicking the structure and function of biological neurons and synapses. Neuromorphic from Latin consists of "neuro", brain, and "morphic", in the form of, with the intention to create hardware and software in the form of the brain seeking to emulate rather than simulate. Carver Mead is considered the founder of the field of neuromorphic computing who explored the potential of analog VLSI and introduced the concept of "silicon neurons," which paved the way for research on neuromorphic computing and artificial neural networks (ANNs) [110][111]. The silicon neurons that take inspiration from the brain's event driven dynamics are referred to as spiking neurons, and can form spiking neural networks (SNNs). Because the all or none events of spiking neurons are not differentiable they cannot normally use the same ingredients that make deep learning with ANNs successful. However, the recent development of surrogate gradient methods have enabled deep SNNs to learn with gradient based learning methods and take advantage of the techniques and model architectures developed for training and deploying ANNs [125] enabling SNNs to achieve comparable performance to ANNs [160] [165] [79]. SNNs are deployable neuromorphic hardware's unique architecture that offers a number of advantages over traditional computing hardware, including lower energy consumption, high parallelism, and the ability to implement learning algorithms directly in hardware [68] [124] [64] [148].

The work presented in this thesis takes inspiration from both Deep Learning and Neuromorphic Computing to develop deep SNN learning algorithms towards creating intelligence at the edge. We investigate the potential of neuromorphic hardware for machine learning with deep neural networks and to develop algorithms and systems that can harness the strengths of this technology by enabling low-power, rapid online learning from streamed data at the edge. We aim to address some of the challenges that arise in implementing machine learning algorithms on neuromorphic hardware, such as the limited memory and processing resources, the need for efficient data representation, and the difficulties in training neural networks in

an event-driven manner. By exploring these challenges, we aim to demonstrate the feasibility of using neuromorphic hardware for learning and to establish it as a valuable platform for building intelligent systems.

1.1 Outline of the Thesis

Aside from being a capstone of my work as a PhD student, the thesis is written with the intention of giving historical insights, food for thought, and some knowledge about AI, ML, neuromorphic computing, and in general learning to those interested. Chapter 1 here introduces the problem of learning, why it is interesting, and what can be done with it in the context of edge learning and devices. The scope narrows in Chapter 2, the background section, which gives an overview of topics related to the primary work presented in the thesis such as neural networks, gradient based learning, and digital neuromorhpic architectures. Chapter 3 begins the primary work of the thesis with applying few-shot learning in digital neuromorphic hardware [161] [162]. Chapter 4 expands upon the work from Chapter 3 with the application of meta-learning algorithms for improved and more generalizable few-shot learning [165]. Chapter 5 details some preliminary work on the development of self-supervised learning algorithms for use in neuromorhpic hardware [159]. Chapter 6 is the discussion and conclusion of the thesis with outlooks on potential future directions of the research presented in the thesis.

Thank you for considering reading my thesis. If even one of these chapters is helpful or interesting to you then taking the time to write this will not have been for naught.

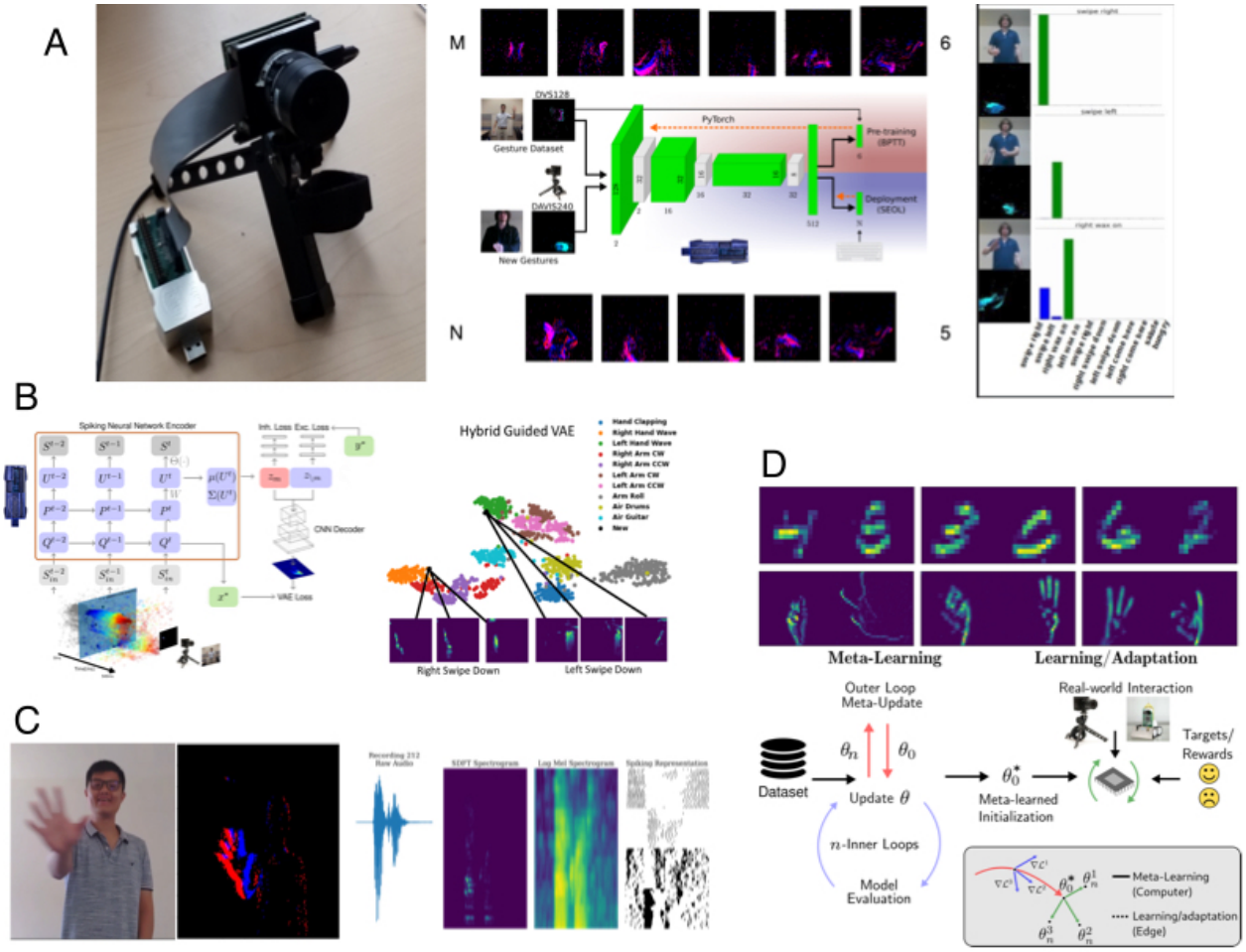


Figure 1.1: Graphic overview of the major work that will be presented in the thesis. A: The basic foundation of my work is formed in [161] and [162] discussed in Chapter 3 involves streaming event based data from a sensor to a neuromorphic processor with an SNN to perform few-shot online learning on unseen classes of data. B: The work shown in A needs labeled data for learning to eliminate the need for learning from labeled data; I developed an unsupervised learning inspired algorithm to pseudo label new data for online learning with a neuromorphic processor [159]. C: The previous work in A and B focused on two already made event vision datasets; therefore I created my own dataset consisting of both frame based and event based data, and I started working with signal data such as audio [166]. D: I developed meta learning for SNNs to improve few-shot learning and rapidly adapt to unseen classes within a domain [165].

Chapter 2

Background

Key concepts central to the research work will be detailed in the following sections. They will serve as a reference or tutorial for those unfamiliar with these concepts.

2.1 Deep Neural Networks

Deep Neural Networks have defined AI and Machine Learning over the last decade. From the humble Multi-Layer Perceptron (MLP) to the mighty transformer, numerous neural network architectures and algorithms have been developed and improved upon over the years. Deep neural networks (DNNs) are a class of artificial neural networks (ANNs) designed to model and learn complex patterns and representations from data. They are inspired by the structure and functioning of the human brain. The "deep" in deep neural networks refers to the presence of multiple layers of interconnected nodes (neurons) between the input and output layers.

Let's break down the key components of DNNs:

1. **Neurons (Nodes)**: Neurons are the fundamental building blocks of neural networks. They are inspired by the neurons in the human brain and are responsible for processing and transmitting information. Each neuron receives input data, performs a mathematical operation on it, and passes the result to the next layer.

2. **Layers**: A deep neural network consists of multiple layers of neurons. The three main types of layers are:

a. *Input Layer*: The first layer that receives raw input data (e.g., images, text, audio).
b. *Hidden Layers*: Layers that come after the input layer. They process and transform the data before passing it to the next layer.
c. *Output Layer*: The final layer that produces the network's predictions or outputs. The number of neurons in the output layer depends on the problem being solved (e.g., binary classification, multi-class classification, regression).

3. **Weights and Biases**: Each connection between neurons has an associated weight. These weights determine the strength of the connection and are adjusted during the learning process. Additionally, each neuron usually has an associated bias, which allows the network to learn transformations beyond simple linear operations.

4. **Activation Function**: An activation function is applied to the output of each neuron in the hidden layers. It introduces non-linearity to the network, enabling it to learn complex relationships between inputs and outputs. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh.

5. **Forward Propagation**: During the forward propagation phase, the input data is fed through the network, and each layer performs its computations and passes the output to the next layer until the final output is generated.

6. **Loss Function**: The loss function measures the difference between the predicted output

and the actual target (ground truth). The goal of training the network is to minimize this loss function, which indicates how well the network is performing on the given task.

7. **Learning Rule: Backpropagation**: Backpropagation is the core learning algorithm used to train deep neural networks. It involves calculating the gradients of the loss function with respect to the weights and biases of the network. These gradients indicate the direction and magnitude of the necessary adjustments to the parameters to minimize the loss. By iteratively updating the weights and biases using optimization algorithms like gradient descent, the network learns to make better predictions.

8. **Training**: The process of training involves repeatedly feeding the training data through the network, computing the loss, and updating the network's parameters using backpropagation. The goal is to optimize the network's parameters θ to make accurate predictions y on unseen data x such that the network learns the function $y = f(x, \theta)$.

Deep neural networks have demonstrated exceptional performance in various fields, such as image and speech recognition, natural language processing, game playing, and more. Their ability to automatically learn hierarchical representations from data allows them to handle complex tasks that were previously challenging for traditional machine learning algorithms. However, training deep neural networks often requires a large amount of data, immense computational power, and careful tuning to avoid issues like over-fitting.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized type of deep neural network designed primarily for processing grid-like data, such as images and videos. CNNs have revolutionized the field of computer vision by enabling computers to learn and extract complex features from visual data, leading to significant advancements in tasks like image classifica-

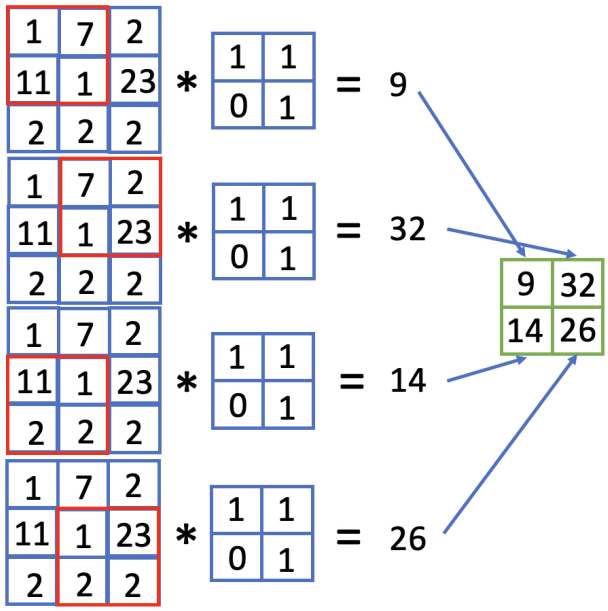


Figure 2.1: An example of the convolution operation. A matrix is filtered with a kernel, a smaller sub-matrix, using element wise multiplication. The convolution layers of CNNs utilize the convolution operation to learn hierarchical features.

tion, object detection, image generation, and more.

The key innovation of CNNs lies in their ability to automatically learn hierarchical features directly from raw data, without the need for manual feature engineering. This is achieved through the use of convolutional layers, pooling layers, and fully connected layers.

Here's how a typical CNN is structured:

1. **Convolutional Layers**: These layers are the heart of CNNs. They use the convolution operation which consists of small filters (also called kernels) that slide across the input data (usually an image), performing element-wise multiplication and aggregation to produce a feature map. These filters detect different patterns such as edges, textures, and simple shapes. An example of the convolution operation is shown in Figure 2.1.Convolutional layers are designed to be translation invariant, meaning they can recognize patterns regardless of their position in the image.

2. **Pooling Layers**: After a series of convolutional layers, pooling layers are often used to reduce the spatial dimensions of the feature maps. Pooling involves selecting a representative value (e.g., maximum or average) from a small region of the feature map. This reduces the computational complexity of the network and helps the model become more robust to small variations in the input.

3. **Activation Functions**: Non-linear activation functions (e.g., ReLU, sigmoid, tanh) are applied element-wise to the output of convolutional and fully connected layers. They introduce non-linearity to the network, allowing it to learn complex relationships in the data.

4. **Fully Connected Layers**: These layers are used to combine the features learned from the previous layers and make final predictions. In classification tasks, fully connected layers often culminate in a softmax activation, which produces a probability distribution over different classes.

CNNs can have multiple such layers stacked together, forming a deep architecture that learns increasingly abstract features as the network gets deeper. This ability to learn hierarchical representations is what makes CNNs particularly effective for visual tasks, as they can automatically learn to detect basic features like edges and combine them to recognize more complex structures like shapes and objects.

Some notable CNN architectures include:

- **LeNet-5**: One of the earliest CNN architectures designed by Yann LeCun for handwritten digit recognition [92].
- **AlexNet**: A landmark CNN architecture that gained significant attention after winning the ImageNet Large Scale Visual Recognition Challenge in 2012 [84].
- **VGG**: Known for its simplicity and use of small 3x3 filters in deep architectures [156].
- **ResNet**: Introduced residual connections to address the vanishing gradient problem in very deep networks [56].

- **Inception (GoogLeNet)**: Known for its utilization of "Inception" modules, which use multiple filter sizes in parallel [169].

These architectures have inspired subsequent developments and continue to be the foundation for many state-of-the-art models in computer vision tasks.

2.1.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [133] are a class of artificial neural networks designed to process sequences of data with temporal dependencies. Unlike feedforward neural networks, which process fixed-size inputs independently, RNNs maintain an internal state or memory that allows them to capture information from past inputs and use it to influence the processing of future inputs in the sequence.

The key feature of RNNs is their ability to create loops within the network, allowing information to persist over time. This recurrent connection enables them to exhibit dynamic temporal behavior and makes them well-suited for tasks involving sequential data, such as natural language processing, speech recognition, time series prediction, and more.

The basic structure of an RNN can be represented as follows:

Here:
- $x(t)$ represents the input at time step t.
- $h(t)$ denotes the hidden state (memory) of the RNN at time step t.
- $y(t)$ is the output at time step t.
- The arrows represent the flow of information through the network, and the recurrent connections allow the hidden state to influence the processing of future inputs.

The computation within an RNN can be described by the following equations:

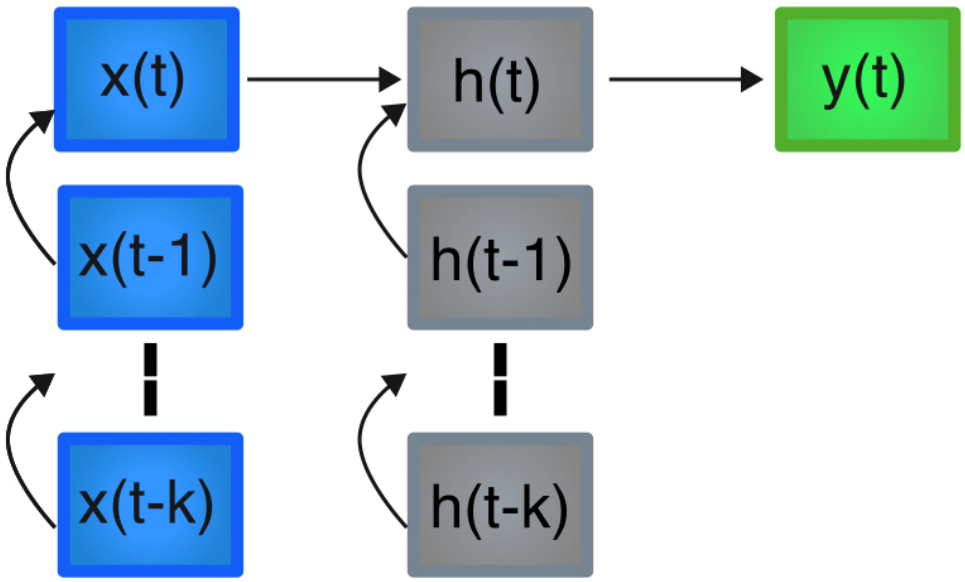


Figure 2.2: Basic structure of an RNN unrolled in time. Curved arrows represent recurrent connections.

1. **Hidden State Update**:

$$h(t) = f(W_{hx} \cdot x(t) + W_{hh} \cdot h(t-1) + b_h)$$

where:

- W_{hx} is the weight matrix connecting the input to the hidden state.
- W_{hh} is the weight matrix representing the recurrent connections.
- b_h is the bias term for the hidden state.
- f is an activation function, commonly the tanh function or the ReLU function.

2. **Output Computation**:

$$y(t) = W_{yh} \cdot h(t) + b_y$$

where:

- W_{yh} is the weight matrix connecting the hidden state to the output.
- b_y is the bias term for the output.
- $y(t)$ can be used for making predictions or passed as input to subsequent steps in a sequence.

The above equations illustrate the recursive nature of RNNs, as the hidden state $h(t)$ depends on the current input $x(t)$ and the previous hidden state $h(t-1)$.

However, traditional RNNs suffer from the vanishing and exploding gradient problems, limiting their ability to capture long-range dependencies. To address these issues, various advanced RNN variants have been developed, including Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs). These variants use specialized gating mechanisms to better retain and update information over long sequences, making them more effective for a wide range of sequential tasks.

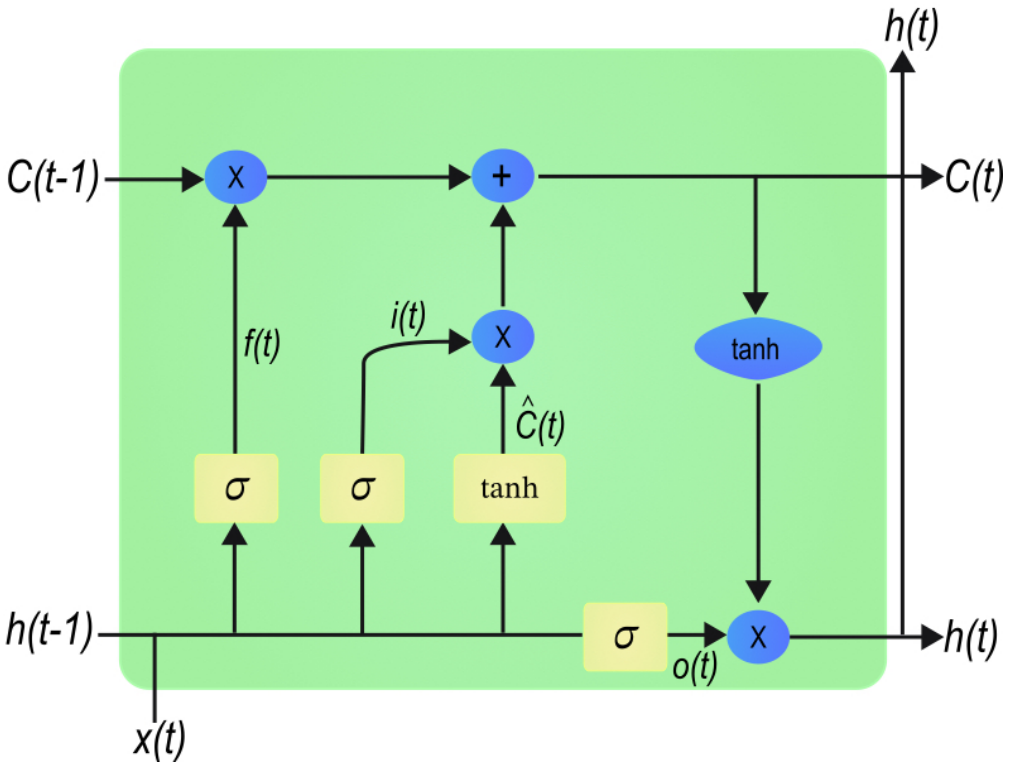


Figure 2.3: Basic structure of an LSTM unrolled in time.

2.1.3 Long Short Term Memory

Long Short-Term Memory (LSTM) [52] is a specialized variant of RNNs designed to address the vanishing and exploding gradient problems that can occur in traditional RNNs. LSTMs are particularly effective in capturing long-range dependencies in sequential data and have become a standard choice for various tasks involving sequential information, such as natural language processing, speech recognition, and time series analysis.

The key innovation in LSTM is the incorporation of a memory cell, which allows the network to learn when to store and when to forget information over time. This memory cell, along with various gating mechanisms, enables LSTMs to selectively retain important information and mitigate the vanishing gradient problem that plagues traditional RNNs.

The core idea of LSTMs is best illustrated by the following components:

1. **Cell State ($C(t)$)**: The cell state acts as the long-term memory of the LSTM. It runs horizontally through the entire sequence and is modulated by various operations at each time step. It serves as a pathway for information flow with minimal interference.

2. **Hidden State ($h(t)$)**: The hidden state is similar to the hidden state in traditional RNNs. It captures the relevant information from the current input and the previous hidden state. However, LSTMs have better control over the information stored in $h(t)$ due to the presence of the cell state.

3. **Gates**: LSTMs use three types of gates to control the flow of information: forget gate ($f(t)$), input gate ($i(t)$), and output gate ($o(t)$).

a. *Forget Gate ($f(t)$)*: The forget gate decides which information from the previous cell state ($C(t-1)$) should be discarded. It takes the current input $x(t)$ and the previous hidden state $h(t-1)$ as inputs and produces a value between 0 and 1 for each element of the cell state. A value close to 0 indicates that the information should be forgotten, while a value close to 1 indicates that it should be retained.

b. *Input Gate ($i(t)$)*: The input gate determines what new information from the current input $x(t)$ should be added to the cell state. It takes the same inputs as the forget gate and produces a value between 0 and 1 to determine the relevance of each element in the cell state.

c. *Output Gate ($o(t)$)*: The output gate decides which information from the updated cell state ($C(t)$) should be output as the hidden state $h(t)$. It takes the current input $x(t)$ and the previous hidden state $h(t-1)$ as inputs and produces a value between 0 and 1 for each element of $C(t)$.

4. **Update Equations**: The LSTM update equations for each time step t are as follows:

a. *Compute the Forget Gate*:

$$f(t) = \sigma(W(f) \cdot [h(t-1), x(t)] + b(f))$$

b. *Compute the Input Gate*:

$$i(t) = \sigma(W_i \cdot [h(t-1), x(t)] + b_i)$$

c. *Compute the Candidate Cell State*:

$$\hat{C}(t) = \tanh(W_C \cdot [h(t-1), x(t)] + b_C)$$

d. *Update the Cell State*:

$$C(t) = f(t) \odot C(t-1) + i(t) \odot \hat{C}(t)$$

e. *Compute the Output Gate*:

$$o(t) = \sigma(W_o \cdot [h(t-1), x(t)] + b_o)$$

f. *Compute the Hidden State*:

$$h(t) = o(t) \odot \tanh(C(t))$$

Where:

- $W(f), W_i, W_C, W_o$ are weight matrices, and $b(f), b_i, b_C, b_o$ are bias vectors associated with the gates.

- σ is the sigmoid activation function, and $\odot$ represents element-wise multiplication.

By using these gating mechanisms, LSTMs can selectively update the cell state and the hidden state, allowing them to capture long-range dependencies and avoid the vanishing gradient problem. This makes LSTMs effective for tasks involving sequential data with complex dependencies over time.

2.1.4 Spiking Neural Networks

Neuromorphic computing systems frequently employ Spiking Neural Networks (SNNs) [63]. In SNNs, neurons communicate through discrete spikes or pulses of activity, emulating the

firing behavior of real neurons in the brain. SNNs can be modeled as a special case of artificial RNNs with internal states akin to the LSTM [125]. The two key differences between RNNs and SNNs are that SNNs use neuron dynamics for their activation function and output spikes.

Due to the complex chemical interactions between neurons involved in their computation there are numerous ways of modeling the dynamics of neurons. One of the simplest models that seeks to capture the dynamics of sodium-potassium ion pumps in nerve cells is the Leaky Integrate and Fire (LIF) neuron [66]. A popular variation of the LIF neuron is the CUrrent BAsed (CUBA) LIF model used by neuromorphic processors such as Intel's Loihi [32]. The CUBA LIF model has two internal state variables: the synaptic response current u(t) and the membrane potential v(t). The synaptic response current is the sum of filtered input spike trains and a constant bias current. The CUBA LIF neuron dynamics are described as follows:

$$\begin{aligned} u[t] &= (1-\alpha_u)\, u[t-1] + x[t], \\ v[t] &= (1-\alpha_v)\, v[t-1] + u[t] + \text{bias}, \\ s[t] &= v[t] \geq \vartheta, \\ v[t] &= v[t]\,(1-s[t]) \end{aligned} \tag{2.1}$$

where $u[t]$ is the synaptic current at time t, α_u is the exponential decay constant for the current, $x[t]$ is the input at time t, $v[t]$ is the synaptic voltage or membrane potential at time t, $s[t]$ is a post-synaptic spike, and ϑ is the threshold at which the neuron fires a spike. Input to neurons are referred to as pre-synaptic, while output is referred to as post-synaptic. An example of an SNN is shown in Figure 2.4 which shows a small SNN receiving spiking input and outputting spikes with example dynamics shown for one neuron.

The change in current is determined by the synaptic weight of the input, which can be positive (excitatory) or negative (inhibitory), and the timing of input spikes because the current will decay over time if no input is received. The change in membrane potential is

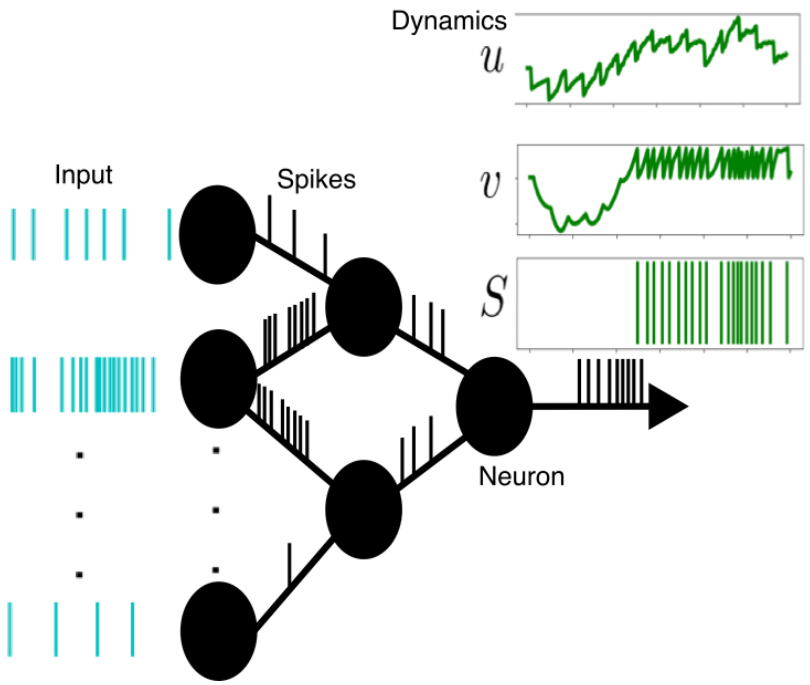


Figure 2.4: Example of an SNN. As neurons receive spike input the current and membrane potential increase, but if they don't receive input the current and membrane potential will decay. If the membrane potential reaches its threshold it will cause the neuron to output a spike.

influenced by the change in current over time, and whether or not the membrane potential reached its spike threshold because the membrane potential will not only decay over time but also reset if a spike is output. Because of these unique properties of the dynamics used in SNNs, the time dimension of data is highly important for processing data and learning. The timing of spikes encode information across time, and the timing information can be encoded into traces of neuron activity which are typically represented as exponential decay filters. This has lead to the development of learning rules that depend on time as a component of whether or not the weight of a synapse in an SNN changes.

2.2 Spike Time Dependent Plasticity

Spike-Timing-Dependent Plasticity (STDP), is a biologically inspired learning rule that governs the strength of the synaptic weights between spiking neurons [8]. STDP is a key mechanism by which synapses can undergo long-term changes based on the relative timing of spikes in the pre-synaptic and post-synaptic neurons [7]. The basic idea behind STDP is that the timing of spikes at the pre-synaptic and post-synaptic neurons determines whether the synaptic connection between them should be strengthened or weakened [24].

The STDP learning rule can be described as follows:

1. If the pre-synaptic neuron fires a spike shortly before the post-synaptic neuron fires a spike, the synapse's strength is increased. This is referred to as the "long-term potentiation" (LTP) phase. Essentially, when the pre-synaptic neuron contributes to the firing of the post-synaptic neuron, the synapse between them is strengthened.

2. On the other hand, if the post-synaptic neuron fires a spike shortly before the pre-synaptic neuron fires a spike, the synapse's strength is decreased. This is referred to as the "long-term depression" (LTD) phase. When the pre-synaptic neuron is less likely to contribute to the firing of the post-synaptic neuron, the synapse is weakened.

The time window over which the pre-post spike pair is considered relevant for synaptic changes is typically on the order of tens of milliseconds.

The change in synaptic weight, denoted as Δw, based on the timing of pre-synaptic and post-synaptic spikes is described as follows:

For pre-before-post spike pairing (LTP - Long-Term Potentiation), where $t_{\text{pre}} > t_{\text{post}}$:

$$\Delta w = X_{\text{plus}} \cdot f(t_{\text{pre}} - t_{\text{post}})$$

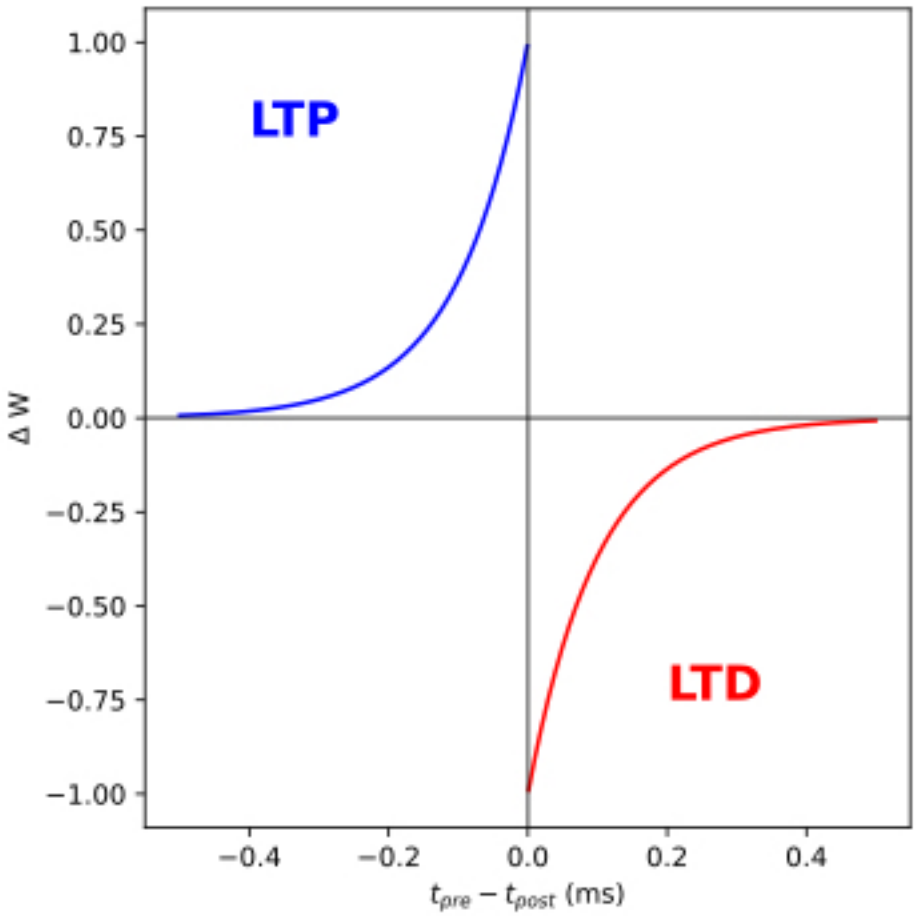


Figure 2.5: Example plot showing the resulting LTP and LTD from STDP learning. When t_{post} is higher LTP will happen while if t_{pre} is higher LTD will occur.

For post-before-pre spike pairing (LTD - Long-Term Depression), where $t_{\text{pre}} < t_{\text{post}}$:

$$\Delta w = -X_{\text{minus}} \cdot f(t_{\text{post}} - t_{\text{pre}})$$

The function $f(t)$ that determines the shape of the weight change curve is often modeled using an exponential function:

$$f(t) = \exp\left(-\frac{t}{\tau}\right)$$

Where τ is the time constant controlling the decay of the weight change effect as the time difference between spikes increases. The exponential function is in general used for decay dynamics for spiking neurons such as α_u and α_v in the CUBA LIF neuron model described in Equation 2.1 and in spike trace activity often used in learning rules for SNNs. Figure 2.5 shows an example plot of how LTP and LTD implements weight change Δw.

The concept of STDP has been widely studied and observed in experimental settings, such as brain slices and cultured neurons as well as in SNNs in neuromorphic hardware [67]. This learning rule has been found in various brain regions and is believed to be an essential mechanism in the development and plasticity of neural circuits. It allows neural networks to adapt their synaptic connections based on the specific patterns of activity they experience, reinforcing important connections while weakening less relevant ones.

2.3 Gradient Based Learning

Gradient-based learning is a fundamental technique used in training deep DNNs [91]. It involves adjusting the parameters of the network using the gradient of a loss function with respect to those parameters. The goal is to find the optimal values for the parameters that minimize the loss function, allowing the network to make accurate predictions.

The chain rule is a crucial mathematical concept used in gradient-based learning. It enables

us to calculate the gradients of complex functions, such as compositions of multiple functions, by breaking them down into simpler components. In a DNN, we have a set of parameters denoted by θ, which include the weights and biases of all the neurons in the network. Let $L(\theta)$ be the loss function that measures the difference between the predicted output y_{pred} and the actual target y_{true}. The loss function typically depends on the network's predictions and the true targets.

The goal of training is to minimize the loss function with respect to the parameters θ. We achieve this by updating the parameters in the opposite direction of the gradient of the loss function, denoted as $\nabla_\theta L(\theta)$. The update step is performed iteratively using an optimization algorithm like gradient descent.

The update equation for the parameters during training can be expressed as follows:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$$

where α is the learning rate, a hyperparameter that determines the step size during each update.

The chain rule is used to calculate the derivative of a composite function. Suppose we have two functions $f(x)$ and $g(x)$, and we want to find the derivative of their composition $h(x) = f(g(x))$.

The chain rule states that the derivative of the composite function $h(x)$ with respect to x is the product of the derivative of f with respect to its argument evaluated at $g(x)$, and the derivative of g with respect to x:

$$\frac{dh}{dx} = \frac{df(g(x))}{dg(x)} \cdot \frac{dg}{dx}$$

In the context of gradient-based learning, the chain rule is used to calculate the gradients of the loss function with respect to the parameters of the neural network. The loss function is

a composition of the network's predictions and the true targets, which are both functions of the network's parameters.

By applying the chain rule iteratively from the output layer to the input layer, we can efficiently calculate the gradients of the loss function with respect to all the parameters in the network. This process is known as backpropagation, and it forms the basis for training deep neural networks.

2.3.1 Surrogate Gradients

While gradient Back-Propagation is the workhorse for training nearly all deep neural network architectures, it is generally incompatible with non-Von Neumann computers, including brains and neuromorphic hardware. By identifying Spiking Neural Networks (SNNs) as a type of Recurrent Neural Network (RNN), recent studies showed two reasons for this incompatibility [125]. Firstly, the spiking neuron has a non-differentiable activation function, which prevents the gradients from flowing across the network. Secondly, the computation of local errors requires the evaluation of a global loss function, which is a spatially and temporally non-local computation.

These problems are addressed using Surrogate Gradient (SG) learning [125]. With surrogate gradients the chain rule can be applied as well allowing for auto-differentiation to be used for training SNNs with Surrogate Gradient Descent. Assuming a global cost function $\mathcal{L}(S^N)$ defined on the spikes S^N of the top layer and targets Y, the gradients with respect to the weights in layer l are [163]:

$$\nabla_{W_{ij}} \mathcal{L}(S^N) = \frac{\partial \mathcal{L}(S^N)}{\partial S_j} \frac{\partial S_j}{\partial v_j} \frac{\partial v_j}{\partial W_{ij}}. \tag{2.2}$$

As discussed earlier, training deep SNNs is challenging due to a spatio-temporal credit assignment problem and non-differentiability of the spiking dynamics. Previous work overcame

the learning problem in multiple layers of SNNs with methods such as feedback alignment [123, 100], backpropagation-through-time (BPTT) [154, 194, 62], and spike-based backpropagation [172, 94]. The successful gradient-based training of deep SNNs usually approximate the spiking function's derivative using a surrogate activation function [125]. By being able to train deep SNNs, the ingredients of deep learning that make ANNs successful such as dropout, batch normalization, convolutions, pooling etc. can be applied to SNNs, in addition to SNNs being compatible with neuromorphic hardware. BPTT is shown to achieve state-of-the-art accuracy on certain target domains such as NMNIST and gesture recognition. However, BPTT is inherently offline as it propagates error through the unrolled network and therefore is not suited for applications where online adaptation is desired.

Surrogate Gradient (SG) methods offer a novel approach to calculating weight updates in a local and differentiable manner by introducing a surrogate network. These methods formulate weight updates as three-factor synaptic plasticity rules, reveals from first principles the mathematical nature of the three factors and a learning dynamic that is temporally continuous and compatible with synaptic plasticity. The three-factor rules consist of a pre-synaptic factor, a post-synaptic factor, and an external error signal. This stands in contrast to Spike Time Dependent Plasticity (STDP), a conventional synaptic plasticity model in neuroscience, which involves only two factors and lacks the external signal component [12]. The incorporation of the third factor in SG significantly enhances learning by effectively projecting task-specific errors to the neurons [9]. This allows for learning with an error driven loss function with gradients in a similar manner to ANNs. Thus SGs bridge the worlds of Artificial Neural Networks (ANNs) and Spiking Neural Networks (SNNs) without resorting to oversimplified assumptions about the latter. Training SNNs with SG methods pave the road towards neuromorphic learning machines with SNN performances similar to deep-learning [74], while being able to learn online, with input streams, spike timing and potentially using a fraction of the energy compared to conventional computers using ANNs.

Using a surrogate gradient approach that approximates the spiking threshold function for gradient estimations, SNNs can be trained to match or exceed the accuracy of conventional neural networks on event-based vision, audio, and reinforcement learning tasks [73, 27, 11, 14, 197, 123]. Interestingly, the surrogate gradient learning rule is compatible with synaptic plasticity in the brain so long as there exists a process that computes (or approximates) and communicates $\frac{\partial \mathcal{L}(s^t)}{\partial s_i^t}$ to neuron i. Whether and how this can be achieved in a local and biologically plausible fashion is under debate, and there exist several methods to approximate this term on a variety of problems [198]. For instance, direct feedback alignment is an important candidate method to overcome this problem in SNNs [123].

2.4 Types of Learning

There are many different approaches to solving problems using deep neural networks. These different approaches involve different classes of learning algorithms. The classes of learning algorithms used in the work of this thesis will be described as follows.

Supervised learning is a machine learning paradigm where an algorithm is trained on a labeled dataset [102]. The process involves using input-output pairs (features and corresponding target labels) to teach the algorithm to make accurate predictions when presented with new, unseen data. During training, the algorithm learns the underlying patterns and relationships within the data, allowing it to generalize and make predictions on previously unseen examples. In summary, supervised learning is about teaching an algorithm to map inputs to outputs based on labeled examples, enabling it to make predictions on new, similar data. Applying DNNs to solve problems gained traction when applied to image classification problems, with AlexNet famously outperforming non-DNN methods on the ImageNet image classification task [84]. Since then supervised learning with DNNs have been applied to a wide variety of problems such as object detection [144], speech recognition [121], facial

recognition [158], and gesture recognition [165].

Unsupervised learning is a machine learning approach where an algorithm is trained on an unlabeled dataset [22]. Unlike supervised learning, it doesn't have target labels to guide the learning process. Instead, the algorithm seeks to find patterns, structures, or relationships within the data on its own. It attempts to identify clusters or groupings of similar data points, detect anomalies, or reduce the dimensionality of the data. Unsupervised learning is mainly used for exploratory analysis and pattern recognition, where the goal is to uncover hidden insights and gain a better understanding of the data's underlying structure. DNN architectures used for unsupervised learning include Variational Auto Encoders (VAEs) [80], Generative Adversarial Networks (GANs) [51], and Gaussian Mixture Models [180] among others.

Self-supervised learning is a type of machine learning where an algorithm learns from unlabeled data by creating its own pseudo-labels through a pretext task [129]. Unlike traditional supervised learning, where external annotations are used for training, self-supervised learning leverages the inherent structure or information present within the data itself. The algorithm generates pseudo-labels by defining a task that is easier to solve, such as predicting missing parts of the input data or transforming the data in a specific way. Once the algorithm has learned from the pretext task, it can then transfer this knowledge to the main task of interest. By doing so, self-supervised learning can efficiently leverage large amounts of unlabeled data, which is often more abundant and easier to acquire than fully labeled datasets. Self-supervised learning has shown promising results in various domains, including computer vision, natural language processing, and speech recognition. Examples of self-supervised learning methods include Siamese Nets [82], and Barlow Twins [196].

Reinforcement learning is a type of machine learning that deals with decision-making in an interactive environment [96]. It involves an agent that learns to take actions to maximize a cumulative reward over time. The agent interacts with the environment and receives feedback

in the form of rewards or penalties based on its actions. The goal of reinforcement learning is for the agent to learn an optimal policy—a strategy that maps states to actions—so that it can make the best decisions in different situations. The agent explores the environment by trying out different actions and learns from the consequences of those actions. Through a trial-and-error process, sthe agent updates its policy using a learning algorithm, such as Q-learning or Deep Q Networks (DQNs), to improve its decision-making abilities over time. Reinforcement learning has been successful in solving various complex problems, including game playing, robotics, autonomous vehicles, and resource optimization, among others [88].

Continual learning, also known as lifelong learning or incremental learning, is a machine learning paradigm focused on enabling models to learn from a continuous stream of data and adapt to new information over time [170]. Unlike traditional static datasets, continual learning deals with dynamic data distributions and the emergence of new tasks. The key challenge is to retain knowledge from previous learning experiences while accommodating new knowledge without catastrophic forgetting. It involves developing techniques such as regularization, replay, generative models, and knowledge distillation to ensure that the model can efficiently adapt and improve its performance as it encounters new data and tasks over its lifetime. Continual learning has significant implications for creating AI systems that can continually learn and evolve, making them more versatile and adaptive in real-world applications. Continual learning methods include but are not limited to Few-shot Class Incremental Learning (FSCIL) [170], Meta Few-shot Class Incremental Learning (Meta-FSCIL) [23], and Intelligent Synapses [199].

2.4.1 Offline Vs Online Learning

The typical way in which DNNs are trained is where first a batch data is fed into the network propagating forward through the layers until the output is obtained. Then the loss

is calculated from the output and through chain-rule auto-differentiation the gradients are propagated backwards through the network to update the network parameters. This process is called offline learning.

Online learning involves learning directly from data as it is streamed to the network. Rather than waiting for all information to propagate forward and backwards through the network, parameters are updated with information local to neurons to more efficiently update enabling parameter updates as data is being input to the network. Being able to learn online can enable personalized intelligence with learning capabilities at the edge in real-time from sensory data with advantages such as lower power, data privacy, and usability. However there are several disadvantages to learning online that have made it impractical for use in DNNs. Firstly, learning via (stochastic) gradient descent requires data to be sampled in an independent and identically distributed fashion [178]. However, when sensory data is streamed and processed during task performance, data samples are generally correlated, leading to many convergence problems, including catastrophic forgetting [108]. Secondly, streaming data effectively requires a batch size of one. While networks with a batch size equal to one eventually converge [90], using a smaller batch size means that learning rates must be smaller as well. Smaller learning rates result in smaller weight updates which require memories or buffers that store the weights or weight updates with higher precision, and hence require more hardware area. Thirdly, surrogate gradient-based SNN training inherits other fundamental issues of deep learning, namely that very large datasets and a large number of iterations are necessary for convergence. The combination of the three problems stated above, *i.e.,* correlated data samples, data inefficiency, and memory requirements hamper the successful deployment of online learning in neuromorphic hardware to solve real-world learning problems. The work presented in Chapters 3 and 4 will address the aforementioned problems to enable a more practical and effective approach to online learning in neuromorhpic hardware.

2.5 Few Shot Learning

The brain's remarkable ability to tackle challenging recognition tasks using only a few samples serves as a key capability of the brain [86]. Few-shot learning aims to achieve rapid adaptation when some prior knowledge of the task domain is available. Rather than iterating many times over many samples of data to learn, few-shot learning utilizes prior knowledge of the task domain to learn in one or just a few sample presentations of the data. Our primary objective is to bring this remarkable capability to neuromorphic hardware, enabling local learning and rapid adaptation in edge and mobile systems. This will make interactive learning tasks such as learning human gestures for device control or swiftly adapting to a user's voice possible. To attain high performance in few-shot learning, some form of pretraining is done to attain prior knowledge of the desired task domain and then the learned prior knowledge is transferred to help models rapidly learn new samples in few-shots.

2.6 Transfer Learning

Previous work has demonstrated that the initial layers of neural networks excel at learning general features, while deeper layers progressively focus on acquiring more task-specific features [195]. This hierarchical learning pattern is a key characteristic of deep learning architectures. Importantly, the general features learned by the early layers possess valuable transferable knowledge. This characteristic forms the foundation of transfer learning, wherein these general features can be effectively transferred to other networks for task-specific training of their deeper layers. By leveraging the knowledge gained from previous tasks, transfer learning significantly enhances the efficiency and effectiveness of training on new and distinct tasks. The capacity of neural networks to learn both general and task-specific features at different depths has led to the powerful concept of transfer learning. This approach leverages

shared knowledge across tasks, fostering faster and more robust learning for a wide range of applications and domains. We train an SNN with transfer learning in order to few-shot learn on neuromorphic hardware.

The concept of transfer learning proves highly valuable in enhancing the performance of neural networks. By initially training the first layers on a dataset to acquire general features, these learned features can then be transferred to a second network, which is specifically trained for a target task [195]. This transfer of knowledge leads to superior generalization in handling the target task compared to training without the benefit of the transferred lower layer features. Transfer learning is particularly advantageous in few-shot learning scenarios, where the target task possesses only a limited amount of training data. In such cases, if a similar task with a more extensive dataset is available, the general features learned on this larger dataset can be transferred to aid the few-shot learning process [179].

2.7 Meta Learning

Meta-learning, also known as "learning to learn," is a subfield of machine learning that focuses on improving the efficiency and effectiveness of learning algorithms. The core idea behind meta-learning is to enable a model to learn from multiple tasks or datasets in such a way that it can quickly adapt to new, previously unseen tasks with minimal training [42].

Traditional machine learning algorithms typically require large amounts of data and computing resources to achieve good performance on a specific task. However, in real-world scenarios, data can be limited, and the cost of obtaining new data or retraining a model for a new task can be high. Meta-learning aims to address these challenges by training models to become more adept at learning new tasks with fewer data points.

The central concept in meta-learning is the distinction between "meta-training" and "meta-

testing":

1. **Meta-Training**: During the meta-training phase, the model learns from a diverse set of tasks or datasets, each representing a different learning problem. The goal is to extract common patterns and generalizable knowledge from this diverse set of tasks.

2. **Meta-Testing**: After the meta-training phase, the model is evaluated on new tasks that it has never seen before. The model's ability to quickly adapt to and perform well on these unseen tasks is the primary measure of its success in meta-learning.

Meta-learning remains an active area of research with potential applications in various domains, particularly in situations where data is scarce, and the ability to learn quickly from new tasks is critical.

Model Agnostic Meta Learning (MAML) [42] is a general algorithm for deploying meta-learning with any type of deep learning model.

Define a neural network model $y = f(x, \theta)$ that produces an output batch y given its parameters θ and an input batch x. For simplicity, we focus here on classification problems, such that y represents logits and $\arg\max y$ is a class, although any supervised learning problem would be suitable. In the classification case, each batch consists of K samples of each class. The parameters θ are trained by minimizing a task-relevant loss function $\mathcal{L}(f(x, \theta), t)$, such as cross-entropy, where t is a batch of targets.

The goal of meta-learning is to optimize the meta-parameters of f, such as the initialization parameters noted as θ_0. This work makes use of the standard second-order MAML algorithm to meta-train the SNN. The standard MAML workflow is designed to optimize the parameters of a neural network model across multiple tasks in a few-shot setting. MAML achieves this using two nested optimization loops, one "inner" loop and one "outer" loop. The inner loop consists of a standard SGD update, where the gradient operations are traced for auto-

differentiation [54]. In the outer loop, an update is made using gradient descent on the meta-parameters.

To make use of MAML, it is essential to set up the experimental framework accordingly. Define three sets of meta-tasks: meta training $\mathcal{T}^{trn}$, meta validation $\mathcal{T}^{val}$, and meta testing $\mathcal{T}^{tst}$. Each task T of meta-task $\mathcal{T}$ consists of a training dataset $\mathcal{D}^{trn}$ and a test dataset $\mathcal{D}^{tst}$ of the form $\mathcal{D} = \{x_i, t_i\}_{i=1}^{M}$. Here x_i denotes the input data, t_i the target (label) and M is the number of target samples. In general, the datasets corresponding to different tasks can have different sizes, but we omit this in the notation to avoid clutter. During learning, a task is sampled from the training meta-task $\mathcal{T}^{trn}$ and N inner loop updates are made using batches of data sampled from $\mathcal{D}^{trn}$. The resulting parameters θ_N are then used to make the outer loop update using the matching test dataset $\mathcal{D}^{tst}$. During each inner loop update, one or more SGD update steps are performed over a task-relevant loss function $\mathcal{L}$:

$$\begin{aligned} \theta_{n+1}(\theta_n, \mathcal{D}^{trn}) &= \theta_n - \alpha \nabla_{\theta_n} \mathcal{L}(f(x, \theta_n), t), \\ &\text{where } \{x, t\} \in \mathcal{D}^{trn} \text{ for } n = 1, ..., N. \end{aligned} \tag{2.3}$$

Here n is the number of inner loop adaptation steps, α is the inner loop learning rate. Note the dependence of θ_{n+1} on the initial parameter set θ_0 at the beginning of the inner loop through the recursion. ∇_{θ_n} here indicates the gradient over the inner loop loss $\mathcal{L}$ on the network using parameters θ_n. The outer loop loss is defined as:

$$\mathcal{L}_{outer}(\theta_0) = \sum_{T \in P(\mathcal{T}^{trn})} \mathcal{L}(f(\theta_N(\theta_0, \mathcal{D}^{trn}), \mathcal{D}^{tst})), \tag{2.4}$$

where $T = \{\mathcal{D}^{trn}, \mathcal{D}^{tst}\}$. In practice the above expression is generally computed over a random subset of tasks rather than the full set $\mathcal{T}^{trn}$. Notice that the outer loop loss is computed over the test dataset $\mathcal{D}^{tst}$, whereas $\theta_N(\theta_0)$ is computed using the training dataset $\mathcal{D}^{trn}$, which is shown to improve generalization. The goal is to find the optimal θ_0, denoted θ_0^* such that:

$$\theta_0^* = \arg\min_{\theta_0} \mathcal{L}_{outer}(\theta_0). \tag{2.5}$$

Provided the inner loop loss is at least twice differentiable with respect to θ_0, the optimiza-

tion can be performed via gradient descent over the initial parameters θ_0, using a standard gradient-based optimizer using gradients $\nabla_{\theta_0}\mathcal{L}_{outer}(\theta_0)$. Successive applications of the chain rule in the expression above results in second order gradients of the form $\nabla^2_{\theta_n}\mathcal{L}(f(x,\theta_n),t)$. If these second-order terms are ignored, it is still possible to meta-learn using the method called first-order MAML (FOMAML) [41] and REPTILE [127].

2.7.1 Non-gradient based meta-learning

MAML is known to learn representations that are general across datasets rather than "learning to learn" [142]. The results of our experiments with freezing model layers showed this is the case for SNN MAML as well indicating the layers already contain good features at meta-initialization. Another meta-learning approach done with artificial recurrent neural networks is to train the optimizer itself modeled using an LSTM [5]. The underlying mechanism relies on the recurrent cell states that capture knowledge that is common across the domain of tasks.

The SNN based work [152] falls into this category. There, meta-learning was applied to SNNs trained using e-prop on arm reach and Omniglot tasks. The approach used for the meta-learning combined a Learning Network (LN) to carry out inner loop adaptation and a Learning Signal Generator (LSG) to carry out outer loop generalization that was both modeled by recurrent SNNs.

2.7.2 Meta-learning with ANNs

Recent work has given several approaches to achieve high performance on few-shot learning tasks without updating the networks. For example, Matching Networks [179] utilize two components to enable one-shot learning; a deep neural network augmented with memory,

and a training strategy tailored for one-shot learning. The Matching Network trains an associative memory implemented by an attention mechanism that stores prior information about training data–label pair associations to produce sensible test labels for unobserved classes.

Another salient example is the Siamese network [16], which was more recently applied to one-shot learning [82]. Siamese networks consist of two networks with a common body that learns similarities between the classes of two inputs. By comparing each sample in a small N-way K-shot test set, the Siamese network can be train to distinguish different classes of samples.

Compared to MAML, Matching networks and Siamese networks do not require updating the network. However, MAML was shown to outperform Matching Networks [41] and Siamese Networks [41, 97]. Furthermore, the fact that no backpropagation is required with frozen layers and that MAML-based approaches outperform the Siamese net, strongly suggest that Matching networks and Siamese networks are less interesting for few-shot learning than our demonstrated approach for neuromorphic hardware.

2.7.3 Non MAML SNN few-shot learning

MAML and other meta-learning algorithms are not the only methods by which to achieve high accuracy on few-shot learning tasks. Prior work have demonstrated using hybrid SNN-ANN models to achieve high performance on few-shot learning tasks without using MAML. One example is the work of [72] that used a Multi Time-Scale Optimization (MTSO) learning to learn approach through the combination of an SNN and an adaptive-gated LSTM trained by BPTT on the omniglot and DVSGesture [4] datasets. The method enabled using two different timescales for neural dynamics to achieve both short term and long term learning for few-shot learning.

Another example is the work from [192] and their Heterogeneous Ensemble-based Spiking Few-shot Online Learning (HESFOL) method that combined an information theoretic approach for training an SNN on features that were extracted from a convolutional ANN on the omniglot dataset and a robot arm motor control task. Their work achieved accuracy on the omniglot dataset comparable to ours on the N-omniglot dataset. Yang *et al.* also demonstrated their method can achieve high performance on noisy data. While both [72] and [192] achieve high few-shot learning performance on their datasets, both have ANN components that would not be implementable on neuromorphic hardware. The focus of our work in chapter 4 is to lay the foundation for meta-learning with MAML on neuromorphic learning machines.

2.7.4 Meta-learning for Neuromorphic Learning Machines

Meta-learning and transfer learning techniques are already presenting themselves as key tools for neuromorphic learning machines, particularly for devices at the edge which are typically more resource constrained than large cloud servers. The work by [147] developed an online-within-online meta-learning method via meta-gradient descent that is applicable for learning in neuromorphic edge devices without transfer learning. The method uses a three-factor local learning rule that does not use an offline pre-deployment training with backpropagation. This allows a fully online meta learning training method called Online-Within-Online Meta-Learning for SNNs (OWOML-SNN) that performs meta-updates on data streamed to the network. To do the meta updates, multiple SNN models are trained in parallel on the same number of datasets from data sampled by the family of tasks for the outer loop update, and a within-task dataset is used to update an inference SNN in an inner loop update. They evaluated their method on the Omniglot and MNIST-DVS datasets with 2-way 5 shot learning.

The work of [112] showed that plasticity can be optimized by gradient descent, or meta-learned, in large artificial networks with Hebbian plastic connections called differentiable plasticity. Network synapses store both a fixed component, a traditional connection weight, and a plastic component as a Hebbian trace that stores a running average of the product of pre- and post-synaptic activity modified by a coefficient to control how plastic the synapse is. Networks using the plastic weights were demonstrated to achieve similar performance to MAML and Matching Networks. A Hebbian-based hybrid global and local plasticity learning rule similar to the differentiable plasticity presented in [112] was applied with SNNs in the the Tianjic hybrid neuromorphic chip [189]. This work meta-optimized Hebbian-based STDP learning rules and local meta-parameters on the Omniglot dataset to examine the performance of the model in few-shot learning tasks on the Tianjic neuromorphic hardware.

Our work in chapter 4 extends the one of [189] by directly optimizing a putative three-factor learning rule on event-based data and surrogate gradients that could be used for few-shot learning in neuromorphic hardware. On the surface, our work has similarities with [189], in the sense that it applies meta-learning to SNNs and proposes this method for neuromorphic hardware. However, there are significant algorithmic differences and certain challenges addressed in our work that are not addressed in [189]. Firstly, although not directly stated in [189], their work used a first order method since their surrogate function is not twice differentiable. Secondly, [189] uses a Hebbian STDP rule in the inner loop update. However, our work makes use of a gradient descent update, which is equivalent to a three factor synaptic plasticity rule. Third, the datasets used for few-shot learning are not converted from images. Relatedly, their models operates at much coarser time scales (8 time steps for 100ms, $\Delta t = 12.5$ms), making their model in line with conventional recurrent neural networks rather than SNNs. In comparison, the mode of operation in this work uses a time step that is in agreement with existing neuromorphic accelerators ($\Delta t = 1$ms). While $\Delta t = 12.5$ms mode of operation renders the training more tractable for image-based datasets, it does not address a key challenge in neuromorphic processing which is the exploitation of

the high temporal precision in event-based neuromorphic sensors [45, 103].

2.8 Edge Learning

Online on-chip learning in an edge device refers to the process of training machine learning models on a small, low-power, and resource-constrained device that is located at the edge, that is the task environment, without relying on a remote data center or the cloud [162]. This type of learning enables the edge device to make decisions and carry out tasks in real-time, without the need for a constant connection to the cloud [65].

In online on-chip learning, the machine learning models are trained and updated continuously as the edge device collects new data. This allows the models to adapt to changes in the environment and improve their accuracy over time. For example, a self-driving car equipped with an edge device could use online on-chip learning to continuously update its driving models based on new sensor data.

One of the key benefits of online on-chip learning is that it can provide low latency and high accuracy for real-time applications, since the data is processed locally and decisions are made quickly. Additionally, this approach can reduce the amount of data that needs to be transmitted to the cloud, thus reducing the bandwidth requirements and latency associated with cloud-based machine learning.

2.9 Event Based Sensing and Data

Event based sensors output events to enable computation with events end-to-end for efficient biologically inspired computing. Carver Mead and his student Misha Mahowald developed the first biologically inspired event based sensor, the silicon retina [106]. Silicon retinas

emulate the function of the human retina, detecting changes in lighting intensity and emitting an event at the pixel the intensity change occurred. A side by side comparison of standard RGB frame based data and event data from a silicon retina, or Dynamic Vision Sensors (DVS) [33], is shown in Figure 2.6. The event based image shown on the right only shows pixels where change has been detected which is typically in response to motion from either the sensor itself or something moving in the scene in this case the performance of a gesture. There have been several iterations of DVS including the DAVIS 128 [99], and Prophesee [136]. These have been used to create event based vision datasets such as the DvsGesture [4], NMNIST [132], and Megapixel Automotive Detection Dataset [136]. DVS sensors detect brightness changes on a logarithmic scale with a user-tunable threshold, instead of RGB pixels like typical cameras as follows

$$log(I_t) - log(I_{t-1}) >= \theta \tag{2.6}$$

where I is the intensity at time t and θ is the threshold needed to be met to output an event. An event consists of its location x, y, timestamp t, and the polarity $p \in [\text{OFF}, \text{ON}]$ representing the direction of change. Using a dense representation, the DVS event stream is denoted $S^t_{DVS,x,y,p} \in \mathbb{N}^+$, indicating the number of events that occurred in the time bin $(t,t+\Delta t)$ with space-time coordinates (x,y,p,t). Δt is the temporal discretization and equal to 1ms in our experiments. Each pixel of a DVS quantizes local relative log-scale intensity changes to generate spike events whenever the change exceeds a threshold θ [98]. The events can be streamed to an SNN for learning the spatio-temporal information embedded in the DVS output.

Silicon cochlea models for audio processing were among the very first neuromorphic devices designed [185] and benchmarks for speech recognition using these models appeared shortly thereafter [126]. These models closely emulate the functionality of the biological cochlea, receiving analog audio input and asynchronously generating spikes across various output channels representing distinct frequency bands. Over the years, both hardware- and

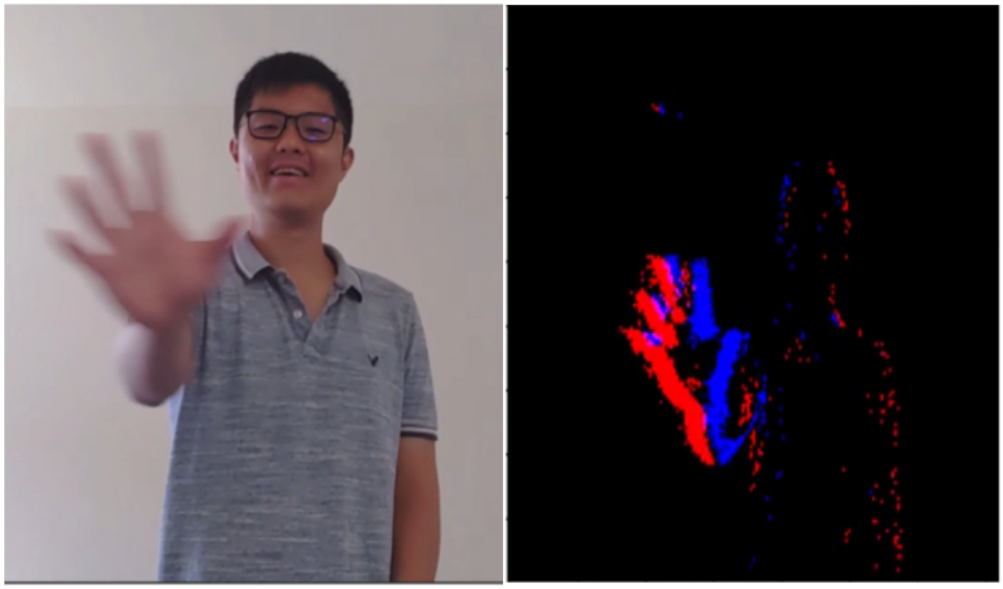


Figure 2.6: Left: Frame based image of a gesture. Right: Event based image of the same gesture recorded by a DAVIS 128 sensor.

software-based silicon cochlea models have been developed, contributing to the creation and benchmarking of speech event data [185, 176, 104, 55, 28]. The versatility of these models is evident as they have been effectively employed in diverse speech recognition tasks, including speaker identification [18, 95], spoken digit recognition [1, 6], and command word recognition [28]. Recently the speech2spikes pipeline was introduced as a tool for the efficient creation of spiking audio datasets from existing datasets, and for real-time encoding of raw audio to spikes for edge audio processing with better performance than current cochlea based models [166].

Because the nature of event based data is derived from signal processing, any signal data can be processed to generate events for event based processing. Several techniques for this are available such as delta encoding [40], and sigma-delta modulation [131]. These techniques enable the use of SNNs and neuromorphic hardware for signal processing applications such as biomedical applications, and can be used when specialized event sensor hardware is not available.

2.10 Neuromorphic Processors

There are currently numerous neuromorphic processors made with different goals in mind for how compute should be handled. They broadly fall into three classes: digital, analog, and mixed-signal. Digital neuromorphic processors compute using typical silicon hardware and are compatible with such hardware. What differentiates digital neuromorphic processors as neuromorphic is the architecture is designed with principles of neuromorphic computing in mind such as parallel event based computation and in-memory computing [70]. This allows for spiking neural network based algorithms prototyped in traditional hardware to be easily transferable to and run far more efficiently in neuromorphic hardware. Examples of digital neuromorphic processors include Intel's Loihi [32], IBM's True North [2], and

BrainChip's Akida [139]. Analog neuromorphic processors do away with the traditional binary signal processing of Von Neumann architectures in favor of the ability to compute with analog values. Examples of analog neuromorphic processors include Innaterra and Rain Neuromorphic's chips. Mixed signal chips use both analog and digital components to try and get the best of both worlds. The Xilinx mixed signal FPGA, and Texas Instruments' MSP430 are examples of mixed signal chips. These are just some examples of neuromorphic processors, for more details refer to these reviews [191, 171, 43, 122].

The work of the thesis is done exclusively with digital neuromorphic processors, namely Intel's Loihi neuromorphic processor. The following subsections describe examples of digital neuromorphic processors with a deeper dive into the Intel Loihi and how the different components of the processor lend itself well to online on-chip learning at the edge and beyond.

2.11 Digital Neuromorhpic Processors

IBM True North

The IBM True North [2] is among the first digital neuromorphic processors created by a large corporation. The TrueNorth chip employs a mix of asynchronous and synchronous circuitry and is fully configurable in terms of connectivity and neural parameters to allow custom configurations for a wide range of cognitive and sensory perception applications. Researchers successfully demonstrated the use of TrueNorth-based systems in multiple applications, including visual object recognition, with higher performance and orders of magnitude lower power consumption than the same algorithms run on von Neumann architectures [2]. The TrueNorth has also been used in applications such as mobile robot to navigate in an unfamiliar environment [64], primate cortex modeling [105], and medical image segmentation [117].

SpiNNaker

SpiNNaker, or Spiking Neural Network Architecture [44], was designed with a focus on scalability and energy efficiency, incorporating communication methods inspired by the brain. SpiNNaker was developed as part of the Human Brain Project, a 10 year large-scale research initiative that pioneered digital brain research. It is capable of simulating large neural networks and executing event-based processing for various applications. SpiNNaker offers two circuit board options: a smaller 4-node board (equivalent to 64,000 neurons) and a larger 48-node board (equivalent to 768,000 neurons). The 48-node board consumes 80 W of power. A second iteration called SpiNNaker 2 [107] increased the number of neurons 10 fold to 10 million neurons, and incorporated features such as workload adaptivity to optimize power efficiency. The extended functionality opens up new application areas such as automotive AI, tactile internet, industry 4.0 and biomedical processing.

BrainChip Akida

The Akida [139] is a digital neuromorphic processor by the BrainChip company. The Akida is an edge neuromorphic processor designed for inference and continuous learning from streamed data with SNNs. Akida uses a homeostatic form of STDP for continuous learning from data [139]. Brainchip's Akida has been used in applications such as chemosensing [177], and comparing ANNs and SNNs for ambient assisted living [128].

Intel Loihi

The Intel Loihi is a neuromorphic processor that integrates a wide range of features such as hierarchical connectivity, dendritic compartments, synaptic delays, and programmable synaptic learning rules [32]. Each Loihi chip is composed of a many core mesh compris-

ing of 128 neuromorphic cores with each core implementing 1024 primitive spiking neural units, three embedded x86 processor cores, with an asynchronous network-on-chip (NoC) for between core communication. Loihi offers a variety of local information for programmable synaptic learning processes such as spike traces with configurable time constants that can have different time constants. The Loihi can deploy SNNs that use the CUBA LIF neuron dynamics described in Equation 2.1. Synaptic weights between neurons can be updated via a learning rule expressed as a finite-difference equation with respect to a synaptic state variable that follows a sum-of-products form as follows [32]:

$$W_{ij}[t+1] = W_{ij}[t] + \sum_k C_k \prod_l F_{kl}[t], \tag{2.7}$$

where W_{ij} is the synaptic weight variable defined for the destination-source neuron pair being updated; C_k is a scaling constant; and $F_{kl}[t]$ may be programmed to represent various state variables, including pre-synaptic spikes or traces, post-synaptic spikes or traces, where traces are represented as first-order linear filters. The weights are stochastically rounded according to the programmed weight precision. Traces are stochastically rounded to 7-bits of precision.

The work of this thesis primarily focuses on the first generation of the Loihi processor but Intel has recently released a second generation of the Loihi to the research community. Intel's Loihi 2 has an increased number of neurons per chip, faster asynchronous processing, and supports a wide range of stateful spiking neuron models with fully programmable dynamics [131]. While the work in this thesis was done using Intel's Loihi 1 chip, it is possible to implement the algorithms presented in the Loihi 2 and beyond as well as other digital neuromorphic processors. In each of Chapters 3, 4, and 5 SNNs will be applied in the Intel Loihi to demonstrate low-power online edge learning in neuromorphic hardware.

Chapter 3

Online Few-shot Learning with Neuromorphic Processors

The most recent advancements in Artificial Neural Networks (ANNs) have propelled them to achieve state-of-the-art results across a diverse range of applications. These include image classification, object recognition, object tracking, signal processing, natural language processing, self-driving cars, healthcare diagnostics, and various other fields [50]. ANNs continue to push the boundaries of what is possible in these areas, opening up new possibilities for innovation and advancement with recent innovations such as transformer based models [17]. ANNs primarily leverage backpropagation of errors as the cornerstone of their learning capabilities, but effective training demands substantial amounts of data and memory. The training process involves GPUs and entails thousands of iterations over the data, sampled in an i.i.d. fashion. However, to achieve the required high throughput for network training, GPUs rely heavily on data movement and tensor-based computations, both of which consume significant energy [120, 77]. This renders GPUs less than ideal for large-scale implementation in mobile systems, where energy efficiency is of paramount importance.

Neuromorphic computing platforms present a compelling and energy-efficient alternative for conducting training and inference in neural networks, especially in power-constrained mobile systems [64]. These systems emulate the brain's event-driven dynamics, distributed architecture, and massive parallelism, effectively addressing the limitations of traditional von Neumann computing architectures [69]. With synaptic plasticity capability, neuromorphic hardware can perform training and inference online, relying on local information [25, 32]. This feature makes them particularly appealing for tasks that necessitate rapid adaptation to new data. Despite significant technical advancements in neuromorphic learning hardware, the practical role of learning and synaptic plasticity in such systems remains challenging. Gradient-based learning, in particular, is notoriously slow, demanding numerous iterations to achieve satisfactory generalization. Moreover, synaptic plasticity operates in an online and local manner, which breaks the i.i.d. assumptions required for neural network convergence when attempting to learn online with streaming data.

We adopt a pragmatic and realistic approach to learning in neuromorphic hardware, harnessing the strengths of both conventional and neuromorphic hardware. Specifically, we employ pre-training of networks on GPUs for a particular set of tasks and subsequently fine-tune the final layer on neuromorphic hardware [162]. By doing so, we effectively offload the non-local and energy-intensive phases of learning to a cloud or mainframe, while deploying the data-sensitive aspects of learning on neuromorphic hardware. To achieve this, we leverage recent theories that integrate machine learning principles with SNNs and few-shot/transfer learning. Our approach is successfully demonstrated through fast online learning of real-world visual patterns from streaming data on a neuromorphic processor. For pre-training, we utilize a functional model of an Intel Loihi neuromorphic processor, and during deployment, the model is fine-tuned on the processor itself using local synaptic plasticity rules. The primary contribution of this work lies in the development of few-shot Surrogate Gradient Online Error-triggered Learning (SOEL) [162], a plasticity rule compatible with neuromorphic hardware. We derive this rule from gradient-descent on SNNs and efficiently implement

it using signals local to the neuromorphic cores. Rather than backpropagating through the network, SOEL computes local errors between pre- and post-synaptic neurons by propagating gradients forward in time [125]. Overall, our novel approach bridges the gap between conventional and neuromorphic hardware, paving the way for enhanced learning capabilities while capitalizing on the unique advantages offered by neuromorphic systems.

Temporal continuity is an essential property in the brain, but continuous weight updates can be both energetically costly and susceptible to dynamical instabilities, particularly when updating a large number of weights. A recent advancement in Surrogate Gradient (SG) learning for spiking neurons proposes an alternative approach, indicating that continuous updates at every timestep might not be necessary. Instead, weight updates are triggered solely by task errors, leading to error-triggered learning [135]. In error-triggered learning, weight updates occur only when the error surpasses a threshold. This reduction in the number of updates significantly decreases the computational burden while incurring only a small penalty on the final accuracy.

Despite this improvement, error-triggered learning still faces catastrophic forgetting in the context of online learning. Catastrophic forgetting occurs when the data-generating process for training the neural network is non i.i.d., which is the case in online learning where streamed data is correlated. To overcome this challenge, various approaches can be employed, such as increasing the complexity of neurons and synapses [200, 81], utilizing experience replays [116], or adopting meta-learning and related few-shot learning techniques [5]. Here we take a few-shot learning approach to overcome the challenge. Our approach to few-shot learning is two-fold: first we pre-train a model on the class of problems of interest, then we make (presumably) few error-triggered updates to enable the model to learn new but related tasks quickly and effectively.

To achieve error-triggered learning on neuromorphic hardware, we implemented the SOEL algorithm, an extension of the capabilities of SG and error-triggered learning, enabling rapid

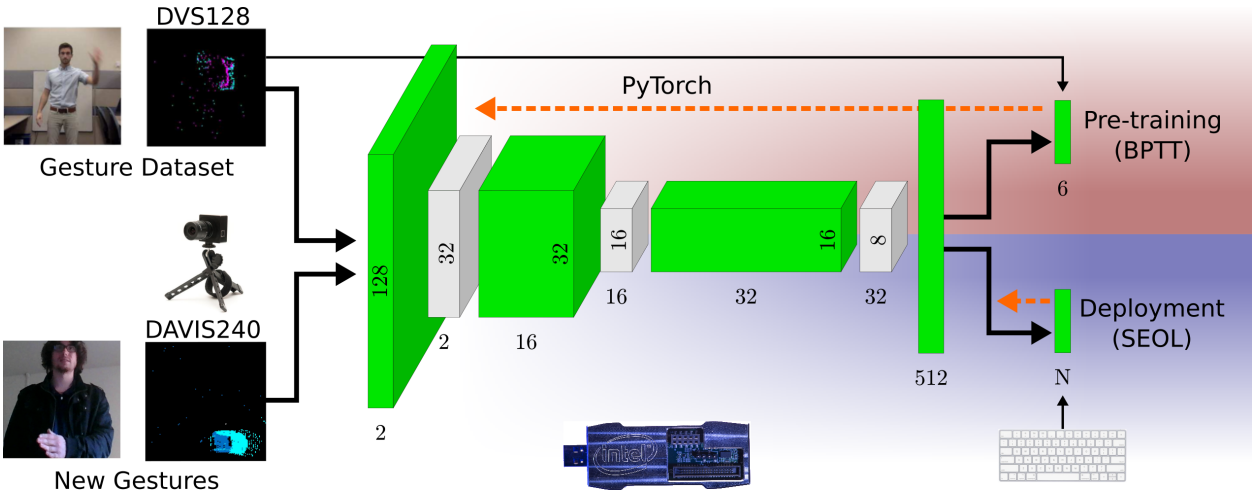


Figure 3.1: Experimental setup. During a pre-training phase, the Loihi compatible convolutional network is trained on a computer using an event-based gestures dataset, the functional simulator, and SLAYER/BPTT. In this work, the pre-training dataset consisted of the IBM DVS Gestures dataset recorded using a DVS128 camera. The entire network along with quantized parameters of the functional simulation are then transferred onto the Loihi cores. During deployment, new gestures recorded using a DAVIS are streamed to an Intel Kapoho Bay. Few-shot learning is performed on the final layer using on-chip SOEL. The deployed network, including inference and training dynamics are performed on the Loihi chips. Dashed orange arrows indicate the extent of the spatial credit assignment, and thus which layers are trained in each of the two phases. Figure adapted from [162]
.

and efficient few-shot learning [162]. Our demonstration of SOEL has been carried out on the Intel Loihi Neuromorphic research chip, leveraging its specialized local plasticity processors to execute weight updates. SOEL represents a significant advancement over a previous implementation [161], effectively addressing and resolving issues to enhance the overall performance of the learning process. By pushing the boundaries of error-triggered learning and leveraging the capabilities of neuromorphic hardware, our work opens up new possibilities for rapid adaptation and few-shot learning in intelligent systems. These advancements have promising implications for real-world applications, where swift adaptation to novel tasks is of utmost importance.

3.1 SNN Model

The primary aim of few-shot learning is to equip the model with the ability to generalize effectively from minimal examples. By incorporating transfer learning into the training of few-shot learning models, both in traditional Artificial Neural Networks (ANNs) and deep Spiking Neural Networks (SNNs), we achieve greater generalization on the target domain using only a few examples [141, 153, 162]. While previous work has shown increasing success in training SNNs using spike-based gradient descent on a variety of tasks, they are trained and tested offline just like ANNs and do not demonstrate online on-chip learning on neuromorphic hardware. [65] demonstrated rapid online on-chip learning using the Intel Loihi neuromorphic research chip but did not use spike-based gradient descent. To our knowledge this work is the first to demonstrate online, on-chip gradient-based learning on a neuromorphic processor. Using mid-air gestures as a case study we show the success of rapid online learning using *SOEL* which can be used for applications that require online adaptation. The neural network model neurons use the CUBA LIF dynamics described in Equation 2.1 in Section 2.1.4.

3.2 Gradient-based Training of SNNs

A number of recent methods for training SNN using gradient descent have recently emerged. The mathematical principle of gradient descent lies on incrementally updating the parameters in the direction opposite to the gradient of a loss function. As mentioned in the previous chapter, difficulties of training SNNs arise due to the spatio-temporal credit assignment problem and the non-differentiability of the activation function. The spatial credit assignment problem arises when the parameters of neurons with no direct target are trained. The temporal credit assignment arises organically due to the temporal dependencies (dynamics) of spiking neurons. For example, the effect of an input spike at a particular time affects the

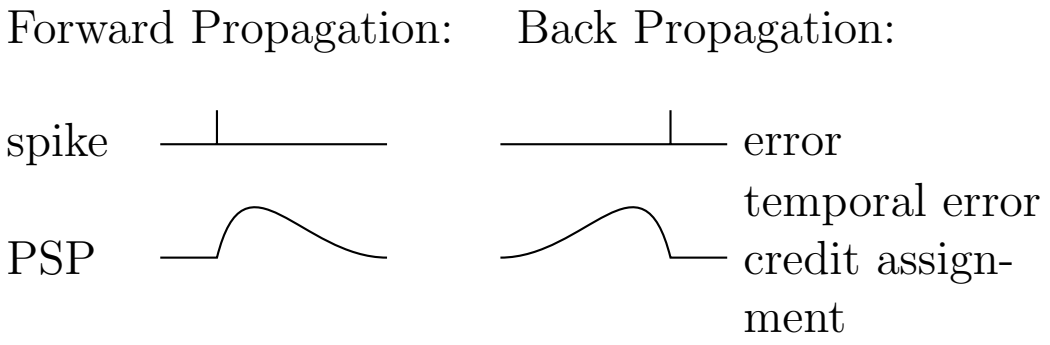


Figure 3.2: Temporal credit assignment of an error at a point in time during SLAYER backpropagation.

membrane potential of a spiking neuron in the future. The magnitude of the effect is determined by the *normalized post-synaptic response* (PSP) of the synapse. The PSP of a synapse is a weighted exponential decay trace of the activity of synapses. Because v and u of the CUBA LIF model are linear with respect to the network layer weights W_{ij}, the weight projecting from pre-synaptic neuron i to post-synaptic neuron j, the dynamics of v_j can be rewritten as:

$$\begin{aligned} v_j(t) &= \sum_{i \in \text{pre}} W_{ij} P_i(t), \\ \tau_{mem} \frac{\mathrm{d}}{\mathrm{d}t} P_i(t) &= -P_i(t) + Q_i(t), \\ \tau_{syn} \frac{\mathrm{d}}{\mathrm{d}t} Q_i(t) &= -Q_i(t) + S_i(t). \end{aligned} \tag{3.1}$$

The states P and Q describe the traces of the membrane and the current-based synapse, respectively. For each incoming spike, each trace undergoes a jump of height 1 and otherwise decays exponentially with a time constant τ_{mem} (for P) and τ_{syn} (for Q). Weighting the trace P_i with the synaptic weight W_{ij} results in the PSPs of neuron j caused by input neuron i. As a consequence, during learning, the credit of the error at a given point of time must be assigned to the input synapse at some point in the past. One factor to the magnitude of this temporal credit assignment is proportional to the normalized post-synaptic response, reversed in time. An illustration of this temporal credit assignment policy is shown in Figure 3.2.

Here we reiterate the surrogate gradient factors for global cost function $\mathcal{L}(S^N)$ defined on the spikes S^N of the top layer and targets Y described by equation 2.2 in section 2.3.1.

$$\nabla_{W_{ij}}\mathcal{L}(S^N) = \frac{\partial \mathcal{L}(S^N)}{\partial S_j}\frac{\partial S_j}{\partial v_j}\frac{\partial v_j}{\partial W_{ij}}. \tag{3.2}$$

We discuss the three factors in the context of few-shot learning for classification problems. For didactic reasons, we proceed first with the second term, then the first term, then the third. The second term is the derivative of the activation function of the spiking neuron which is non-differentiable. As discussed earlier, the SG approach consists in using a smooth surrogate function in place of the non-differentiable step function, such as the boxcar function [123, 74]. The first term on the right-hand side describes how the loss changes as the spiking states in the network, S_i, change. If the loss function is the mean-squared error and the network consists of only one layer, the first term becomes the task error $(Y_j - S_j^N)$. Computing $\frac{\partial \mathcal{L}(S^N)}{\partial S_i}$ for hidden layers is non-trivial and equivalent to solving a spatiotemporal credit assignment problem. Two methods exist to solve this problem: (1) it can be computed *offline* using gradient backpropagation on the time-unfolded graph (*i.e.,* BPTT), or *online* by using local loss functions [74]. The third factor significantly enhances learning by effectively projecting task-specific errors to the neurons [9] allowing for the use of an error driven loss function with gradients in a similar manner to ANNs. Therefore the third factor bridges the worlds of deep Artificial Neural Networks (ANNs) and deep Spiking Neural Networks (SNNs) without resorting to oversimplified assumptions about the latter. This work uses a combination of offline and online SNN learning, namely SLAYER and SOEL for pre-training hidden layers and online three-factor rules for learning in output layers, respectively. In the following sections, we provide further detail about these two learning methods.

3.3 SLAYER Offline Training for Loihi

SLAYER is a gradient computation method for training deep SNNs directly in the spiking domain [154]. It treats the inputs and outputs of the SNN as temporal signals and backpropagates the error at the output layer accordingly. There are two basic guiding principles in SLAYER: Temporal error credit assignment, and the surrogate gradient. Temporal credit assignment is done by unfolding the temporal dynamics in time and backpropagating through the unfolded graph. Further, it is typical for the normalized post-synaptic response to decay to practically zero after some time. Therefore, it is sufficient in practice to apply temporal error credit assignment only up to a finite point in history. SLAYER uses a proxy function as an approximation of spike function derivative, similar to the surrogate gradient learning described in Section 3.2. These principles form the essential link in the computational graph used to calculate the gradients of the weights of the SNN and train it using standard deep learning optimization methods.

SLAYER PyTorch[1] also supports training an SNN with the CUBA leaky integrate and fire neuron model compatible with the Loihi chip. For one-to-one mapping of the trained network in Loihi hardware, the SNN is modeled with a functional Loihi simulator and the normalized post-synaptic response is used for temporal error credit assignment during backpropagation. In addition, since Loihi only supports integer weights, a strategy of full precision shadow weights [61, 26] is used, which are quantized during the forward inference phase only. The computational blocks for SLAYER-Loihi training are shown in Figure 3.3.

]

[1] `https://github.com/bamsumit/slayerPytorch`

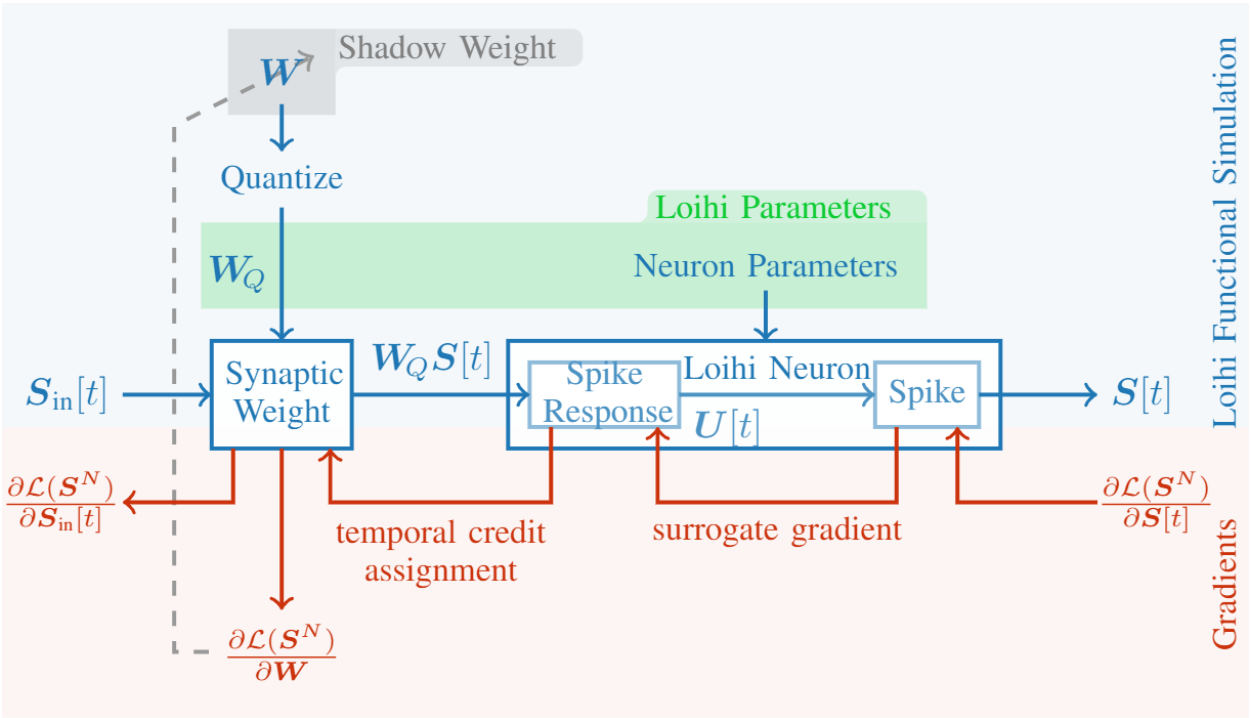


Figure 3.3: Computational blocks for offline pre-training of SNN for Loihi using SLAYER. The SNN is modeled with a functional Loihi simulator and the normalized post-synaptic response is used for temporal credit assignment. Since Loihi uses integer weights full precision shadow weights are quantized and used during the forward inference phase.

3.4 SOEL Online training with Loihi

Online training on physical substrate requires all the information necessary for computing the gradient to be available at the synaptic plasticity processor. The first two terms of equation 2.2 discussed in section 2.3.1 are errors and postsynaptic states. The last term in equation 2.2 can be computed from the neuron dynamics such as the CUBA LIF used in SLAYER and Loihi described in section 2.1.4. Therefore, we are left with the following three factors:

$$\nabla_{W_{ij}} \mathcal{L}(S) = -(Y_j - S_j)\sigma'(v_j)P_i. \tag{3.3}$$

Provided that pre-synaptic traces P_i, membrane potential v_j and errors are available at the synapse, learning can be performed locally as a synaptic plasticity rule. In computational neuroscience, rules of this type are referred to as three-factor rules [49]. Three-factor rules are consistent with biological synapse dynamics and constitute a normative theory of learning in the brain.

Equation 3.3 prescribes updates at every timestep. While this is consistent with biological dynamics, it is not efficient in hardware. Updates can instead be made when the error $(Y_j - S_j^N)$ crosses a threshold, thus forming a binary "error event" [135]. This is reminiscent of STDP, where updates are triggered when pre-synaptic or post-synaptic neurons events occur [12]. Here, updates are instead triggered by error events. Error-triggered learning allows the conditional activation of the plasticity operations, which can drastically reduce the footprint of online learning. Recent work showed that the number of updates can be reduced by 20 fold for a small loss in accuracy [135]. Using a piecewise SG function, $B_j = \sigma'(v_j)$ becomes a box function where $B_j \in \{0, 1\}$. Then, the SOEL rule can be written in the following compact, three-factor form:

$$\nabla_{W_{ij}} \mathcal{L}(S^N) \propto -E_j B_j P_i. \tag{3.4}$$

where E_j is a integer error event for neuron j.

Intuitively, SOEL can be interpreted as follows. If err is higher than θ, meaning the neuron is spiking at too high a frequency, then there is a positive error and the weight of the synapse will be penalized. Conversely if err is below $-\theta$ then the weight of the synapse weight will be increased. The term σ' (the "second factor" in equation 3.3) cannot be implemented directly on the Loihi because the membrane state is not available at the plasticity processor. Since only the final layer N is trained, setting this term to 1 regardless of the membrane value only has the effect of continued learning even after the neuron output saturates in either direction. This strategy, referred to straight-through estimator in the machine learning field, has the disadvantage of yielding biased estimated of the gradients, but the advantage of faster learning since every error leads to an update. Since our goal is to perform fast, one-shot learning, SOEL implemented here uses a straight-through estimator. The full learning rule can then be expressed as:

$$W_{ij} = W_{ij} + \eta(E_j - C)P, \tag{3.5}$$

where W_{ij} is the synapse from pre-synaptic neuron i to post-synaptic neuron j, η is the learning rate, E_j is the error, and P_i is the pre-synaptic trace. The learning rule can be implemented in Intel Loihi as:

$$\begin{aligned} X_i^1[t+1] &= \alpha^1 X_i^1[t] + S_i^1, \\ X_i^2[t+1] &= \alpha^2 X_i^2[t] + S_i^2, \\ Y_j[t] &= E_j[t], \\ \Delta W_{ij} &= \eta(X_i^2[t] - X_i^1[t])(Y_j[t] - C). \end{aligned} \tag{3.6}$$

Here, X^2 and X^1 are pre-synaptic trace variables available in the Loihi whose subtraction in the third equation yields the second order kernel equivalent to P_i in equation 3.1.

$$P_i[t] \propto (X_i^2[t] - X_i^1[t]). \tag{3.7}$$

A Loihi Lakemont core computes the spike count $\bar{S}$ and evaluates err_j at regular intervals T. If the error exceeds the threshold θ, the post-synaptic trace value in the plasticity processor, Y, is written with the error E_j and a plasticity operation is initiated. As in equation 3.9, C is a constant bias term to account for negative error because traces cannot be negative.

Table 3.1: Network architecture.

Layer	Kernel	Output	Training Method
input		128×128×2	DVS128/DAVIS240C (Sensor)
1	4a	32×32×2	SLAYER (BPTT)
2	16c5z	32×32×16	
3	2a	16×16×16	
4	32c3z	16×16×32	
5	2a	8×8×32	
6	-	512	
output	-	N	SOEL

Notation: `Ya` represents `~YxY~` sum pooling, $XcYz$ represents X convolution filters (`YxY`) with zero padding. N is the number of classes, which is task-dependent.

3.5 Online few-shot learning using SOEL plasticity for Loihi

SOEL requires the pre-synaptic trace P to be a second-order linear filter. Second-order kernels can be implemented as a subtraction of the two first-order kernels [48]. This subtraction is enabled by the sum-of-products formulation of the plasticity rule (Equation 2.7). The error, err_j, is computed with the post-synaptic neuron using the following:

$$err_j[t] = Y - \bar{S}_j[t] \tag{3.8}$$

where Y is the target, $\bar{S}_j = \sum_{t-T}^{T} S_j[t]$ is defined here as the number of post-synaptic spikes by neuron j in the previous T timesteps. T is a constant number of timesteps that is a fraction of the total presentation time of the sample. This number determines the rate at which errors are computed.

Using a spike-count instead of spike states is an approximation because the update will be subsequently made using the states in the final timestep, *i.e.,* $P_i[t]$. However, for T smaller than the neuron and synaptic time constants, $P_i[t]$ will not vary much during this time window, and the approximation will remain close to the exact case. Since the post-synaptic trace is not necessary for the SG rule, SOEL writes the error on the same register used for the post-synaptic trace. This enables the error value to be available in the plasticity processor for learning. On the chip, post-traces can only be positive but errors can be both positive and negative. This problem is solved by offsetting the weight updates with a constant term C.

$$E_j[t] = \begin{cases} C + err_j[t], \text{ if } err_j > \theta \text{ or } < -\theta \\ C, \text{otherwise} \end{cases} \tag{3.9}$$

where θ is an error threshold.

We trained a spiking CNN using SLAYER for Loihi (figure 3.1) on the DVS Gesture dataset [4]. It has eleven output gestures, out of which six (the even classes) were used for offline training using SLAYER. The input is a 128×128 spatial event with two polarities (ON and OFF). The spiking CNN architecture shown in table 3.1 consisted of 7 layers. The input spikes are OR'ed in 1 ms time bins and then fed to the network. N refers to the number of output classes, which depend on the experimental conditions.

The threshold for all the neurons were set to 80×2^6 and the current decay and voltage decay were set to 1024 (time constant of 32ms) and 128 (time constant of 4ms) respectively. The weights of the network were trained to be in the set $\{-256, -254, \cdots, 254\}$ i.e., 8 bit signed weights with step of 2.

The network was pre-trained for 2000 epochs. For better generalization performance, the input was augmented during training: x-y jitter of up to 8 pixels, rotation jitter of up to 10°,

and random sampling of 1450ms spike sequence. The Nadam [37] optimizer was used with a learning rate of 0.003 and default $\beta = (0.9, 0.999)$. The network without the final fully connected output layer (layers 1–6), is the feature extraction network which is subsequently used in our on-chip learning experiments described below.

3.6 Results on the Loihi

We present a system for online learning of gestures from DVS data using only a few shots. Our workflow consist of a pre-training phase, followed by a deployment phase. The pre-training phase uses SLAYER and its functional Loihi simulator to train a Loihi compatible convolutional network on a GPU. For our targeted human gesture recognition application, we use the event-based IBM DVS Gestures dataset to pre-train this network. The trained network and quantized parameters are then transferred to the Loihi cores. During deployment on Loihi, few-shot learning of new gestures is performed on the top layer on-chip with SOEL. The algorithm implementation of SOEL on Loihi is described in algorithm 1. The system is shown in figure 3.1 and its components are detailed in the following subsections.

The datasets used in this work are obtained using neuromorphic sensors, namely the DVS and DAVIS cameras. In our experiments we use data from a DVS 128, and a DAVIS 240C [15]. The IBM DvsGesture dataset used here for pre-training consists of recordings of 29 different individuals performing 10 different actions such as clapping and an unspecified gesture for a total of 11 classes. The actions are recorded using a DVS camera, an event-based neuromorphic sensor, under three different lighting conditions. The task is to classify an action sequence video. Samples from the first 23 subjects were used for training and the last 6 subjects were used for testing. The training set contains 1078 samples and the test set contains 264 samples. Each sample consists of the first 1.45 seconds of the gesture performed. Visual input to the model was recorded with either a DVS 128 in the case of the

IBM DVSGesture[2] dataset [4] or with a DAVIS 240C [15] in the case of real-world gestures. The network was pre-trained data recorded with a DVS 128, which has a smaller resolution compared to the more recent DAVIS 240C. During experiments involving input live-streamed from a DAVIS 240C to an Intel Kapoho Bay, data was scaled down to the same dimensions as the DVS 128 before being input into the network. Data was taken by one subject under three different lighting conditions, natural light from the sun, incandescent light, and fluorescent light which is shown in figure 3.5.

We used SOEL to train and test the last layer of the neural network pre-trained with SLAYER on 6 of the 11 gestures from the DVSGesture dataset, training the last layer with only a few-shots, ranging from 1 to 20, of the remaining 5 gestures of the dataset for a few-shot 6+5 way gesture classification task. The 6 refers to the 6 gestures the network is pre-trained to classify using SLAYER, and the 5 refers to the 5 new gestures we are training the last layer of models on using only a few-shots. "Train" refers to classification accuracy on training samples using saved weights with plasticity disabled. "Test" is the classification accuracy on the samples held out of the training procedure. Models were trained on one, five, or twenty shots of data and then tested on 100 held out samples. The results are obtained from performing a 5-fold cross-validation. A gesture is considered correctly classified if the desired neurons spike frequency is highest during the presentation time. We compare the accuracy of the model using SOEL to two other models, one whose last layer is trained using vanilla SGD used in [161] and another whose last layer is trained using SLAYER. Each model was trained on samples from the DVSGesture test dataset, and tested on samples from the test dataset not seen during training. Table 3.2 shows the accuracy comparisons of the different models trained on the few-shot 6+5-way gesture classification task. The results show the SOEL trained network is overall better than the vanilla SGD method from [161], achieving on average significantly better results at test time after seeing only one shot of training data, and better generalization. However while SOEL does better at training time

[2]The DVS Gesture dataset is used under a Creative Commons Attribution 4.0 license.

on 1 shot experiments than the pure SLAYER network, SLAYER is better at generalizing than SOEL. This could be due to the SOEL model tending to over-fit on samples presented and the straight-through estimator. For similar reasons, we speculate that the accuracy of the SOEL model is more variable than SLAYER. When overfitting, samples that deviate too much from those samples will be more likely to be classified incorrectly, but all experiments show SOEL to be significantly better than our previous implementation [161]. We also compare the time taken and energy consumption needed for SOEL and [161] to train each gesture shown in table 3.3. The results indicate that SOEL uses more energy but takes less time to train and achieves higher accuracy than [161] using normal surrogate gradient descent. Because the Intel Kapoho Bay does not support energy probing, energy and time measurements were taken with an Intel Nahuku board consisting of 32 Loihi chips.

Algorithm 1: SOEL

Result: Error-triggered Synaptic Plasticity
$\theta = 0$;
if *neuron i is learning* **then**
  $\bar{S}_j \leftarrow \bar{S}_j + S_j$;
  $err_j \leftarrow Y - \bar{S}_j$;
  if $err_j > \theta$ *or* $err_j < -\theta$ **then**
    $E_j \leftarrow C + err_j$
    $W_{ij} \leftarrow W_{ij} - \eta(E_j - C)P_i$;
    increase θ by constant;
    $\bar{S}_j \leftarrow 0$;
  else
    decrease θ by constant;
  end
end

3.7 Real-World Gesture Learning

In addition to the few-shot 6+5 way classification we also tested SOEL in a real-world gesture learning and recognition setting. To demonstrate rapid online gesture learning in a real-world setting we streamed gesture data in real time from a DAVIS 240C sensor connected to an

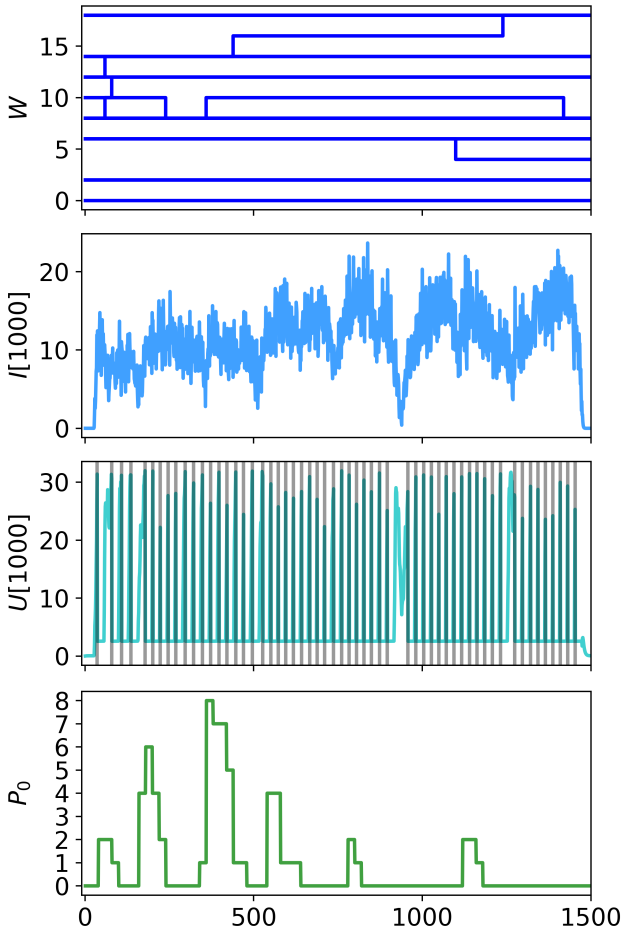


Figure 3.4: Dynamics of one learning neuron when learning a new gesture. Only a subset of the synaptic weights W are shown. The weights only change when P_0 is non-zero during a learning epoch. The current I, and membrane potential U of the learning neuron are shown over the duration of the sample. Spikes are shown as grey vertical lines overlaying the membrane potential plot.

Table 3.2: 6+5-way few-shot classification on the DvsGesture dataset

Dataset	Learning Method	Shots	Train	Test
DVSGesture	SOEL [162]	**1**	**96±4%**	**64.7±4.6%**
		5	**88±5.2%**	**65.1±5.1%**
		20	**87.7±2.3%**	**80.2±4.3%**
	SGD [161]	1	40%	40%
		5	60%	43.3%
		20	73.5%	56.2%
	SLAYER [154]	1	78.5±2.6%	83.5±2.32%
		5	95.9±1%	83.5±2.9%
		20	99.7±.3%	91.2±1.9%

Table 3.3: Comparison of time and energy taken for learning one gesture

Measurement	SOEL [162]	SGD [161]	% Diff
Learning Time (s)	.037	.31	-87.71%
Learning Energy (mJ)	167.58	37.04	358.46%
Learning Power (mW)	6.18	11.48	-46.17%
Total Time (s)	1.04	1.21	-14.23%
Total Energy (mJ)	481.98	323.2	49.13%
Total Power (mW)	511.65	391.07	30.83%

Intel Kapoho Bay. For the experiment we tested one subject in a single lighting condition where the subject was under fluorescent light. The neural network model on the Kapoho Bay was pre-trained on all 11 gestures of the DVSGesture dataset using SLAYER, but the last layer is reset, made plastic, and trained using SOEL. The task was to train the network to classify 10 predetermined gestures outside of the DVSGesture dataset using as few shots as possible. Figure 3.5 shows an example of the learning and inference of the gestures. After being shown a gesture for a one second presentation, the network is able to classify other samples of the gesture. Additionally, training other gestures does not interfere with the networks ability to classify previously learned gestures. However, performance can degrade if the learned gestures spatially overlap because unique gestures within the same space may be seen as the same gesture.

The results of which some are shown in figure 3.5 demonstrate the capability of the SOEL learning rule to perform rapid few-shot learning on a neuromorphic processor from real-world data. A link to a video showing a live demonstration of the rapid learning of 10 new gestures is here https://drive.google.com/file/d/1JR161oJxEGS_Rrxvz3mvazZOj2U1qBhV/view?usp=sharing.

3.8 Discussion

We presented SOEL, a new surrogate gradient based learning algorithm for few-shot online learning on an Intel Loihi neuromorphic processor using gesture recognition as a case study.

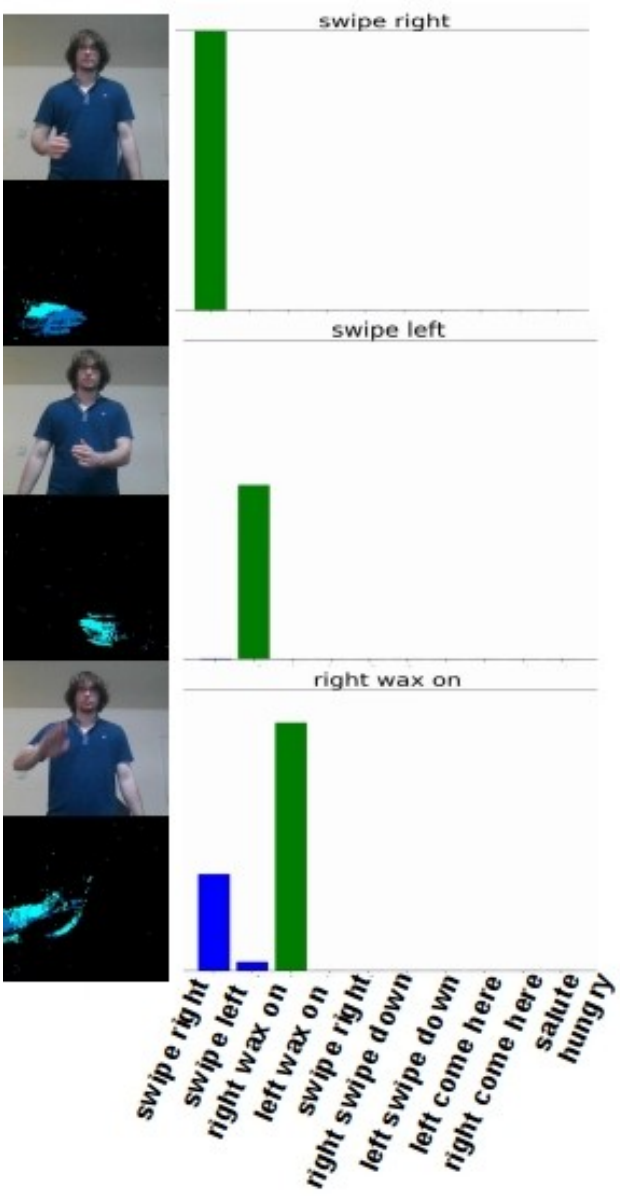


Figure 3.5: Rapid online learning of gestures using data streamed from a DAVIS240C to an Intel Kapoho Bay. The upper part of the figure shows a person performing a gesture in front of a DAVIS240C, and the corresponding DAVIS240C output events shown in blue. The histogram shows the spiking frequency of each neurons response to the presented gesture after learning. After only a single one second presentation of each gesture the network can correctly classify the gestures it trained on.

To accomplish this we first pre-trained an Intel Loihi compatible SNN on a GPU using the current state-of-the-art SLAYER method, and then deployed the network on an Intel Kapoho Bay and retrained the last layer on few-shots of data using SOEL. We found that

like with ANNs, using a pre-trained network for transfer learning with SNNs significantly boosts few-shot learning accuracy. While we have achieved real-time online gesture learning using SOEL, there are limitations to our method. Currently, SOEL only supports training the last layer of the network. Being a local learning rule, SOEL only has information from pre- and post-synaptic neurons within its layer. Therefore training other layers will incur the spatial credit assignment because the neurons will not have a direct target to train on outside of the last layer. Consequently, if the signal is not separable in the penultimate layer then the last layer cannot learn. This can be potentially solved using layer wise local loss functions [74] and is beyond the scope of this article. Another limitation stems from the approximations made with the SOEL algorithm. First, the algorithm assumes that states do not change across the time window in which the error is calculated. This is beneficial to speed up training and can be adjusted to match the error dynamics. Second, due to limitations of the plasticity processor, the second term of the three factor rule cannot be implemented exactly and is instead ignored (set to one). These two approximations are likely to reduce the accuracy of the final result. The few-shot learning experiments using SOEL on gesture data with the Intel Loihi neuromorphic processor are slightly worse compared to training the last layer using GPU SLAYER. This discrepancy is expected since SOEL yields biased estimates of the gradients. The bias in the estimates is caused mainly by the straight-through estimator, and the approximate spike count loss which is computing using the neural states of the last time step. Furthermore, the discretization of neural and synaptic states, and limited range of effective learning rates further widen the gap between GPU simulations and Loihi simulations. However, in the regime of interest, *e.g.* between one shot and five shots, the discrepancy remains acceptable. Furthermore, they are a major improvement from our previous work. Unlike vanilla SGD, which learns at every timestep, SOEL only learns when there is sufficient error to trigger learning. This error-triggered learning helps prevent weight saturation and catastrophic forgetting leading to increased accuracy. However the increased accuracy of SOEL comes with an increase in power consumption when compared to vanilla

SGD of the previous implementation from [161]. We speculate that the power consumption for gesture recognition using SLAYER with a GPU is at least an order of magnitude higher than using SOEL with the Intel Loihi. Additionally we also showed SOEL is capable of few-shot learning from real world data. These experiments also showed SOEL was able to adapt to the differences of data taken from both a DAVIS 240 and a DVS 128 and was able to learn using data from both.

While few-shot online learning with SOEL was successful, the final accuracies on the gesture data the model was tested on were low for one or a handful of shots which is not practical for real-world deployment. In the next chapter we explore how this problem was overcome through the deployment of bi-level learning and meta-learning.

Chapter 4

Meta-Learning with Surrogate Gradients for Neuromorphic Hardware

4.1 Introduction

Rapid adaptation to unfamiliar and ambiguous tasks are hallmarks of cognitive function and long-standing goals of Artificial Intelligence (AI). Neuromorphic electronic systems inspired by the brain's dynamics and architecture strive to capture its key properties to enable low-power, versatile, and fast information processing [111, 69, 31]. Several recent neuromorphic systems are now equipped with on-chip local synaptic plasticity dynamics [25, 137, 32]. Such neuromorphic learning machines hold promise to building fast and power-efficient life-learning machines [124].

Here, we demonstrate that gradient-based meta-learning on SNNs can solve these problems in practical cases with technological interest, and are particularly well suited to the

constraints of neuromorphic hardware and online learning (Fig. 4.1). To do so we combine Model Agnostic Meta Learning (MAML), a second-order gradient-based method that optimizes the network hyperparameters, and the surrogate gradient method [125]. Two ingredients were key to the results of our work. First, surrogate functions used to estimate SNN gradients can be made twice differentiable, hence are suitable for second-order learning as in MAML. Second, the definition of suitable event-based datasets to demonstrate meta-learning on SNNs. While MAML had been previously applied to SNNs, prior work focused on meta-training Hebbian STDP dynamics on non-event-based datasets which do not take any advantage of the event-based nature of SNNs [188]. Furthermore, surrogate gradient learning implementing stochastic gradient descent can be implemented as a form of three-factor learning [49, 197, 74, 11] that vastly exceeds the performance of classical STDP, while being compatible with neuromorphic hardware implementations [135, 27]. As our results suggest, the meta-training of SNNs using the surrogate gradient method can be used as a tool to adapt and tune synaptic plasticity circuits.

We study the SNN MAML approach in the context of few-shot learning, whereby a model is trained on a set of labeled tasks drawn from a given *domain* of tasks to adapt to unseen ones of the same domain using a small number of samples and iterations. Examples of few-shot learning are learning novel hand or body gestures, agents learning to take new goal-driven actions in a new maze, or optimizing automatic speech recognition to the individual pronunciation of the subject.

One important obstacle to meta-learning research in neuromorphic engineering is the lack of suitable datasets. Neuromorphic hardware implementing SNNs is most suitable to processing event-based datasets and loses most of its salient features when applied to static data [31]. The Omniglot [87] and MiniImagenet [179] datasets have been pivotal in pushing the field of meta-learning ahead. However, there exists no event-based dataset that is comparable to Omniglot or MiniImagenet where modeling dynamics are crucial to solving the problem.

The recently published N-Omniglot dataset partially fills this gap [97]. Taking inspiration from existing meta-learning benchmarks that fuse multiple datasets [168], we define new benchmarks that consist of combinations of event-based datasets recorded using neuromorphic vision sensors. We demonstrate performances that are comparable to the performance of conventional neural networks trained on these datasets. We note here that the goal of this work is not to exceed the performance of conventional ANNs, but to demonstrate that MAML can successfully be used in the regime corresponding to gradient-descent of SNN dynamics, and thus to fine-tune synaptic plasticity circuits. We demonstrate the latter by training an approximate gradient-based synaptic plasticity rule previously employed in neuromorphic hardware.

Finally, we analyze the updated statistics, which reveal that the MAML not only results in fast learning but does so using large weight updates. These results have promise for online learning using low-precision weight memory, and are reinforced by meta-learning using differentiable quantization techniques.

Specific Contributions

This work provides 1) a method of parameter initialization that enables neuromorphic hardware to few-shot learn new tasks; 2) a method to construct meta-datasets using data taken from the DVS neuromorphic sensor, with two examples made publicly available: Double NMNIST, Double ASL-DVS, N-Omniglot; and 3) the effectiveness of second-order meta-training of SNNs synaptic plasticity rules. This work is thus a stepping stone towards implementing MAML with SNNs in neuromorphic hardware for fast adaptation to streamed event-based sensor data.

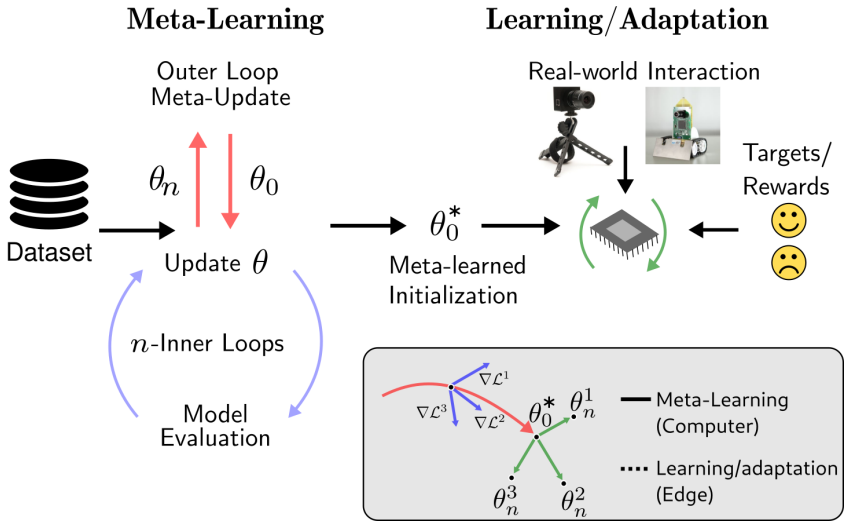


Figure 4.1: Meta-Learning for SNNs using Surrogate Gradients. In the first phase, an SNN or a functional simulator of a neuromorphic hardware's SNNs is meta-trained using surrogate gradient methods on a class of tasks T_i stored on a computer. The goal of meta-training is to learn an initial parameter set θ_0^* such that out-of-sample tasks (*e.g.* θ_n^1, θ_n^2, θ_n^3, ...θ_n^*) can be learned quickly. In the envisioned application, θ_0^* would be learned offline on a conventional computer and learning/adaption would take place at the edge, using neuromorphic sensing and processing.

4.2 Methods

4.2.1 Model Agnostic Meta-Learning

For training and evaluating our SNNs we use the MAML method described in Section 2.7. In our experiments, we use the ADAM optimizer for the outer loop loss function and vanilla SGD for the inner loop loss. This choice is motivated by a hybrid learning framework whereby the outer loop training can occur offline with large memory and compute resources (e.g. ADAM which requires more memory and compute), whereas the inner loop is constrained by hardware at the edge. The model is validated and tested on $\mathcal{T}^{val}$ and $\mathcal{T}^{tst}$, respectively. In the following, we describe the SNN model used with MAML.

4.2.2 MAML-compatible Spiking Neuron Model

The neuron model used in the SNNs in our work follows Leaky Integrate & Fire (LIF) dynamics as described in [74]. For completeness, we summarize the dynamics of the neuron model here:

$$\begin{aligned} u_i^t &= \sum_j w_{ij} p_j^t - \rho r_i^t + b_i, \\ s_i^t &= \Theta(u_i^t), \\ p_j^{t+\Delta t} &= \alpha p_j^t + (1-\alpha) q_j^t, \\ q_j^{t+\Delta t} &= \beta q_j^t + (1-\beta) s_j^t, \\ r_i^{t+\Delta t} &= \gamma r_i^t + (1-\gamma) s_i^t, \end{aligned} \tag{4.1}$$

where u_i is the membrane potential, w_{ij} are the synaptic weights between pre-synaptic neuron j and post-synaptic neuron i and Δt is the timestep. Neurons emit a spike s_i^t at time t when the threshold of their membrane potential $\Theta(u_i^t)$ is reached. $\Theta(u_i^t)$ is the unit step function, where $\Theta(u_i) = 0$ if $u_i < u_{th}$, otherwise 1. p and q describe the traces of the membrane potential of the neuron and the current of the synapse, respectively. For each incoming spike to a neuron, each trace undergoes a jump of height 1 and decays exponentially if no spikes are received. The constants

$$\alpha = \exp(-\frac{\Delta t}{\tau_{\text{mem}}}), \beta = \exp(-\frac{\Delta t}{\tau_{\text{syn}}}) \text{ and } \gamma = \exp(-\frac{\Delta t}{\tau_{\text{rfr}}}) \tag{4.2}$$

reflect the time constants of the membrane p, synaptic q, and refractory r dynamics. Weighting the trace p_j with the synaptic weight w_{ij} results in the Post-Synaptic Potential (PSP) of post-synaptic neuron i caused by pre-synaptic neuron j. The constant b_i is a bias current representing the intrinsic excitability of the neuron. The reset mechanism is captured by the dynamics of r_i, and the factors τ_{mem},τ_{syn} and τ_{ref} are time constants of the membrane, synapse, and reset dynamics respectively. Note that 4.1 is equivalent to a discrete-time version of the SRM with linear filters [47].

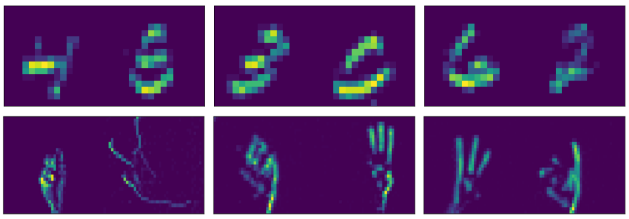


Figure 4.2: Top: Examples of Double N-MNIST tasks. Each sample contains a combination of two N-MNIST digits to make two-digit numbers. Bottom: Examples of Double ASL-DVS tasks. Each sample contains a combination of two ASL-DVS letters. In all examples, the images are DVS events summed over 100ms into frames.

To compute the second-order gradients, MAML requires the SNN to be twice differentiable. However, the spiking function Θ is non-differentiable. The surrogate gradient approach, where Θ is replaced by a differentiable surrogate function σ for computing gradients, has been used to successfully side-step this problem [125]. For MAML, the surrogate function can be chosen to be a twice differentiable function. Although many suitable surrogate gradient functions exist, the fast sigmoid function described in [201] strikes a good trade-off between simplicity and effectiveness in learning and is twice differentiable: $\sigma'(x) := \frac{1}{(\beta|x|+1)^2}$. All simulations in this work use the fast sigmoid function as a surrogate function.

Because SNNs is a special case of recurrent neural networks, it is possible to apply Automatic Differentiation tools for implementing the gradient [10, 134]. This also applies to the calculation of the second-order gradients needed for backpropagating the gradient of the inner loss.

4.2.3 Datasets

We benchmark our models on modifications of datasets collected using event-based vision sensors [98, 138], the Neuromorphic MNIST (N-MNIST) [132], and the American Sign Language Dynamic Vision Sensor (ASL-DVS) [13] datasets. NMNIST consists of 32×32, 300ms long event data streams of MNIST images recorded with an ATIS Camera [138]. The dataset

contains 60,000 training event streams and 10,000 test event streams. From the N-MNIST dataset, we create Double N-MNIST datasets. Each event stream of Double N-MNIST is a combination of two N-MNIST event streams to make 64×32, 300ms long event data streams of double-digit numbers that are downsampled to 32×16, 100ms long event data streams. Because there are ten digits in the original N-MNIST dataset, 100 different double-digit numbers can be created. These 100 different numbers can be used to create a meta-dataset with K=100 tasks, where each double-digit number represents one task. To train and evaluate the performance of meta-trained models we use an N-shot K-way few-shot learning approach. In N-shot K-way learning the model trains on N samples, or shots, of K number of classes, out of all of the available samples and classes in a dataset. The goal of this few-shot learning method is to train models to be able to generalize after seeing as few samples as possible.

We create N-shot K-way meta training, meta validation, and meta-test Double N-MNIST datasets from the training and test N-MNIST dataset. Each meta dataset consists of a subset of the 100 total possible tasks. The meta training dataset contains 64 classes, the meta validation dataset contains 16 classes, and the meta test dataset contains 20 red classes.

The ASL-DVS dataset contains 24 classes corresponding to letters A-Y, excluding J, in American Sign Language recorded using a DAVIS 240C event-based sensor [15]. Data recording was performed in an office environment under constant illumination. The dataset contains 4200 240×180 100ms long event data streams of each letter, for a total of 100,800 samples. Like Double N-MNIST, each event stream of Double ASL-DVS is a combination of two ASL-DVS event data streams to make 480×360, 100ms long event data streams of two letters that are downsampled to 80×30, 100ms long event data streams. Out of the 24 classes, 576 different tasks consisting of double ASL letter event streams can be used to create N-shot K-way meta datasets with each double ASL letter representing a task. From the ASL-DVS dataset, we create Double ASL-DVS N-shot K-way meta training, validation, and test datasets. The meta training dataset contains 369 red classes, the meta validation

dataset contains 92 red classes, and the meta test dataset contains 115 red classes. Example images from the Double N-MNIST and Double ASL-DVS datasets are shown in Fig. 4.2.

The N-Omniglot dataset [97] is a neuromorphic variant of the Omniglot dataset. The original Omniglot dataset contains 1623 categories of handwritten characters, and 20 samples for each category written by Amazon's Mechanical Turk participants [86] and stroke data was recorded. The N-Omniglot leveraged the stroke data to animate writing tracks on a standard 60Hz display. A davis346 camera was then used to capture these spatiotemporal visual patterns with μs timestamps [97]. The tracks for each character sample lasted several seconds. Both spatial dimensions of the davis346 output was downsampled by a factor 10, resulting in $2 \times 34 \times 26$ dimensional event streams. To reduce the effect of the 60Hz flicker, we downsampled the timestamp by a factor of 100000. Since the SNN is simulated on a time step equivalent to $1ms$, this means an acceleration by a factor of 100 with respect to the davis346 operation, because 100 ms of the davis346 is used for 1 time step of the simulated neuron. In comparison, the SNN baseline provided by the authors in [97] used an acceleration factor that was varied between 333 and 1000. We used the same training and validation split as with the Omniglot dataset. Note there there is no test dataset in this split.

For all datasets, gradients were truncated beyond the last 30ms from the end of the sequence. Therefore, not all of the gradients are used across the time sequence for learning. 4.3 shows explorations with a different number of truncation values showing that reducing the number of truncation steps gradually reduced the accuracy, and truncation at the final step did not converge. In 4.3 step size refers to the learning rate of the inner loop adaptation. The step size is increased as the truncation is decreased to keep the learning rate constant proportional to the truncation. Truncation at 30ms provided the best trade-off between accuracy and memory footprint.

4.2.4 Model Architecture

Table 4.1: SNN MAML Network Architecture

Layer	Kernel	NMNIST Output	ASL-DVS Output	N-Omniglot Output
input		32×16×2	80×30×2	34×26×2
1	32c5p0s1	32×16×32	80×30×32	34×26×32
2	2a	16×8×32	40×15×32	17×13×32
3	64c5p0s1	16×8×64	40×15×64	17×13×64
4	2a	8×4×64	20×7×64	8×6×64
5	128c5p0s1	8×8×128	20×7×128	8×6×128
6	2a	4×2×128	10×3×128	4×3×128
output	-	K=5	K=5	K=5

Notation: `Ya` represents `YxY` max pooling, `XcYpZsS` represents `X` convolution filters (`YxY`) with padding Z and stride S.

The architecture for all models trained are similar, consisting of three convolutional layers and a linear output layer. The SNN model architecture used here followed closely existing models for Deep Continuous Local Learning [74], and is summarized in Table 4.1. SNN output membrane potentials are encoded into classes by using the output neuron with the highest membrane potential as the classification.

4.3 Results

4.3.1 Few-shot Learning Performance on Double NMNIST and Double ASL Tasks

For the Double NMNIST and the ASL datasets, we ran 5-way 1-shot learning experiments on models trained using MAML. The MAML models were meta-trained on the meta training tasks ($\mathcal{T}^{trn}$) with the meta validation tasks ($\mathcal{T}^{val}$) used to compute the loss gradient in the outer loop. The meta-trained models were then tested on the meta-test tasks ($\mathcal{T}^{tst}$). A summary of our results for each dataset is shown in Table 4.2. The results in Table 4.2 are obtained from averaging the inference performance of a meta-trained model over 10

trials on the test datasets (D^{tst}) of the meta validation and meta test tasks. Each trial has different random batches of data sampled from training tasks $\mathcal{T}^{trn}$ and validation tasks $\mathcal{T}^{val}$ respectively. All experiments only used a single inner loop gradient step (*i.e.,* in MAML N was set to 1). Additionally, we compare the results of the SNNs to equivalent non-spiking models meta-trained with MAML as well as SNNs trained with first-order MAML. First-order MAML ignores all terms involving the second-order gradients and is thus similar to the joint training of the tasks. This has the advantage of reducing the memory footprint required for learning but is known to reduce the accuracy of the meta-trained model [41]. For the non-spiking models, the input data is first converted from the address event representation to static images by summing the events over the time dimension.

The results show that both spiking and non-spiking MAML achieve 1-shot learning performance on the datasets comparable to the state-of-the-art performance shown by non-meta models. The state-of-the-art test accuracy for standard non-meta model training on the NMNIST dataset with SNNs is 99.2±0.02% accuracy [154]. The SNN achieves 98.23±1.12% test accuracy on Double NMNIST in a one-shot learning scenario. Our SNN network achieves a test accuracy on the Double ASL-DVS dataset of $96.04 \pm 2.31\%$. For both datasets, first-order MAML performed significantly worse, highlighting the importance of differentiable surrogate gradients for successful meta-training.

On average MAML on SNNs tend to match or outperform non-spiking MAML on event-based datasets. This is likely because the dynamics of the SNN neurons are well suited for processing and learning the Spatio-temporal patterns of the event-based data streams the datasets are composed of.

Table 4.2: 1-Shot 5-Way Accuracy Results. Validation accuracy indicated accuracy over the test datasets (D^{tst}) in the meta validation set $\mathcal{T}^{val}$ and Test Accuracy indicates accuracy on the meta test set $\mathcal{T}^{tst}$. FOMAML is First Order MAML which does not use second order gradients therefore only needing one gradient update step instead of two.

Task	Algorithm	Validation Accuracy	Test Accuracy
Double N-MNIST	MAML (SNN)	98.76±1.05%	98.23±1.12%
	MAML (CNN)	99.09±.53%	98.35±1.26%
	FOMAML (SNN)	92.59±1.0%	92.63±.74%
Double ASL DVS	MAML (SNN)	95.77±.88%	96.04±2.31%
	MAML (CNN)	94.93±.92%	94.97±1.63%
	FOMAML (SNN)	94.97±1.12%	94.27±.6%

Table 4.3: Effect of truncation of gradients on double NMNIST learning accuracy

Truncation	Test Accuracy	Step Size
50	98.32 %	1.0
30	98.46 %	1.0
15	98.44 %	2.0
10	97.66 %	3.0
5	98.16 %	6.0
1	34.20 %	15.0
1	29.59 %	30.0

4.3.2 One-shot Learning on the N-Omniglot Dataset

We tested the same model as above on the N-Omniglot dataset [97]. This dataset consists of 1623 categories of characters, each repeated 20 times by a different writer. Stroke data was used to drive a target on a screen, which was recorded by a neuromorphic vision sensor. As in the work published by Li *et al.*, we binned the data samples in space and time (see methods).

On the 5-way, 1-shot task considered here, our MAML SNN outperformed all baselines reported in Li *et al.* (Tab. 4.4).

The higher number of time steps is most likely the reason for the increased performance, as our work sees more of the data, however the differences in the network architecture could

Algorithm	Time Steps	Test Accuracy
MAML (SNN, this work)	100	93.01±.66%
MAML ([97])	4	77.2±.4%
Siamese Net ([97])	4	73.1±0.6%

Table 4.4: Comparison of Meta-learning Methods on the N-Omniglot Dataset

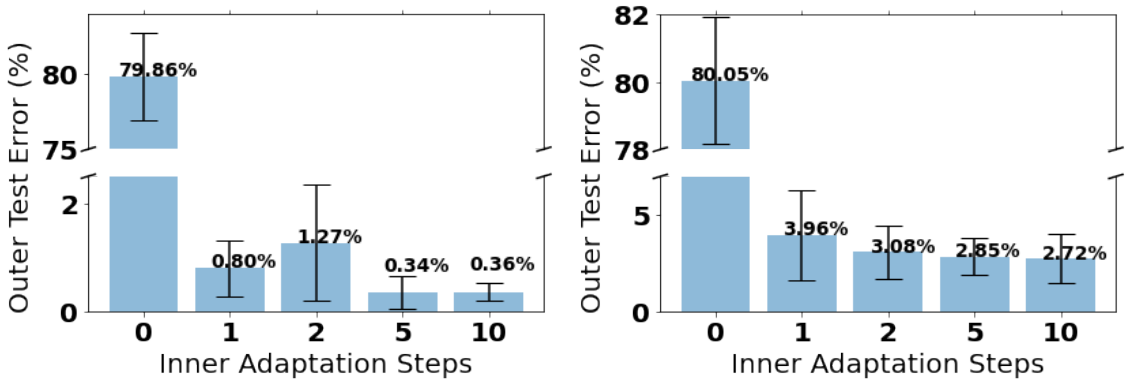


Figure 4.3: Example of how changing the number of inner loop training steps during meta-testing affects the error of the meta-trained model. Accuracy increases as the number of gradient steps increases and without adapting to a new task the model will have very high error. Left: Double NMNIST. Right: Double ASL-DVS.

have influenced our higher performance as well.

4.3.3 Generalization of Learning Performance

MAML requires selecting hyper-parameters such as the number of update steps and the learning rates. In real-world scenarios, the input constraints cannot be tightly controlled, leading to potential mismatches with the MAML hyper-parameters. For example, in a real-time gesture learning scenario, the parameter update schedule may not be tightly linked to the time the gesture is presented. Here we study the ability of MAML trained SNNs to generalize across different input conditions.

The ability of MAML to generalize learning performance across its settings, such as the number of steps, has been previously documented for conventional ANNs [42].

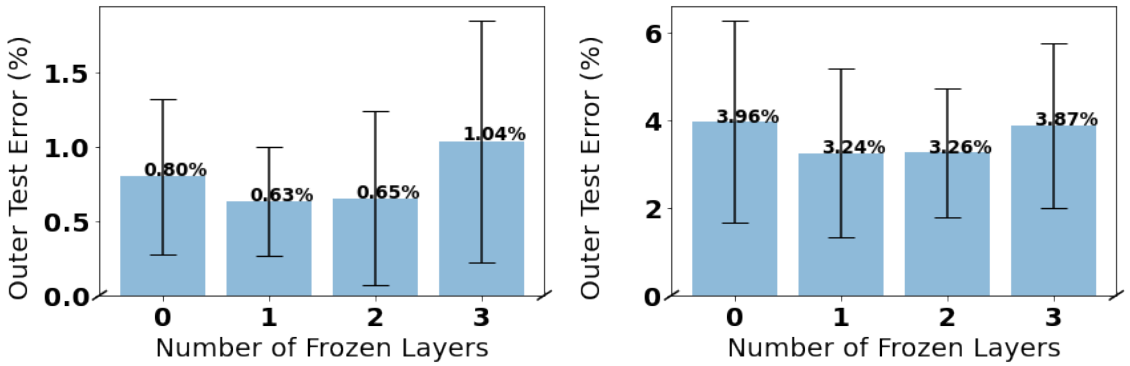


Figure 4.4: Example of how freezing layers of the network during inner loop adaptation does not greatly impact learning performance. This supports the network is using feature reuse as in [142]. Left: Double NMNIST. Right: Double ASL-DVS.

Here, we demonstrate that this feature extends to our SNN. Using an SNN MAML network meta-trained on the Double NMNIST dataset, we varied the number of gradient steps during inner loop adaptation on test data. The Fig. 4.3 shows how changing the number of gradient steps during inner loop adaptation affects the one-shot 5-way learning performance on each dataset. On both datasets, as the number of inner loop gradient steps increases the performance increases. Therefore there is a trade-off between the computational overhead of performing multiple gradient steps during one-shot learning and accuracy.

We also show how the learning performance is affected when layers of the network are frozen during few-shot learning. Using a network meta-trained on the Double NMNIST dataset, we progressively froze layers of the network to observe the impact on performance, which is shown in Fig. 4.4. Even when all layers of the network were frozen, in this case, three, there is not a big impact on performance. This gives further evidence to the claim that MAML learns a suitable representation for few-shot learning instead of rapid learning [142]. This is interesting from an engineering perspective, as the network with a meta-learned initialization can achieve high performance on learning new tasks with only one gradient update and only at the final layer. When all but the last layer are frozen during the inner loop, there is no requirement to backpropagate the errors. This greatly simplifies the learning rule implemented in hardware and is thus suited for real-time adaptation in neuromorphic

Table 4.5: Comparison of the Magnitude of Updates Between MAML and non-MAML Learning

Algorithm	Avg Magnitude	Sum of Magnitudes	Max Magnitude
MAML Outer Loop	7.72e-05±.0002	0.296±0.599	.002±.0017
MAML Inner Loop	.0044±.0007	17.03±2.69	0.1548±0.16
Non-Meta	0.0005±.0002	0.5499±0.2412	0.0011±.0002

hardware as demonstrated in previous work [164].

4.3.4 MAML Few-shot Learning Relies on Few, Large Magnitude Updates

To obtain adequate generalization, conventional deep learning relies on many, small magnitude updates across a large dataset. This is achieved using a relatively small learning rate. The usage of small learning rates is challenging on a physical substrate, as it requires high precision memory to accumulate the gradients across updates. This problem is further compounded by the fact that learning on a physical substrate cannot be easily performed using batches of samples.

Interestingly, few-shot learning has the opposite requirements: few but large magnitude updates. This result is extremely relevant for neuromorphic hardware which uses low precision parameters.

Likewise, we observe that the SNN MAML model only needs a few, large magnitude parameter updates for few-shot learning. The Table 4.5 shows the truncated values of the update magnitude between two training iterations of the output layer for a meta-model and an equivalent non-meta model both trained on Double NMNIST. The Fig. 4.5 gives a more detailed picture by showing histograms of the weight updates. Comparing the MAML inner loop and the non-meta model's magnitudes, the average update of the inner loop is an order

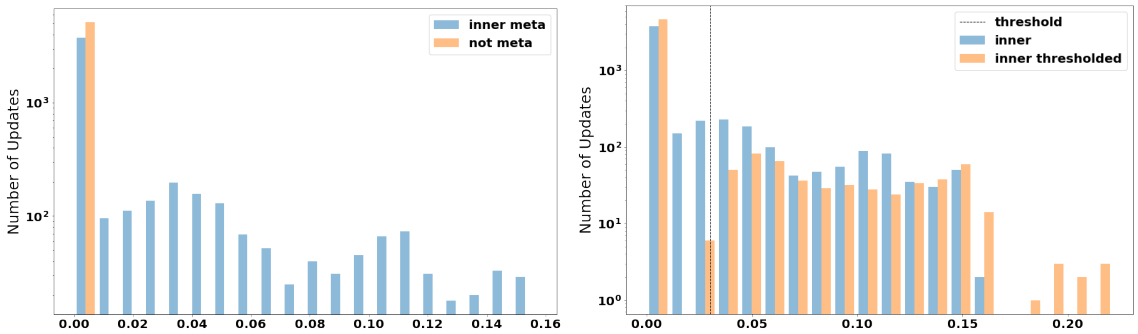


Figure 4.5: A comparison of the weight update magnitudes on Double NMNIST data, shown on a log scale, of an inner loop update and (left) and equivalent not meta trained model update; and (right) an inner loop update that is thresholded. MAML makes fewer non-zero weight updates that are large in magnitude compared to non-meta models. Additionally when thresholded MAML makes fewer non-zero weight updates that are larger in magnitude.

of magnitude larger than the equivalent non-meta model's update. To summarize, we find that, first, meta-trained models only need one adaptation step to achieve high accuracy when learning a new task (see Table 4.2), and second, that these models only need a few updates with a large magnitude to perform few-shot learning (see Table4.5, Fig. 4.5).

Additionally, the magnitude of the weight updates in the inner loop during meta training and adaptation can be thresholded to use even fewer and larger magnitude updates. During the inner loop adaptation, instead of always updating the parameters the update is gated by a threshold which is described in Eq. (4.3).

$$\theta_k^n = \begin{cases} \theta_k, \text{if} \Delta w > \Theta, \\ \theta_{k-1}, \text{if} \Delta w < \Theta \end{cases} \tag{4.3}$$

where θ_k^n are the parameters of the model, Δw is the magnitude of the update, and Θ is the threshold. Thresholding the updates forces the parameters to be larger with fewer updates, which is shown in the Fig. 4.5. The threshold used in the Fig. 4.5 was equal to 5% of the value of the range of magnitude updates.

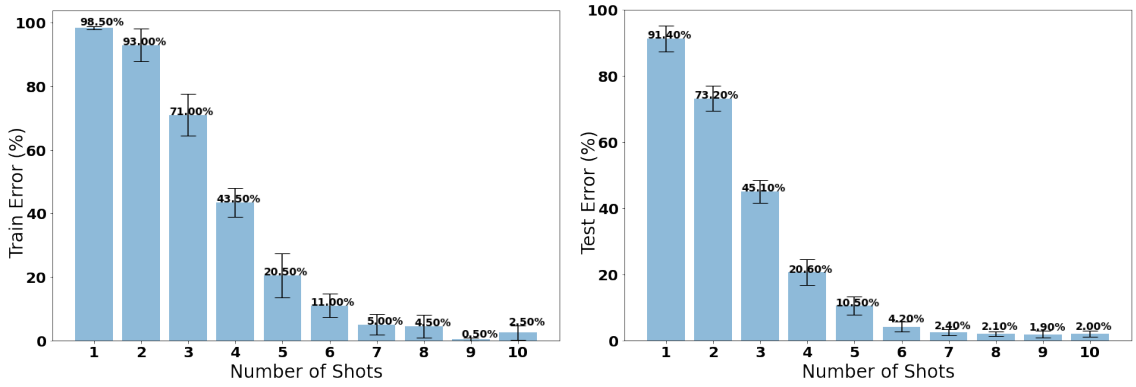


Figure 4.6: The error of a pre-trained non-meta model on Double NMNIST training classes using transfer learning to learn the test set of classes. The model requires more shots than MAML to achieve comparable performance.

4.3.5 Comparison to Transfer Learning

A generalization problem involves learning a function, or model, whose behavior is constrained through a dataset that can make predictions about (*i.e.,* learn features than can transfer to) other samples. A task domain consists of datasets that are related by a common domain, for example, datasets that all consist of double-digit numbers. Learning on one task in the domain to improve performance on another task is commonly referred to as transfer learning. Meta-learning can cast transfer learning as a generalization problem because each example, or task, in a given task domain, is a generalization problem instance in meta-learning, which means generalization in meta-learning corresponds to the ability to transfer knowledge between the different problem instances[5, 83].

We compare meta-learning to the conventional transfer learning for few-shot learning where a model is pre-trained on a subset of classes within a task domain and then the pre-trained features are transferred to another model that learns to classify new classes within the task domain. For comparison to the Double NMNIST SNN MAML model we first pre-trained an SNN model on the 64 classes of the training dataset. Then we transferred the features to a new model that had an untrained last layer. The model was trained and tested on 20 of the remaining classes where all layers except the last layer were frozen. The model was trained

on one shot of data at a time and then tested on 20 unseen shots of data. The few-shot transfer learning results are shown in Fig. 4.6. The results shown in the table were averaged over 10 trials. After the model trains on about 9 or 10 shots of data, the model achieves comparable accuracy to the SNN MAML model shown in Table 4.2. Comparing that to the high accuracy of the SNN MAML models on new tasks after using only one shot of data (see. Fig. 4.3, Table 4.2.), we can conclude that SNN MAML can adapt to a new task using fewer shots of data.

4.3.6 Quantization of Parameters during Meta-Training and Meta-testing

The limited availability of memory in neuromorphic devices dictates the necessity for quantization. In most mixed signal and digital CMOS hardware, weights are stored in Static RAM in a fixed point format. In the Loihi chip for instances, weights can be programmed to be quantized between 1 bit and 8 bits [32], and updates weights are rounded stochastically [119]. MAML presents an interesting opportunity to meta-train a network with quantization. Furthermore, quantization in the outer loop can be decoupled from the quantization in the inner loop. We have used the following quantization scheme:

$$w \leftarrow Q_{C,S}(w + \Delta w) \tag{4.4}$$

where $Q_{C,S}$ is a quantization function that updated real-values weights stochastically ($S = 1$) or deterministically ($S = 0$). The subscript C refers to allowed levels, which were equally spaced within the interval [-1,1] and symmetric around 0. Gradients of Q were set to straight through, *i.e.*, $\nabla_x Q_{C,S}(w(x)) = \nabla_x w(x)$. This quantization scheme makes it compatible with second-order MAML and does not require a copy of real-valued weights during inner loop learning as typically done when quantizing neural networks [167].

qout	2	2s	3	3s	4	4s	8	8s
qin								
2	21.6%	23.8%	21.3%	23.2%	95.7%	96.3%	91.4%	95.9%
2s	96.7%	N/A	95.0%	N/A	83.7%	N/A	91.5%	N/A
3	22.0%	20.8%	20.9%	25.8%	93.3%	95.0%	94.7%	87.2%
3s	95.7%	N/A	91.1%	N/A	94.2%	N/A	96.6%	N/A
4	92.6%	81.9%	98.8%	56.5%	99.0%	97.9%	98.8%	99.1%
4s	97.5%	N/A	98.2%	N/A	98.8%	N/A	97.9%	N/A
8	97.5%	88.3%	97.0%	85.6%	98.5%	97.8%	98.9%	99.2%
8s	98.6%	N/A	98.7%	N/A	97.9%	N/A	98.5%	N/A

Table 4.6: MAML SNN Quantization Experiments. qin: Number of bits quantized to in the inner loop, qout: Number of bits quantized to in the outer loop, s: Stochastic Rounding

The test accuracy on selected configurations of C and S are shown in table 4.6 with C converted to the bits of precision quantized to. Our explorations revealed that quantization is achievable with a very small loss in performance even when quantized to only 2 bits of precision if stochastic rounding is used for inner loop updates as experimental results show in Table 4.6. However using stochastic rounding in the outer loop for low precisions is detrimental with Table 4.6 showing these models do not converge unlike the models that have stochastic rounding in the inner loop at low precisions.

Furthermore, we found that the quantization in the outer loop is more robust to fewer levels of quantization compared to the inner loop quantization. This interesting fact reveals that MAML can learn a scaffold of network weights on which inner loop synaptic plasticity can learn adequate weights for solving a new task. We speculate that this differential quantization scheme may become useful in hybrid memory designs such as in [3], where initialization weights are stored in precise but area-expensive Static RAM, while online weight updates are made in imprecise but power- and area-efficient emerging devices.

4.3.7 Meta-Training with Approximations of Gradient Descent Learning Rules

Several methods for training SNNs using gradient descent have been introduced. The relationship of gradient descent with synaptic plasticity means that meta-training is a form of programming of synaptic plasticity. Programming synaptic plasticity is important for mixed-signal neuromorphic hardware implementation of synaptic plasticity because such circuits are prone to mismatch [25, 140]. Even in digital neuromorphic technologies, training programming synaptic plasticity can be useful to approximate an optimal learning rule that would otherwise be impossible or too expensive to implement [32].

To demonstrate the meta-training of a synaptic plasticity rule, we demonstrate MAML on error-triggered synaptic plasticity [135], a variant of Equation 2.2 that has been implemented in the neuromorphic Loihi chip [164] using the SOEL local learning rule described in Section 3.4. Here we demonstrate that MAML can meta-train a neural network with such a rule, including the estimation of the threshold parameter θ. In this work, we used floating point division instead of an integer division.

Table 4.7 shows the results of meta-training with error-triggered synaptic plasticity on the double N-MNIST, double ASL DVS, and N-Omniglot datasets shown previously without error-triggered synaptic plasticity. In all tested cases, the test accuracy of each dataset is comparable to using MAML without the error-triggered rule. Therefore our method adapted from [164] previously used in neuromorphic hardware is suitable for use in neuromorphic hardware.

These results demonstrate that an approximate gradient-based synaptic plasticity rule can be fine-tuned via meta-learning to classification results on par with the exact learning rule. This is particularly interesting for neuromorphic hardware implementations, where exact synaptic plasticity rules are in contradiction with hardware constraints.

Table 4.7: 1-Shot 5-Way Accuracy Results with error-triggered synaptic plasticity. Validation accuracy indicated accuracy over the test datasets (D^{tst}) in the meta validation set $\mathcal{T}^{val}$ and Test Accuracy indicates accuracy on the meta test set $\mathcal{T}^{tst}$.

Task	Algorithm	Validation Accuracy	Test Accuracy
Double N-MNIST	MAML SOEL $\theta = .05$	98.57±1.13%	98.72±0.20%
Double ASL DVS	MAML SOEL $\theta = .05$	97.81±0.58%	97.61±0.96%
N-Omniglot	MAML SOEL $\theta = .05$	89.41±2.70%	-

4.4 Meta Trained Models on Loihi

Here we evaluate SNNs meta-trained with the Loihi simulator described in section 3.3 that perform one-shot adaptation on the loihi hardware using the SOEL learning rule. To meta-train models compatible with the Intel Loihi we trained SNNs with Intel's Lava-Dl open source library [131] which is the updated version of SLAYER. Lava-Dl is a library of deep learning tools that support offline training and online training and inference methods for Deep SNNs. These tools include a functional differentiable simulator of the Intel Loihi's neuron dynamics, and quantization aware training with the same stochastic rounding and precision used in the Loihi. Therefore models meta-trained with Lava-Dl are transferable for use with Intel's Loihi hardware.

The results of our meta-learning experiments are shown in Table 4.8. In Table 4.8 we compare the implementation of Lava-Dl trained models, models ran on the Loihi with the SOEL learning rule, and the full-precision Decolle based models with increasing constraints towards the same constraints Lava-Dl and Loihi use. All experiments were done with 5-way 1-shot learning, and only used a single inner loop gradient step. The results show the accuracy on meta-test data. To test models, each model is first given 1-shot of data to learn from in an inner loop gradient step, then average the inference performance on 10-shots of the test data, and repeat the process 200 times. The two main differences between the neuron model used by Lava-Dl/Loihi and Decolle are 1) how the neuron resets

Table 4.8: One-shot Learning Accuracy Comparison

Dataset	Method	Test Accuracy
Double NMNIST	Pytorch MAML Decolle [165]	98.2 ± 1.1%
	Pytorch MAML Decolle+hard reset	94.7 ± 1.6%
	Pytorch MAML Decolle+quantized	97.7±1.0%
	Pytorch MAML Decolle+hard reset+quantized	91.5 ± 2.4%
	Pytorch MAML Lava-dl	91.1 ± 2.5%
	MAML 3-F Plasticity on Loihi 1	93.3 ± 1.3%
	KNN	83.0 ± 9.8%
Double ASL	Pytorch MAML Decolle [165]	96.0 ± 2.3%
	Pytorch MAML Decolle+hard reset	93.2 ± 2.2%
	Pytorch MAML Decolle+quantized	91.4 ± 2.1%
	Pytorch MAML Decolle+hard reset+quantized	92.4 ± 2.0%
	Pytorch MAML Lava-dl	92.14 ± 2.6%
	MAML 3-F Plasticity on Loihi 1	91.3 ± 3.6%
	KNN	39.6 ± 9.6%
DvsGesture actor+class	Pytorch MAML Decolle	95.1 ± 1.6%
	Pytorch MAML Decolle+hard reset	96.9 ± 1.3%
	Pytorch MAML Decolle+quantized	95.6 ± 1.4%
	Pytorch MAML Decolle+hard reset+quantized	95.1 ± 2.0%
	Pytorch MAML Lava-dl	95.7 ± 1.0%
	MAML 3-F Plasticity on Loihi 1	97.1 ± 2.0%
	KNN	23.2 ± 4.7%

Table 4.9: Loihi One Sample Learning Stats

Dataset	Energy (mJ)	Time (ms)	Power (mW)	Cores
Double NMNIST	0.751±	0.900±	0.834±	14
Double ASL	0.974±	1.16±	0.839±	14
DvsGesture Actor	1.031±	1.140±	0.904±	26

and 2) the quantization. Decolle neuron's by default use floating point precision and their membrane potential decays exponentially after spiking rather than resetting to zero, or hard resetting, like the Loihi neuron model. Therefore we demonstrate how the performance is impacted by the difference in the neuron model's reset and quantization in Table 4.8. On all of the datasets the Lava and Loihi models performed similarly to the Decolle model that used the neuron dynamics of the Loihi. The accuracy on the Double NMNIST and Double ASL datasets drops by approximately 5% when using the Loihi neuron model while the DvsGesture dataset performance is not impacted by the Loihi neuron model dynamics.

This could be because the DvsGesture dataset is much smaller than the other two datasets. The 5-way 1-shot learning on the DvsGesture dataset with a meta-trained model on Loihi using SOEL for last layer adaptation gives approximately a 31% improvement over the 5-way 1-shot learning results using SOEL on Loihi demonstrated in prior work that did not use meta-learning [164]. Accuracy is determined based on the highest firing neuron. If the neuron with the highest firing rate corresponds to the correct class then the sample being presented is correctly classified.

Additionally we compare the SNN meta-trained models to a k-nearest-neighbors (KNN) model. KNNs use supervised learning to make predictions based on the similarity of a new data point to its k nearest neighbors in a training dataset. We train and test the KNN in a similar fashion to the meta-trained SNN models, giving 1-shot of 5-classes of data to train on before testing on 10-shots of test data and averaging the results over 200 trials. The KNN performs on average about 15-73% worse on the 5-way 1-shot learning tasks than the meta-trained SNN models, particularly on the Double ASL and DvsGesture datasets with the KNN performance close to random on the DvsGesture dataset.

Table 4.9 shows energy, time, and power statistics for learning one sample of data using SOEL from the meta-trained model. For each sample, meta-trained models on Loihi using SOEL for one-shot learning use approximately 1mJ of energy or less and take only about 1ms to learn a sample making one-shot learning with Loihi using our methods not only accurate but power efficient as well.

4.5 Discussion

Neuromorphic hardware is particularly well suited for online learning at the edge. Here, we demonstrated how to pre-train SNNs to perform one-shot learning using MAML. The

SNN MAML models used the surrogate gradients method to overcome non-linearities that occur in gradient-based training of SNNs. We demonstrated our results on combinations of event-based datasets recorded using a neuromorphic vision sensor.

The effective batch size (=1), the precision required for learning from scratch, and the potential correlation in data samples in neuromorphic learning are serious obstacles to deploying neuromorphic learning in practical scenarios. Fortunately, learning from scratch on the device is generally not even desirable due to robustness and time-to-convergence issues, especially if the devices are intended for edge applications. Some form of offline pre-training can alleviate these issues, and MAML is an excellent tool to automate this pre-training.

Our results showed that a meta-trained SNN MAML model can learn new event-based tasks in one or a few shots within a task domain. This enables learning in real-world scenarios when data is streaming, online, and observed only once. Additionally, the model can relearn prior learned tasks in one or few shots which greatly reduces the impact of catastrophic forgetting because the model does not need to retrain for many iterations.

In hardware for training neural networks, weight updates must be rounded to fall onto values that are resolved at the desired resolution, thereby placing a lower bound to the learning rate. Conveniently, in few-shot learning, updates are of large magnitude (Fig. 4.5) and a single update must be sufficient to make a change in the output, effectively implying that the learning rate is large.

One important limitation of MAML is its time and space complexity. Computing second order gradients as in MAML is both memory and compute intensive (second order gradients are one order more expensive than computing gradients). Meta-training SNN MAML models require considerable computing power and memory. The tasks are stored in memory as a sequence of length T with network computations calculated over the sequence. Our method re-initializes the network dynamics each iteration of training by inputting a partial sequence

of a data sample into the network that executes the dynamics but does not update the network. Additionally, gradients must be computed and stored both in the inner loop and outer loop and backpropagated through the network to meta train. This largely prevents learning for a large number of inner loops. While first-order MAML requires less memory and compute for each update, it performed significantly worse than MAML on our model and benchmarks. Other first-order MAML methods such as REPTILE [127] are an alternative that can reduce memory usage at the cost of more compute time and some accuracy.

We note that the model agnostic property of MAML renders it complementary to techniques that improve the performance of SNNs, such as batch normalization [79] and dropout [123], or methods to train SNNs such as parameter conversions [34, 149] and neural architecture search [39] such as those applied to SNNs [78]. The latter methods can be directly integrated an additional step in the outer loop update.

We argue that successful meta-learning on SNNs holds promise to help reduce training data iterations at the edge, making it essential to designing and deploying neuromorphic learning machines to real-world problems; prevent catastrophic forgetting and learn with low-precision plasticity mechanisms. As a bi-level learning mechanism, our results point towards a hybrid framework whereby SNNs are pre-trained offline for online learning. As a result, we expect a strong redefinition of synaptic plasticity requirements and exciting new learning applications at the edge.

So far the methods detailed in the work of the thesis have concerned supervised learning from labeled data. In most real-world scenarios however data is not readily labeled for learning. In the next chapter we describe a self-supervised learning approach that can overcome the problem of learning from unlabeled data in the real world.

Chapter 5

Towards Self-supervised Learning at the Edge: Feature Representation Learning With Variational Auto-Encoders

5.1 Introduction

Prior work has demonstrated how online supervised learning with labeled data can be used for tasks such as rapid, event-driven learning from neuromorphic sensor data [163]. However, real-world data is unlabeled and, in the case of classification, can have classes that were not anticipated. Therefore, to leverage real-world data, labels must be generated by a supervisor in real-time without *a priori* knowledge of the number of classes.

Additionally, while a trained classifier is trained to generate class labels, it cannot generalize to new classes. This is compounded by the fact that neural network classifiers trained

using gradient descent are usually overconfident of their classification, making the learning of new classes impractical. An alternative approach is to use the intermediate layers of a trained neural network classifier as pseudo-labels or features for learning classes [129]. In this work, we formalize this idea using an event-driven guided Variational Auto-Encoder (VAE) which is trained to generate an embedding that disentangles according to labels in a labeled dataset and that generalizes to new data. The resulting embedding space can then either be used for (pseudo)labels for supervised learning or for measuring data similarity. We focus our demonstration of the guided VAE on DVS gesture learning problem because of the availability of an event-based gesture dataset and the high relevance of gesture recognition use cases [4].

Neuromorphic Dynamic Vision Sensors (DVS) inspired by the biological retina capture temporal, pixel-wise intensity changes as a sparse stream of binary events [45]. This approach has key advantages over traditional RGB cameras, such as faster response times, better temporal resolution, and invariance to static image features like lighting and background. Thus, raw DVS sensor data intrinsically emphasizes the dynamic movements that comprise most natural gestures. However, effectively processing DVS event streams remains an open challenge. Events are asynchronous and spatially sparse, making it challenging to directly apply conventional vision algorithms [46, 45].

Here, we take advantage of SNNs ability to efficiently process and learn from event-based data while taking advantage of temporal information [125] by incorporating a convolutional SNN into a Variational Autoencoder (VAE) to encode spatio-temporal streams of events recorded by the DVS (Figure 5.1). The goal of the VAE is to embed the streams of DVS events into a latent space which facilitates the evaluation of event data similarity for self-supervised learning from real-world data. To best use the underlying hardware, we implement a *hybrid* VAE to process the DVS data, with an SNN-based encoder and a conventional (non-spiking) convolutional network decoder. To ensure the latent space represents features which are

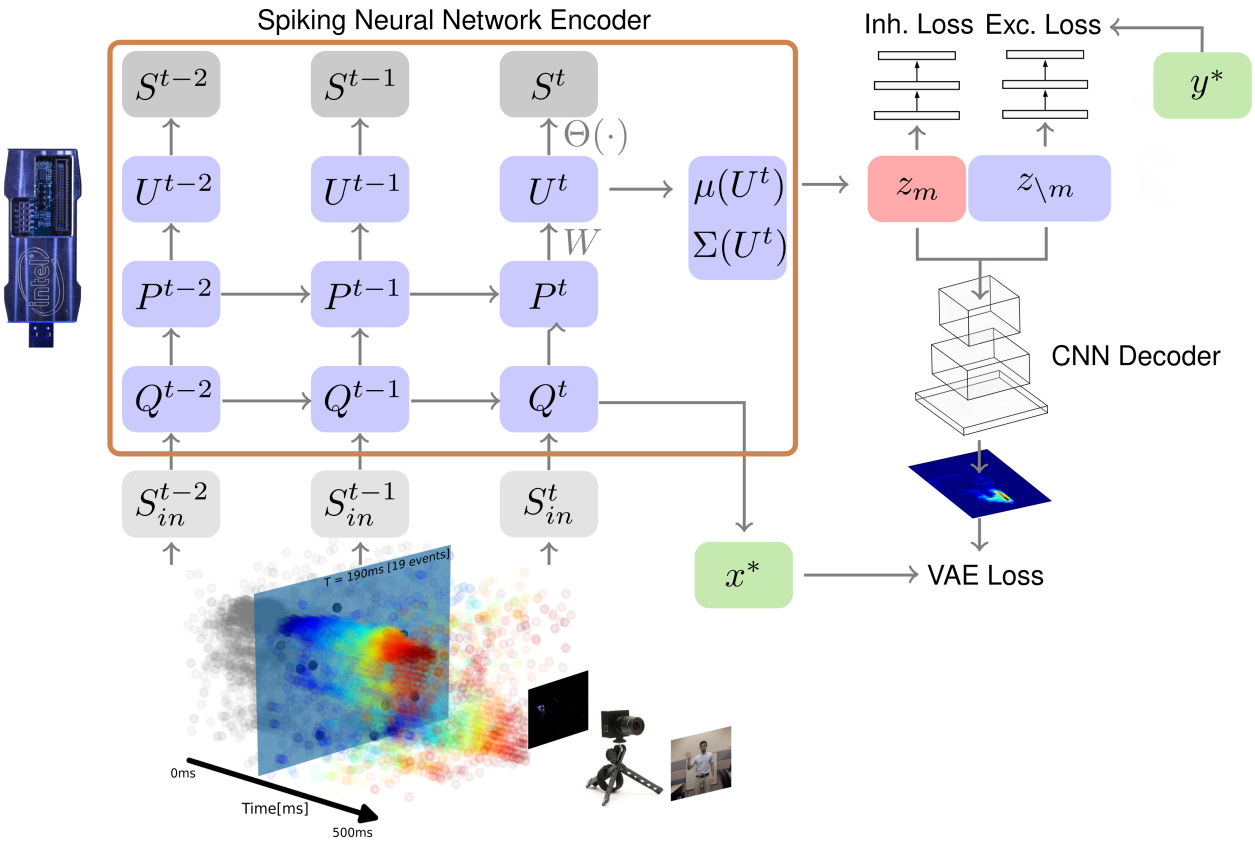


Figure 5.1: The Hybrid Guided-VAE architecture. Streams of gesture events recorded using a Dynamic Vision Sensor (DVS) are input into a Spiking Neural Network (SNN) that encodes the spatio-temporal features of the input data into a latent structure z. P and Q are pre-synaptic traces and U is the membrane potential of the spiking neuron. For clarity, only a single layer of the SNN is shown here and refractory states R are omitted. To help disentangle the latent space, a portion of the z equal to the number of target features y^* is input into a classifier that trains each latent variable to encode these features (Exc. Loss). The remaining z, noted $z_{\setminus m}$ are input into a different classifier that adversarially trains the latent variables to not encode the target features so they encode for other features instead (Inh. Loss). The latent state z is decoded back into x^* using the conventional deconvolutional decoder layers.

perceptually salient and useful for classification tasks, we use a guided VAE to disentangle the features that account for variation in the underlying structure of the data.

Our Hybrid Guided-VAE encodes and disentangles the variations of the structure of event data allowing for the clustering of similar patterns, such as similar looking gestures, and assigning of pseudo-labels to novel samples, effectively learning a similarity measure between data samples. The key contributions of this work are:

1. End-to-end trainable event-based SNNs for processing neuromorphic sensor data event-by-event and embedding them in a latent space.
2. A Hybrid Guided-VAE that encodes event-based camera data in a latent space representation of salient features for clustering and pseudo-labeling.
3. A proof-of-concept implementation of the Hybrid Guided-VAE on Intel's Loihi Neuromorphic Research Processor.

The ability to encode event data into a disentangled latent representation is a key feature to enable learning from real-world data for tasks such as mid-air gesture recognition systems that are less rigid and more natural because they can adapt to each user.

5.2 Methods

5.2.1 Measuring Data Similarity

To automatically and objectively measure the similarity of event data between and across classes for automatic pseudo-labeling, we build a novel Hybrid Guided-VAE model that can take advantage of the DVS's temporal resolution and robustness while being end-to-end trainable using gradient descent on a variational objective. Hybrid VAEs that combine both

spiking and ANN layers have been used before on DVS event data for predicting optical flow. The hybrid architecture efficiently processes sparse spatio-temporal event inputs while preserving the spatio-temporal nature of the events [93].

5.2.2 Variational Autoencoders

VAEs are a type of generative model which deal with models of distributions $p(x)$, defined over data points $x \in X$ [80]. A VAE commonly consists of two networks, 1) an encoder (Enc) that encodes the captured dependencies of a data sample x into a latent representation z; and 2) a decoder (Dec) that decodes the latent representation back to the data space making a reconstruction $\tilde{x}$ using Gaussian assumptions for the latent space :

$$Enc(x) = q(z|x) = N(z|\mu(x), \Sigma(x)), \tag{5.1}$$

$$\tilde{x} \approx Dec(z) = p(x|z), \tag{5.2}$$

where q is the encoding probability model into latent states z that are likely to produce x, and p is the decoding probability model conditioned on z. The functions $\mu(x)$ and $\Sigma(x)$ are deterministic functions whose parameters can be trained through gradient-based optimization.

Using a variational approach, the VAE loss consists of the sum of two terms resulting from the variational lower bound:

$$\log p(x) \geq \underbrace{\mathbb{E}_{z\sim q} \log p(x|z)}_{\mathcal{L}_{ll}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\mathcal{L}_{prior}}. \tag{5.3}$$

The first term is the expected log likelihood of the reconstructed data computed using samples of the latent space, and the second term acts as a prior, where D_{KL} is the Kullback-Leibler divergence. The *VAE* loss is thus formulated to maximize the variational lower bound by maximizing $-\mathcal{L}_{prior}$ and $\mathcal{L}_{ll}$. A VAE's latent space captures salient information for representing the data X, and thus similarities in the data [89].

5.2.3 Disentangling Variational Autoencoders

VAEs do not necessarily disentangle all the factors of variation which can make the latent space difficult to interpret and use. Several approaches have been developed to improve the disentangling of the latent representation, such as Beta VAE [57] and Total Correlation VAE [22]. However, the disentangled representation these unsupervised methods learn can have high variance since disentangled labels are not provided during training. Furthermore, these methods do not learn to relate labels to the representations making them unable to provide pseudo-labels from unlabeled data. For these reasons, we employ a Guided-VAE, which has been developed specifically to disentangle the latent space representation of key features in a supervised fashion [35]. We describe here the supervised Guided-VAE algorithm, which is the basis of our hybrid model described in the next section.

To learn a disentangled representation, a supervised Guided-VAE trains latent variables to encode existing ground-truth labels while making the rest of the latent variables uncorrelated with that label. The supervised Guided-VAE model targets the generic generative modeling task by using an adversarial excitation and inhibition formulation. This is achieved by minimizing the discriminative loss for the desired latent variable while maximizing the minimal classification error for the rest of the latent variables. For N training data samples $X = (x_1, ..., x_N)$ and M features with ground-truth labels, let $z = (z_1, ..., z_m, ...z_M) \oplus z_{\setminus m}$ where z_m defines the "guided" latent variable capturing feature m, and $z_{\setminus m}$ represents the rest of the latent variables. Let $y_m(x_n)$ be a one-hot vector representing the ground-truth label for the m-th feature of sample x_n. For each feature m, the excitation and inhibition

losses are defined as follows:

$$\begin{aligned}\mathcal{L}_{Exc}(z,m) &= \max_{c_m}\left(\sum_{n=1}^{N}\mathbb{E}_{q(z_m|x_n)}\log p_{c_m}(y=y_m(x_n)|z_m)\right),\\ \mathcal{L}_{Inh}(z,m) &= \max_{k_m}\left(\sum_{n=1}^{N}\mathbb{E}_{q(z_{\setminus m}|x_n)}\log p_{k_m}(y=y_m(x_n)|z_{\setminus m}),\right)\end{aligned} \tag{5.4}$$

where c_m is a classifier making a prediction on the m-th feature in the guided space and k_m is a classifier making a prediction over m in the unguided space $z_{\setminus m}$. By training these classifiers adversarially with the VAE's encoder, it learns to disentangle the latent representation, with z_m representing the target features and $z_{\setminus m}$ representing any features other than the target features.

5.2.4 Dynamic Vision Sensors and Preprocessing

The recorded DVS event stream is provided as input to the Hybrid Guided-VAE network implemented on a GPU (network dynamics described in the following section). Each polarity of the event stream (x, y, p) is fed into one of the two channels of the first convolutional layer. Within the time step, most pixels have value zero, and very few have values larger than two.

The VAE targets take a different form because the decoder is not an SNN. Time surfaces (TS) are widely used to preprocess event-based spatio-temporal features [85]. TS can be constructed by convolving an exponential decay kernel through time in the event stream as follows:

$$TS^t_{x,y,p} = \epsilon^t * S^t_{DVS,x,y,p} \text{ with } \epsilon^t = \mathrm{e}^{-\frac{t}{\tau}} \tag{5.5}$$

where τ is a time constant. Here, we convolve over the time length of the input event data stream $S^t_{DVS,x,y,p}$. This results in two 2D images, one for each polarity, that are used as VAE

targets (*i.e.*,, for the reconstruction loss).

In continuous time, the Time Surface (TS) is defined as:

$$(\epsilon * S_{in})(t) = \int_0^t \epsilon(t-s) S_{in}(s) \mathrm{d}s \text{ with } \epsilon(s) = e^{\frac{-s}{\tau}}. \tag{5.6}$$

We demonstrate here that the solution of Q is equal to $(\epsilon * S_{in,j})(t)$ when $\tau = \tau_{syn}$. The differential equation governing $Q(t)$ is linear and can be integrated in a straightforward manner:

$$\begin{aligned} \tau_{syn} \frac{\mathrm{d}}{\mathrm{d}t} Q(t) =& - Q(t) + S_{in}(t), \\ Q(t) =& \int_0^t \exp(-\frac{t-s}{\tau_{syn}}) S_{in}(s) \mathrm{d}s \\ =& (\epsilon * S_{in})(t) \end{aligned} \tag{5.7}$$

where we have assumed $Q(0) = 0$. The last line is true if $\tau = \tau_{syn}$.

5.2.5 Hybrid Guided Variational Auto-Encoder

DVS cameras produce streams of events containing rich spatio-temporal patterns. To process these streams, event-based computer vision algorithms typically extract hand encoded statistics and use these in their models. While efficient, this approach discards important spatio-temporal features from the data [45]. Rather than manually selecting a feature set, we process the raw DVS events while preserving key spatio-temporal features using a spiking neural network (SNN) trained end-to-end in the Hybrid Guided-VAE architecture shown in Figure 5.1.

A key advantage of the VAE is that the loss can be optimized using gradient backpropagation. To retain this advantage in our hybrid VAE, we must ensure that the encoder SNN is also trainable through gradient descent. Until recently, several challenges hindered this: the spiking nature of neurons' nonlinearity makes it non-differentiable and the continuous-time

dynamics raise a challenging temporal credit assignment problem. These challenges are solved by the surrogate gradients approach [125], which formulates the SNN as an equivalent binary RNN and employs a smooth surrogate network for the purposes of computing the gradients. Our Hybrid Guided-VAE uses a convolutional SNN to encode the spatio-temporal streams in the latent space, and a non-spiking convolutional decoder to reconstruct the TS of the data. We chose an event-based encoder because the SNN can bridge computational time scales by extracting slow and relevant factors of variation in the gesture [186] from fast event streams recorded by the DVS. The SNN part encodes the sparse event data from the DVS into a lower dimensional latent representation that highlights the similarity between data samples, both within and across classes. We chose a conventional (non-spiking) decoder for three reasons: (1) for similarity estimation, we are mainly interested in the latent structure produced by the encoder, rather than the generative features of the network, (2) as we demonstrate in the results section, a dedicated neuromorphic processor [69, 32] only requires the encoder to produce this latent structure, and (3) SNN training is compute- and memory-intensive. Thus a conventional decoder enables us to dedicate more resources to the SNN encoder.

Our Hybrid Guided-VAE network architecture is shown in Figure 5.1 and the architecture description is provided in Table 5.1. The SNN encoder consists of four discrete leaky integrate and fire convolutional layers (see following section) followed by linear layers, and outputs a pair of vectors (μ, Σ) for sampling the latent state z. According to the guided VAE, part or all of latent state z is input into one of three connected networks, the excitation classifier, the adversarial inhibition classifier, or the decoder. The excitation classifier takes as input the first M latent variables in the latent space, where M is the number of target classes or features, and each of these latent variables, z_m, are trained to classify a target class or feature. The target features for the excitatory network are given as one-hot encoded vectors of length M. The excitation classifier is jointly trained with the encoder to train the first M latent variables to only encode information relevant to the corresponding target feature. The

inhibition classifier takes as input the remaining latent variables in the latent space, $z_{\setminus m}$, and are adversarially trained on two sets of targets. One set of targets is the same target features that the excitation classifier trains on. The other set of target features is a vector of length M but all of the values are set to 0.5 indicating that none of the values correspond to any target. The inhibition classifier is jointly trained with the encoder to train the remaining $z_{\setminus m}$ latent variables to not encode any information relevant to target features, forcing them to encode information for other features instead. The decoder is a transposed convolutional network that takes the full latent state z as input to construct the TS, denoted $\tilde{x}$ in Figure 5.1.

Table 5.1: Hybrid Guided-VAE architecture

Layer	Kernel	Output	Layer Type
input		32×32×2	DVS128
1	2a	16×16×2	SNN LIF Encoder
2	32c7p0s1	16×16×32	
3	1a	16×16×32	
4	64c7p0s1	16×16×64	
5	2a	8×8×64	
6	64c7p0s1	8×8×64	
7	1a	8×8×64	
8	128c7p0s1	8×8×128	
9	1a	8×8×128	
10	-	128	
11	-	100	$\mu(U^t)$ (ANN)
12	-	100	$\Sigma(U^t)$ (ANN)
13	-	128	ANN Decoder
14	128c4p0s2	4×4×128	
15	64c4p1s2	8×8×64	
16	32c4p1s2	16×16×32	
output	2c4p1s2	32×32×2	Time Surface

Notation: `Ya` represents `YxY` sum pooling, `XcYpZsS` represents `X` convolution filters (`YxY`) with padding Z and stride S.

5.2.6 Encoder SNN Dynamics

To take full advantage of the event-based nature of the DVS input stream and its rich temporal features, the data is encoded using an SNN. SNNs can be formulated as a type of recurrent neural network with binary activation functions (Figure 5.1) [125]. With this formulation, SNN training can be carried out using standard tools of autodifferentiation. We use the same neuron model defined in equation 4.1 described in chapter 4.2.2. The SNN follows a convolutional architecture, as described in Table 5.1. Similar networks were used for classification tasks on the DVSGesture dataset, leading to state-of-the-art accuracy on that task [74, 154]. The SNN encodes the input sequence S_{in}^t into a membrane potential variable U^t in the final layer. The network computes $\mu(U^t)$ and $\Sigma(U^t)$ as in a conventional VAE, but uses the final membrane potential state U^t. Thanks to the chosen neural dynamics, the TS can be naturally computed by our network. In fact, using an appropriate choice of $\tau = \tau_{syn}$ for computing the TS, it becomes exactly equivalent to the pre-synaptic trace Q^t of our network. Hence, our choice of input and target corresponds to an autoencoder in the space of pre-synaptic traces Q^t.

5.2.7 Datasets

We trained and evaluated the model using the NMNIST and IBM DVSGesture datasets, both of which were collected using DVS sensors [98, 138].

In our work we scale each sample to 32x32 and only use a randomly sampled 200ms sequence to match real time learning conditions. For both datasets, to reduce memory requirements, the gradients were truncated to 100 time steps (*i.e.,*, 100ms worth of data). For both datasets, the model learns a latent space encoding that can be used to reconstruct the digits or gestures and to classify instances accurately.

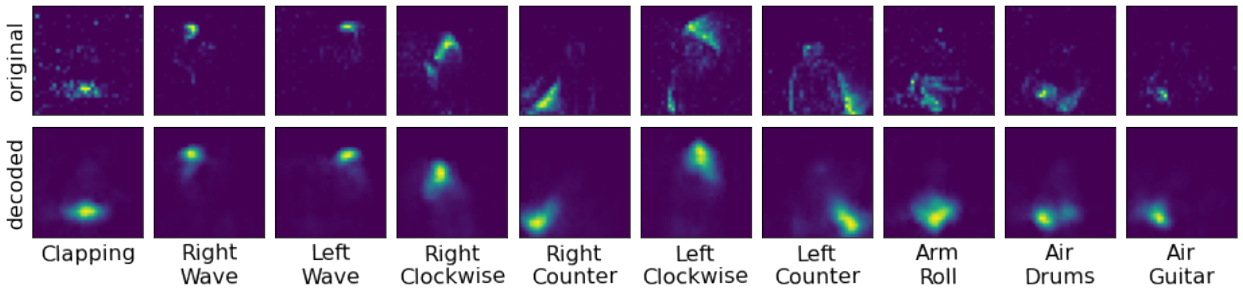


Figure 5.2: Original (top) and reconstructed (bottom) time-surfaces for a sample gesture from each class. The reconstructions reflect the location of each gesture but with some smoothing of the detail.

5.2.8 Neuromorphic Hardware Implementation

As a first step towards an online self-supervised learning system that uses the Hybrid Guided-VAE as a pseudo-labeler, we developed a proof-of-concept implementation of our SNN encoder which can be trained and run on the Intel Loihi. Since only the encoder is required for estimating the feature embeddings, it is not necessary to map the decoder onto the Loihi. We trained the neuromorphic encoder with our Hybrid Guided-VAE method with a couple key differences in order for the encoder to work on Loihi as follows: 1) To train with the same neuron model and quantization as the Loihi we used SLAYER which has a differentiable functional simulator of the Loihi chip [154, 162] allowing for one-to-one mapping of trained networks onto the hardware; and 2) The μ and Σ parts of the network were made spiking and used the quantized membrane potential of the neuron for the latent representation instead of ANN trained full precision values.

5.2.9 System Specifications For Measurement

The training of the hybrid Guided-VAE model and the latent space classifier were performed with Arch Linux 5.6.10, and PyTorch 1.6.0. The machine consists of AMD Ryzen Threadripper CPUs with 64GB RAM and Nvidia GeForce RTX 1080Ti GPUs. For the neuromorphic

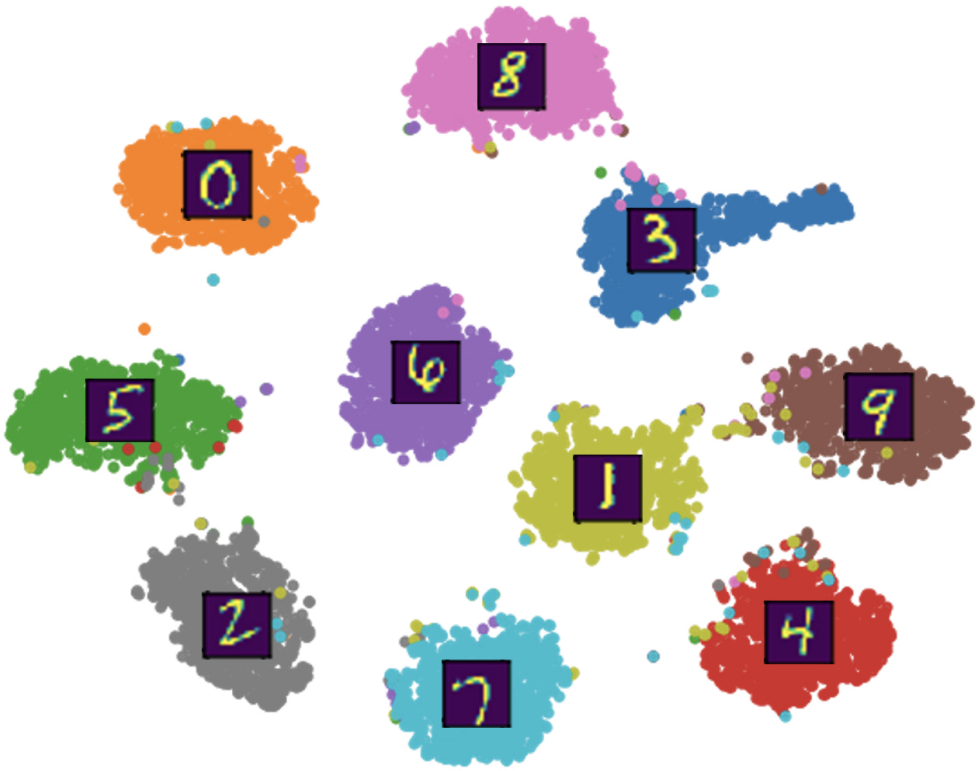


Figure 5.3: A T-SNE plot of the z_m portion of the latent space of the encoded NMNIST dataset. Color of the data points corresponds to the digit classes. Clear separation between classes indicates that the algorithm learns to encode the spiking data into a latent space that strongly emphasizes class-relevant features over other variation.

hardware implementation, an Intel Nahuku 32 board consisting of 32 Loihi chips running on the Intel Neuromorphic Research Community (INRC) cloud were used with Nx SDK 1.0. The machine consists of an Intel Xeon E5-2650 CPU with 4GB RAM.

5.3 Results

5.3.1 NMNIST Accuracy and Latent Space

The use of the NMNIST spiking digit classification data allows us to test the ability of the algorithm to learn to encode spiking input data in a manner that preserves information

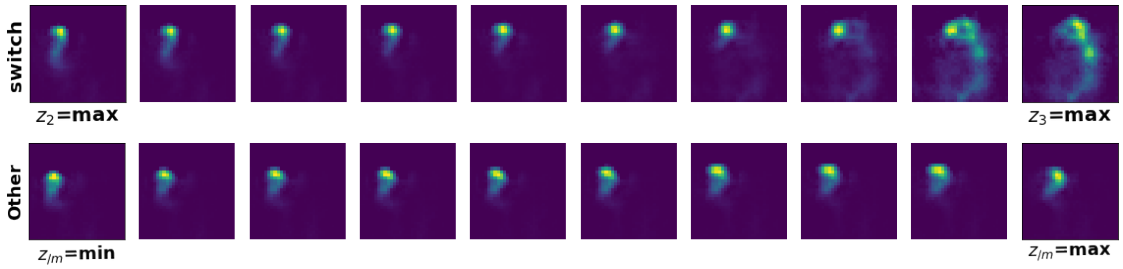


Figure 5.4: Traversals of the latent space learned from the DVSGesture dataset. (Top) Beginning with the right hand wave latent variable maximized and the left hand wave variable minimized, traverse along the latent space by gradually decreasing the right hand wave latent variable and increasing the left hand wave latent variable. Note the initial TS shows a small, focal area of motion in the top-left corresponding to the participant's right hand waving. (Bottom) The latent traversal along all of the non-target $z_{\setminus m}$ latent variables illustrates the relative insensitivity of the model to these features.

needed for accurate classification and captures salient features in the latent space. Trained on the NMNIST dataset, the excitation classifier achieved both training and test accuracy of approximately 99% indicating that the SNN encoder learned a latent representation that clearly disentangles digit classes.

We use T-SNE to visualize the learned representations of the algorithm. T-SNE embeds both the local and global topology of the latent space into a low-dimensional space suitable for visualization [118], allowing us to observe clustering and separation between gesture classes and lighting conditions. As shown in Figure 5.3, each digit class in the NMNIST dataset is clearly separable in the latent space, with only a few data points inaccurately clustered.

5.3.2 DVSGesture Accuracy, Latent Space, and Reconstructions

To analyze the learned representation of gestures in the latent space we examine the accuracy of the excitation classifier in correctly identifying a gesture, the T-SNE projections of the different parts of the latent space, and traversals of the latent space. Finally, we observe the quality of the embeddings based on our own DVS recordings of novel gestures.

The excitation classifier results on the DVSGesture dataset achieves a training accuracy of approximately 97% and test accuracy of approximately 87%. Qualitatively, the SNN encoder learns a disentangled latent representation of features unique to each gesture class but has some difficulty distinguishing between gestures that are very similar.

In Figure 5.2, a sample gesture from each of the gesture classes is visualized as a TS. Colors in the samples correspond to the TS value of the events at the end of the sequence. Note that the TS leaves a significant amount of fine detail intact. In contrast, encoding and then decoding the samples results in a reconstruction that preserves the general structure of the gesture but smooths out some of the detail. Note that, for the purposes of estimating gesture similarity and producing labels from novel data, the fidelity of the reconstruction is less important than the capacity of the model to disentangle the gesture classes. Furthermore, disentangling autoencoders are known to provide lower quality reconstructions compared to unguided VAEs [22].

We use T-SNE to examine the capacity of the network to disentangle salient gesture features in the latent space. Figure 5.5 shows a T-SNE plot of the guided portion of the latent space trained on the DVSGesture dataset. This plot indicates clear clustering of gesture classes, with some overlap between similar gestures such as left arm clockwise and counterclockwise. This global structure in the learned representations indicates the Hybrid Guided-VAE is identifying useful, class-relevant features in the encoder and suppressing noise and unhelpful variability from the spiking sensor.

As an additional tool to investigate the structure of the latent space representations, we use latent traversals to interpret the features of the space. Traversals consist of a set of generated TS based on positions in the space computed using off polarity events. The positions traverse a line in the latent space, revealing how the network encodes salient features.

Figure 5.4 contains two traversals illustrating the shift in position and intensity of motion

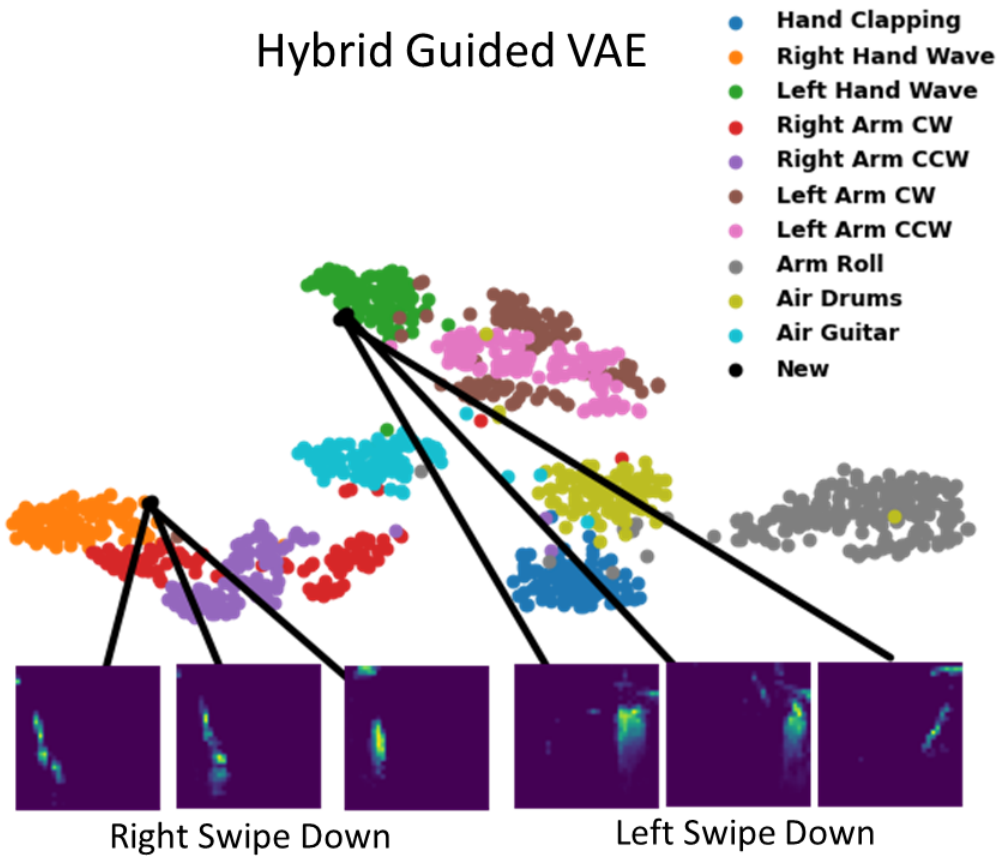


Figure 5.5: A T-SNE plot of the z_m portion of the latent space of the encoded DVSGesture dataset. Additionally, projections of the z_m portion of the latent space of encoded new gestures we recorded using a different DVS, and not part of the DVSGesture dataset are shown. Bottom color plots are the TS of the new gestures.

in the TS encoded by the latent variables z_2 and z_3. Because those features correspond to distinguishing, salient characteristics of the gesture classes "Right Hand Wave" and "Left Hand Wave", this encoding allows the model to disentangle the gestures.

5.3.3 Labeling Unlabeled Gestures

To test the generalization of the learned encoder, we evaluated how the VAE model performs when provided with new gesture data captured in a new environment intended to replicate ecologically valid conditions for a real-world gesture recognition system. We recorded gestures belonging to two new classes, right and left swipe down, which are not present in the

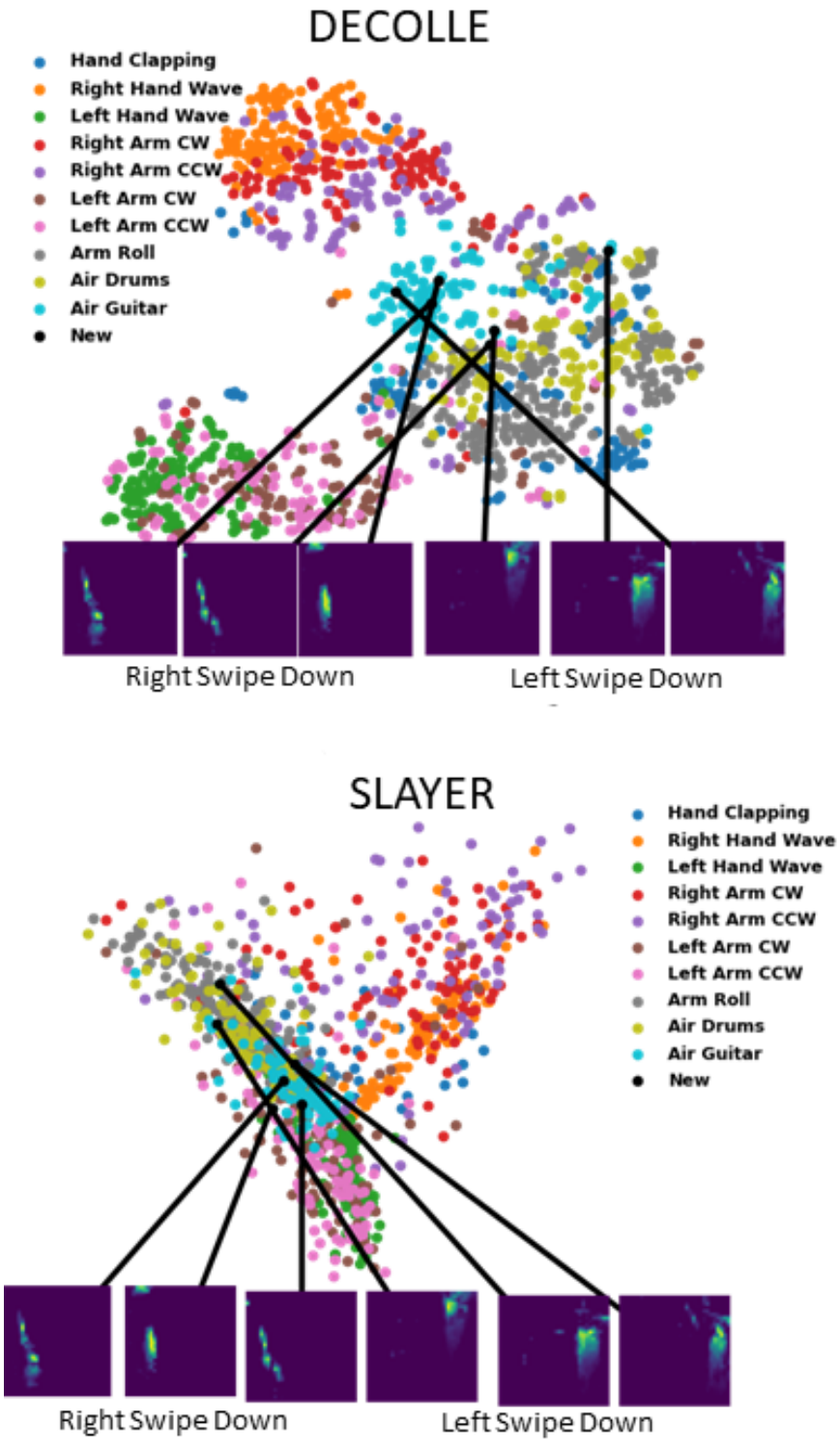


Figure 5.6: A T-SNE plot DVSGesture dataset and real-world gestures using the convolution layer output of SLAYER and DECOLLE models.

DVSGesture dataset. We used a different DVS sensor (the DAVIS 240C sensor [15]) and processed the data with the trained Hybrid Guided-VAE. Each gesture was repeated three times for approximately 3s by the same subject under the same lighting conditions.

Figure 5.5 shows the new gesture TS and associated T-SNE embeddings in the z_m portion of the latent space. The right swipe down gestures were represented by the model similar to right hand wave gestures, as indicated by their proximity in the T-SNE plot of the latent space. Similarly, the left swipe down gestures were represented most similar to left hand wave gestures. Interestingly, both new classes cluster near the edges of the existing classes, possibly indicating the presence of a feature gradient.

We also compare the clustering and ability of the VAE model to pseudo-label novel real-world data to two methods that give state-of-the-art classification accuracy on the DVSGesture dataset, SLAYER [154] and DECOLLE [74]. To compare the models we did a T-SNE visualization of the features learned by the convolutional layers of the SLAYER and DECOLLE models which are shown in Figure 5.6. The features learned by the models do not clearly disentangle the classes. Additionally, when the new gesture classes outside of the DVSGesture dataset are given to the models the are not clustered together near the space of a particular class and instead are identified as being closest to classes such as "Air Guitar" and "Air Drums" instead.

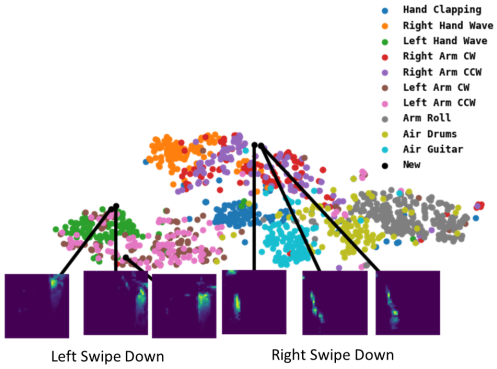
These results demonstrate that the Hybrid Guided-VAE is capable of appropriately representing novel gestures in a manner that support pseudo-labeling. With additional data points of new gestures, the hybrid Guided-VAE can eventually learn new classes of gestures on it own.

Table 5.2: Classification from latent space

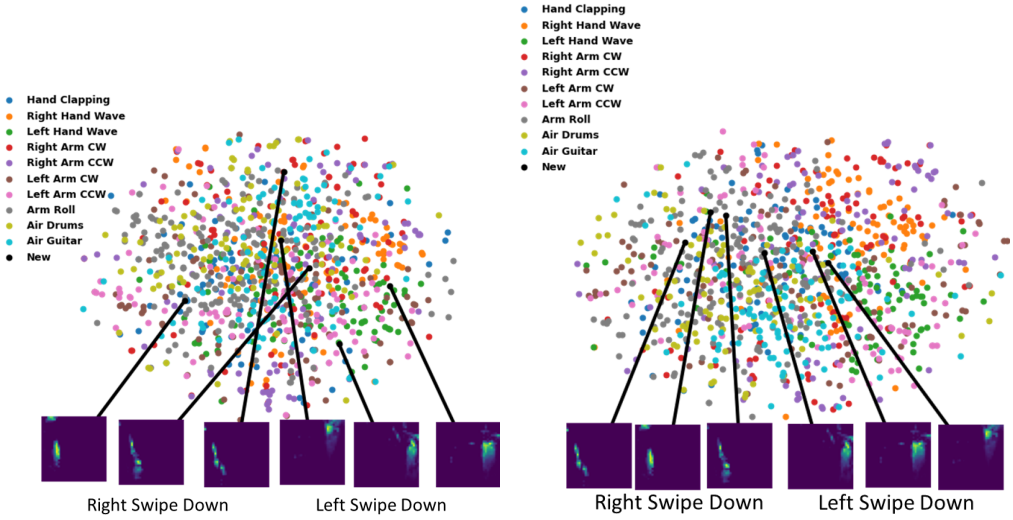
Algorithm	Dataset	Train	Test
Hybrid Guided VAE	DVSGesture	**97.6%**	**86.8%**
	NMNIST	**99.6%**	**98.2%**
CNN Guided VAE	DVSGesture	86.7%	82.3%
	NMNIST	97.2%	96.8%
Hybrid VAE	DVSGesture	99.7%	38.3%
	NMNIST	96.8%	92.4%
CNN VAE	DVSGesture	97.5%	28.4%
	NMNIST	96.8%	91.5%

5.3.4 Ablation Study

We present the results of an ablation study of our Hybrid Guided-VAE method to demonstrate why we used this method for latent space disentanglement. We compare the Hybrid Guided-VAE in our work to a hybrid VAE with the guided part ablated, as well as a Guided-VAE that does not use an SNN encoder and instead use a CNN encoder, and an ordinary CNN VAE. Comparing the clustering of the hybrid VAEs and the CNN VAEs in Figure 5.7, both the CNN and hybrid VAEs are able to disentangle and cluster the latent space when using the guided method, with our hybrid method in Figure 5.5 showing less overlap between clusters and therefore better disentanglement. For classification from the latent space representation of the data, Table 5.2 shows that the CNN and hybrid VAEs both achieve high accuracy on both datasets, with the hybrid VAEs achieving higher performance. Therefore, our hybrid guided VAE approach is more suitable for latent space disentanglement with data taken from event-based sensors. The disentangled latent space shows the hybrid VAEs learn a measure of similarity in event-based data, such as gesture similarity, which demonstrate the effectiveness of the Hybrid Guided VAEs semi-supervised learning from the event data.



(a) CNN Guided VAE



(b) CNN Unguided VAE (c) Hybrid Unguided VAE

Figure 5.7: TSNE plots of the DVSGesture latent space. The images below each figure show novel gestures and where they are placed in the latent space by the different models.

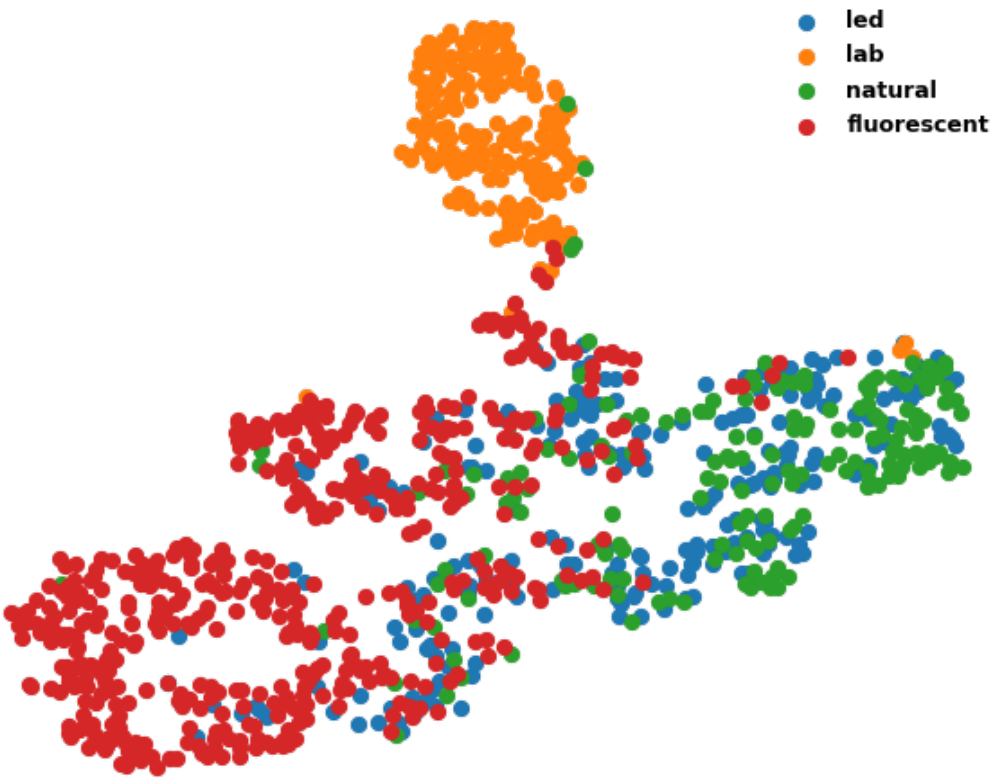


Figure 5.8: A T-SNE plot of the guided z_m latent space using lighting condition labels.

5.3.5 Guiding on Other Factors of Variation: Lighting

A key feature of the guided VAE is to incorporate alternative features not directly related to the gesture class to disentangle the factors of variation in the data. To demonstrate this, in a separate experiment, we trained on the lighting condition provided in the DVSGesture dataset instead of the gesture class. The T-SNE projection of the z_m latent space is shown in Figure 5.8. The model clusters the lighting conditions with some overlap between LED lighting and the other lighting conditions. This is likely due to the fact that the lighting conditions under which the gestures are performed are combinations of the labeled lighting conditions, with the label given to the most prominent lighting condition used [4].

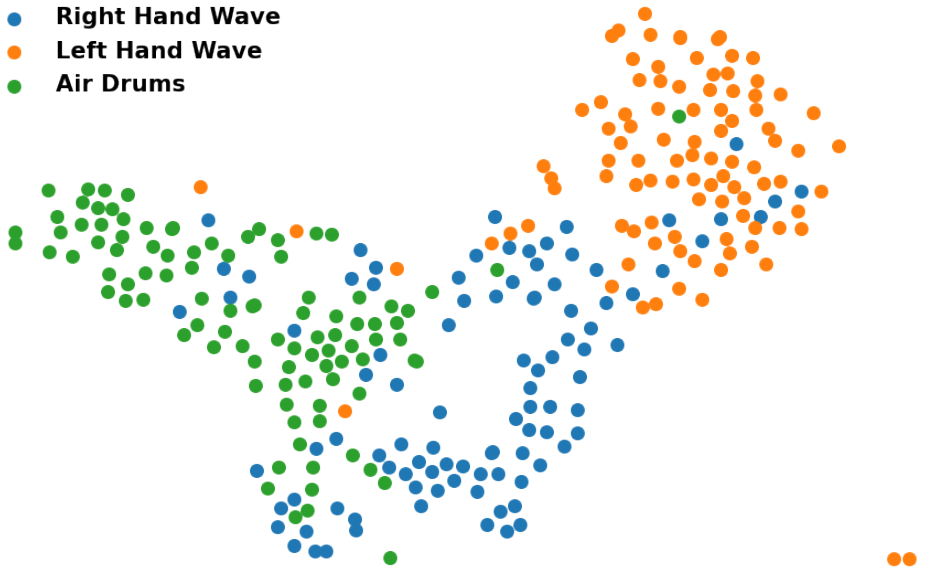


Figure 5.9: T-SNE plot of the z_m portion of the latent space disentanglement of three gesture classes implemented on the Intel Loihi.

5.3.6 Neuromorphic Implementation Results

Figure 5.9 shows the latent space representations of three gesture classes mapped using the neuromorphic encoder. With just three classes, the model running on Loihi demonstrates separation between gestures and global structure, but these features are not as defined as in the conventional model. With all ten classes of gestures used to train the neuromorphic encoder, there was no clear disentanglement or obvious structure to the latent space. It is possible that the lack of separation for the more challenging 10-way task is due to the low-precision integers used for synaptic weights and membrane potentials of spiking neurons on the neuromorphic chip, thus adding variability to the embedded positions.

5.4 Conclusion

We presented a novel algorithm to process DVS sensor data and show that it is capable of learning highly separable, salient features of real-world gestures to automatically generate labels from unlabeled event-based gesture data. The Hybrid Guided-VAE contains an encoder model that learns to represent extremely sparse, high-dimensional visual data captured at the neuromorphic sensor in a small number of latent dimensions. The encoder is jointly trained by two classifiers such that the latent space disentangles and accurately represents target features. The algorithm represents a significant step towards neuromorphic self-supervised learning by enabling the generation of labels from unlabeled data. In addition, we demonstrated a first-of-its-kind implementation of the algorithm on neuromorphic hardware. Due to the sparse nature of event-based data and processing, the SNN encoding implementation offers significant benefits for applications at the edge, including extremely low power usage and the ability to learn on-device to avoid intrusive remote data aggregation. This could enable flexible recognition capabilities to be embedded into home electronics or mobile devices, where computing power is limited and privacy is paramount. While the model performance currently decreases running on the neuromorphic device compared to the conventional GPU-based implementation, improvements to the neuromorphic hardware or the adoption of more powerful learning algorithms, e.g. [163], could alleviate these limitations. These techniques may owe some measure of their success to the computational principles they share with human perceptual systems and we expect that this approach will open new possibilities for interaction between humans and intelligent machines.

Chapter 6

Discussion

The overarching goal of the work presented is to push the field of Artificial Intelligence forward by developing algorithms that can enable rapid online learning to make systems be able to intelligently adapt to new data and scenarios with a combination of neuromorphic computing, deep learning, and local learning. To this end we have focused on learning from one or few samples of data online from streamed data for rapid generalization to new samples of that class of data in a manner similar to biological intelligence. The experimental results indicate that learning offline with gradient based learning methods and then transferring the knowledge learned for deployment in hardware where learning can be done locally online for adaptation to new data is effective for accurate low-power few-shot online learning in neuromorphic hardware. This could enable applications such as learning new gestures performed by someone from one or few examples from data streamed from a DVS to a neuromorphic processor. We demonstrated that using bi-level optimization with a method such as meta-learning is particularly effective for enabling highly accurate few-shot learning within a task domain. We additionally explored how to use unsupervised learning methods such as VAEs to learn feature representations that could be used to pseudo label data for self-supervised learning without being explicitly given a label. Learning without explicitly

given labels would make learning far more efficient because humans would not always need to be in the loop providing labels for data and there are scenarios where they cannot be such as an autonomous robot navigating in a hostile environment.

The work of the thesis has primarily focused on classification problems using data streamed from DVS sensors. However the methods presented in the thesis should be generally applicable to any time-series data such as audio data or biomedical sensor data. For example the MAML meta-learning method applied for few-shot learning in Chapter 4 should be extensible to solve other types of problems such as reinforcement learning [42] and few-shot class incremental learning type problems [23]. While the methods have primarily focused on learning network weights, SNNs and certain types of neuromorphic hardware such as the Intel Loihi have more parameters that could be learned or adapted to solve different problems as well such as learning axonal delays for navigation problems or learning to learn current and membrane potential dynamics which could be useful for SNNs implemented in noisy ReRAM hardware.

The core idea of combining supervised and unsupervised learning methods presented in Chapter 5 of trying to learn representations for classes to learn from unlabeled data by learning a distance measure between prototypical clusters has been expanded upon and adopted in conjunction with other work such as the Prototype-based Open Deep Network (P-ODN) algorithm [155]. Intel has recently developed the Continual Learning Prototypes (CLP) algorithm for their Lava framework [30]. The CLP algorithm uses spiking neurons to explicitly learn the cosine similarity measure between labeled samples, and uses a fixed-point neural network novelty detector to determine of a data sample is novel enough to be considered a new prototype or not. The CLP algorithm has been demonstrated on the COIL-100 dataset, an object image classification dataset. Perhaps the concepts demonstrated by CLP, and the work of Chapters 4 and 5 could be used to extend the algorithm for online learning of event-based sensor or time-series data. This could enable a wide range of

applications that need or could benefit from continuous online learning from unlabeled data such as learning to recognize new accents and words in speech recognition, detecting new anomalies and security threats, and learning new gestures from a diverse group of people and scenarios for Human Computer Interaction (HCI) and Extended Reality (XR) applications.

There is a great deal of work being done by the Neuromorphic Research Community (NRC) related to the work described in this thesis and beyond. While the NRC has been growing in recent years, it is still described as a nascent field that is in an uphill battle to prove its worth against the incredible breakthroughs seen in recent years by traditional computing hardware and the AI/ML algorithms running on them.

Much of what has been discussed in the thesis runs adjacent to the efforts of the TinyML (Tiny Machine Learning) community. Researchers developing in the TinyML space have many of the same goals currently as neuromorphic researchers, in that they are focusing more on small models and solutions that can be deployed at the edge with resource efficient algorithms and hardware [75]. Many of the target applications of the neuromorphic community are shared by the TinyML community such as gesture recognition, speech recognition, biomedical sensor processing, and environmental sensing. The main difference between the neuromorphic and TinyML communities are that the TinyML community is focused on optimizing current algorithms and hardware for Von Neumann architectures while the neuromorphic community is driven by a more bottom up approach to computing trying to rethink how computers and algorithms should work by taking greater inspiration from biology. The TinyML community has demonstrated great success in bringing ML to the edge for applications such as object detection [60], facial recognition [60], and speech recognition [187]. TinyML has even demonstrated edge online learning for anomaly detection with a neural network on a micro controller [145].

On the other end of the spectrum very large ANN models such as the GPT-4 Large Language Model (LLM) have been revolutionary [130]. These models have demonstrated to be highly

capable at natural language processing [17], multi-modal tasks combining vision and language in robotic automation [38], and code writing [21]. The models can use in-context learning (ICL) to be able to learn new classes and concepts from a few examples in context without explicitly updating the model parameters [36]. ICL is often demonstrated with LLMs where the context is given in the form of natural language templates, then ICL concatenates a query question to the template to form a prompt that is input to the LLM for prediction expecting the model to learn from the context given by the templates to make the right prediction. Instead of the explicit gradient update step ICL learns from analogy based on the given context for a training-free learning framework. By not explicitly updating the model parameters problems such as catastrophic forgetting are avoided. Models can be pre-trained to be able to perform ICL at inference time, such as MetaICL [114], and self-supervised ICL [20]. Then at inference time Demonstration Designing algorithms such as Q-Learning [202], and Self-Instruct [184], and Scoring Function algorithms such as Channel Prompt Tuning [113] and kNN-prompting [190] are used to enable ICL during inference. There is a growing interest in the NRC of developing transformer type models for neuromorphic hardware, and some of the work from this thesis such as spiking meta-learning could be extended for ICL and transformers.

The success of both tiny and large ML models run in traditional computing hardware has led even prominent members of the NRC to question the value add of neuromorphic and if the significant investment in the ground up approach of redesigning computing hardware and algorithms will be viable given the competition. Therefore benchmarking applications is currently of utmost importance in the NRC. If neuromorphic cannot prove a significant value add, such as a 100x energy improvement for an exciting application with little to no drop in performance, or provide an exciting application that is simply not possible on traditional hardware then the future of the field will remain uncertain. To reach such a benchmark will take a community wide effort and smart hardware software co-design to go beyond the high bar set by the researchers and industry of more traditional computing and ML. To prove

the value of neuromorphic computing to the wider world an ongoing benchmarking effort is in the works called Neurobench [193]. Neurobench is a community driven effort to create benchmarks for neuromorphic computing algorithms and systems [193]. The current focus of Neurobench is to introduce applications that the community identifies as important for a first round of benchmarking. These benchmarks are: object detection, motor prediction, chaotic function prediction, and few-shot continual learning. The few-shot continual learning benchmarking effort is much in line with the work of the thesis and the author has been one of the leaders in the creation of the benchmark for the task, and is extending the work done in Chapter 4 for meta few shot class incremental learning [23] with SNNs for learning new words from audio data. Intel has also released a benchmarking competition for speech denoising with neuromorphic computing with the target platform being their Intel Loihi 2 processors [173]. There is also a growing interest in bringing the capabilities of LLMs and other transformer based models to neuromorphic hardware with ongoing efforts such as SpikeGPT [203] to develop GPT and other transformer based models compatible with neuromorphic hardware such as the Intel Loihi. Neurorobotics is another area where neuromorphic could shine with applications such as autonomous robots and vehicles [19], and adaptive prosthetic control. These are just a handful of examples of the ongoing efforts by researchers in both academia and industry to try and prove the value of neuromorphic to the wider world to motivate further growth and development and make it a worthy resource efficient addition to traditional computing methods for intelligent computing.

Bibliography

[1] M. Abdollahi and S.-C. Liu. Speaker-independent isolated digit recognition using an aer silicon cochlea. In *2011 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 269–272, Sand Diego, CA, USA, 2011. IEEE.

[2] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015.

[3] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. Di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. Farinha, et al. Equivalentaccuracy accelerated neuralnetwork training using analogue memory. *Nature*, 558(7708):6067, 2018.

[4] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, et al. A low power, fully eventbased gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, page 72437252, 2017.

[5] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, page 39813989, 2016.

[6] J. Anumula, D. Neil, T. Delbruck, and S.-C. Liu. Feature representations for neuromorphic audio spike streams. *Frontiers in Neuroscience*, 12:9–14, 2018.

[7] J. Arthur and K. Boahen. Learning in silicon: Timing is everything. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, USA, 2006.

[8] D. Badoni, M. Giulioni, V. Dante, and P. Del Giudice. An aVLSI recurrent network of spiking neurons with reconfigurable and plastic synapses. In *International Symposium on Circuits and Systems, (ISCAS), 2006*, page 12271230. IEEE, May 2006.

[9] P. Baldi, P. Sadowski, and Z. Lu. Learning in the machine: The symmetries of the deep learning channel. *Neural Networks*, 95:110133, 2017.

[10] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):55955637, 2017.

[11] G. Bellec, F. Scherr, E. Hajek, D. Salaj, R. Legenstein, and W. Maass. Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets. *arXiv preprint arXiv:1901.09049*, 2019.

[12] G. Bi and M. Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.*, 18(24):1046410472, 1998.

[13] Y. Bi, A. Chadha, A. Abbas, , E. Bourtsoulatze, and Y. Andreopoulos. Graph-based object classification for neuromorphic vision sensing. In *2019 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2019.

[14] T. Bohnstingl, S. Woźniak, W. Maass, A. Pantazi, and E. Eleftheriou. Online spatiotemporal learning in deep neural networks. *arXiv preprint arXiv:2007.12723*, 2020.

[15] C. Brandli, R. Berner, M. Yang, S. Liu, and T. Delbruck. A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor. *IEEE Journal of SolidState Circuits*, 49(10):23332341, 2014.

[16] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993.

[17] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

[18] S. Chakrabartty and S.-C. Liu. Exploiting spike-based dynamics in a silicon cochlea for speaker identification. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 513–516, Paris, France, 2010. IEEE.

[19] K. Chen, T. Hwu, H. J. Kashyap, J. L. Krichmar, K. Stewart, J. Xing, and X. Zou. Neurorobots as a means toward neuroethology and explainable ai. *Frontiers in Neurorobotics*, 14, 2020.

[20] M. Chen, J. Du, R. Pasunuru, T. Mihaylov, S. Iyer, V. Stoyanov, and Z. Kozareva. Improving in-context few-shot learning via self-supervised training. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3558–3573, Seattle, United States, July 2022. Association for Computational Linguistics.

[21] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code, 2021.

[22] R. T. Q. Chen, X. Li, R. B. Grosse, and D. K. Duvenaud. Isolating sources of disentanglement in variational autoencoders. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[23] Z. Chi, L. Gu, H. Liu, Y. Wang, Y. Yu, and J. Tang. Metafscil: A meta-learning approach for few-shot class incremental learning. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14146–14155, 2022.

[24] E. Chicca, G. Indiveri, and R. Douglas. An adaptive silicon synapse. In *International Symposium on Circuits and Systems, (ISCAS), 2003*, page I81I84. IEEE, May 2003.

[25] E. Chicca, F. Stefanini, and G. Indiveri. Neuromorphic electronic circuits for building autonomous cognitive systems. *Proceedings of IEEE*, 2013.

[26] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[27] B. Cramer, S. Billaudelle, S. Kanya, A. Leibfried, A. Grübl, V. Karasenko, C. Pehle, K. Schreiber, Y. Stradmann, J. Weis, et al. Training spiking multilayer networks with surrogate gradients on an analog neuromorphic substrate. *arXiv preprint arXiv:2006.07239*, 2020.

[28] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7):1–14, 2020.

[29] J. Dabney, M. Meyer, and S. Paabo. Ancient dna damage. *Cold Spring Harbor Perspectives in Biology*, 5(7), 2013.

[30] R. Danielle. Continual learning prototypes. `https://github.com/lava-nc/lava/blob/main/tutorials/in_depth/clp/tutorial01_one-shot_learning_with_novelty_detection.ipynb`, 2023.

[31] M. Davies. Benchmarks for progress in neuromorphic computing. *Nature Machine Intelligence*, 1(9):386388, 2019.

[32] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, P. Joshi, A. Lines, A. Wild, and H. Wang. Loihi: A neuromorphic manycore processor with onchip learning. *IEEE Micro*, PP(99):11, 2018.

[33] T. Delbruck. Framefree dynamic digital vision. In et al. (Eds.) K. Hotate, editor, *Proc. of the Intl. Symp. on SecureLife Electronics*, volume 1, page 2126. University of Tokyo, 2008.

[34] J. Ding, Z. Yu, Y. Tian, and T. Huang. Optimal annsnn conversion for fast and accurate inference in deep spiking neural networks. *arXiv preprint arXiv:2105.11654*, 2021.

[35] Z. Ding, Y. Xu, W. Xu, G. Parmar, Y. Yang, M. Welling, and Z. Tu. Guided variational autoencoder for disentanglement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[36] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, L. Li, and Z. Sui. A survey on in-context learning, 2023.

[37] T. Dozat. Incorporating nesterov momentum into Adam. In *ICLR Workshop*, 2016.

[38] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence. Palm-e: An embodied multimodal language model, 2023.

[39] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):19972017, 2019.

[40] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu. Training spiking neural networks using lessons from deep learning, 2022.

[41] C. Finn, P. Abbeel, and S. Levine. Modelagnostic metalearning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine LearningVolume 70*, page 11261135. JMLR. org, 2017.

[42] C. Finn and S. Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. In *International Conference on Learning Representations*, 2018.

[43] S. Furber. Largescale neuromorphic computing systems. *Journal of neural engineering*, 13(5):051001, 2016.

[44] S. B. Furber, F. Galluppi, S. Temple, L. Plana, et al. The spinnaker project. *Proceedings of the IEEE*, 102(5):652665, 2014.

[45] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, et al. Eventbased vision: A survey. *arXiv preprint arXiv:1904.08405*, 2019.

[46] G. Gallego, T. Delbruck, G. M. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020.

[47] W. Gerstner and W. Kistler. *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

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

[49] W. Gerstner, M. Lehmann, V. Liakoni, D. Corneil, and J. Brea. Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian threefactor learning rules. *Frontiers in neural circuits*, 12:53, 2018.

[50] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[51] I. Goodfellow, J. PougetAbadie, M. Mirza, B. Xu, D. WardeFarley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, page 26722680, 2014.

[52] A. Graves. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, page 513. Springer, 2012.

[53] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *International Conference on Machine Learning*, 2010.

[54] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[55] D. Gutierrez-Galan, J. Dominguez-Morales, A. Jimenez-Fernandez, A. Linares-Barranco, and G. Jimenez-Moreno. Opennas: Open source neuromorphic auditory sensor hdl code generator for fpga implementations. *Neurocomputing*, 436:35–38, 2021.

[56] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, page 770778, 2016.

[57] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[58] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):15271554, 2006.

[59] G. E. Hinton and J. Sejnowski. Optimal perceptual inference. In *""*, 1983.

[60] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[61] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[62] D. Huh and T. J. Sejnowski. Gradient descent for spiking neural networks. *arXiv preprint arXiv:1706.04698*, 2017.

[63] P. K. Huynh, M. L. Varshika, A. Paul, M. Isik, A. Balaji, and A. Das. Implementing spiking neural networks on neuromorphic architectures: A review, 2022.

[64] T. Hwu, J. Krichmar, and X. Zou. A complete neuromorphic solution to outdoor navigation and path planning. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[65] N. Imam and T. A. Cleland. Rapid online learning and robust recall in a neuromorphic olfactory circuit. *Nature Machine Intelligence*, 2(3):181–191, Mar 2020.

[66] G. Indiveri. A low-power adaptive integrate-and-fire neuron circuit. In *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, volume 4, pages IV–IV, 2003.

[67] G. Indiveri and E. Chicca. A VLSI neuromorphic device for implementing spikebased neural networks. In *Neural Nets WIRN11 Proceedings of the 21st Italian Workshop on Neural Nets*, page 305316, Jun 2011.

[68] G. Indiveri, F. Corradi, and N. Qiao. Neuromorphic architectures for spiking deep neural networks. In *2015 IEEE International Electron Devices Meeting (IEDM)*, page 42. IEEE, 2015.

[69] G. Indiveri, B. LinaresBarranco, T. Hamilton, A. van Schaik, R. EtienneCummings, T. Delbruck, S. Liu, P. Dudek, P. Häfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. Saighi, T. SerranoGotarredona, J. Wijekoon, Y. Wang, and K. Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5:123, 2011.

[70] G. Indiveri and S. Liu. Memory and information processing in neuromorphic systems. *Proceedings of the IEEE*, 103(8):13791397, 2015.

[71] H. Jerison. *Evolution of The Brain and Intelligence*. Elsevier Science, 2012.

[72] R. Jiang, J. Zhang, R. Yan, and H. Tang. Fewshot learning in spiking neural networks by multitimescale optimization. *Neural Computation*, 33(9):24392472, 2021.

[73] J. Kaiser, H. Mostafa, and E. Neftci. Synaptic plasticity for deep continuous local learning, Mar 2019.

[74] J. Kaiser, H. Mostafa, and E. Neftci. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14:424, 2020.

[75] R. Kallimani, K. Pai, P. Raghuwanshi, S. Iyer, and O. L. A. López. Tinyml: Tools, applications, challenges, and future research directions, 2023.

[76] L. E. Kay. *Who wrote the book of life?: A history of the genetic code*. Stanford University Press, 2000.

[77] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie. Quantifying the energy cost of data movement in scientific applications. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*, pages 56–65, 2013.

[78] Y. Kim, Y. Li, H. Park, Y. Venkatesha, and P. Panda. Neural architecture search for spiking neural networks. *arXiv preprint arXiv:2201.10355*, 2022.

[79] Y. Kim and P. Panda. Revisiting batch normalization for training lowlatency deep spiking neural networks from scratch. *Frontiers in neuroscience*, page 1638, 2020.

[80] D. P. Kingma and M. Welling. Autoencoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[81] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. GrabskaBarwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, page 201611835, 2017.

[82] G. Koch, R. Zemel, R. Salakhutdinov, et al. Siamese neural networks for oneshot image recognition. In *ICML deep learning workshop*, volume 2, page 0. Lille, 2015.

[83] L. Kotthoff and J. Vanschoren. *Meta-Learning*, pages 35–61. Springer International Publishing, 2019.

[84] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, page 10971105, 2012.

[85] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman. Hots: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1346–1359, 2017.

[86] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Humanlevel concept learning through probabilistic program induction. *Science*, 350(6266):13321338, 2015.

[87] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

[88] M. Lapan. *Deep Reinforcement Learning HandsOn: Apply modern RL methods, with deep Qnetworks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[89] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of Machine Learning Research*, volume 48, pages 1558–1566, New York, New York, USA, 20–22 Jun 2016. PMLR.

[90] L. B. Y. LeCun and L. Bottou. Large scale online learning. *Advances in neural information processing systems*, 16:217, 2004.

[91] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436444, 2015.

[92] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradientbased learning applied to document recognition. *Proceedings of the IEEE*, 86(11):22782324, 1998.

[93] C. Lee, A. K. Kosta, A. Z. Zhu, K. Chaney, K. Daniilidis, and K. Roy. Spike-flownet: Event-based optical flow estimation with energy-efficient hybrid neural networks, 2020.

[94] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in neuroscience*, 14:119–119, Feb 2020. PMC7059737[pmcid].

[95] C.-H. Li, T. Delbruck, and S.-C. Liu. Real-time speaker identification using the aerear2 event-based silicon cochlea. In *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1159–1162, Seoul, Korea (South), 2012. IEEE.

[96] Y. Li. Deep reinforcement learning: An overview. *CoRR*, abs/1701.07274, 2017.

[97] Y. Li, Y. Dong, D. Zhao, and Y. Zeng. Nomniglot: a largescale neuromorphic dataset for spatiotemporal sparse fewshot learning, 2021.

[98] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128x128 120 dB 15 us latency asynchronous temporal contrast vision sensor. *SolidState Circuits, IEEE Journal of*, 43(2):566576, Feb. 2008.

[99] P. Lichtsteiner, C. Posch, and T. Delbruck. An 128x128 120dB 15μslatency temporal contrast vision sensor. *IEEE J. Solid State Circuits*, 43(2):566576, 2008.

[100] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7, 2016.

[101] A. Litt, C. Eliasmith, F. W. Kroon, S. Weinstein, and P. Thagard. Is the brain a quantum computer? *Cognitive Science*, 30(3):593–603, 2006.

[102] Q. Liu and Y. Wu. *Supervised Learning*, pages 3243–3245. Springer US, Boston, MA, 2012.

[103] S. Liu and T. Delbruck. Neuromorphic sensory systems. *Current Opinion in Neurobiology*, 20(3):288295, 2010.

[104] S.-C. Liu, A. van Schaik, B. A. Minch, and T. Delbruck. Asynchronous binaural spatial audition sensor with $2 \times 64 \times 4$ channel output. *IEEE Transactions on Biomedical Circuits and Systems*, 8(4):453–464, 2014.

[105] M. P. R. Löhr, C. Jarvers, and H. Neumann. Complex neuron dynamics on the ibm truenorth neurosynaptic system. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 113–117, 2020.

[106] M. Mahowald. The silicon retina. *Scientific American*, 264:7682, 1991.

[107] C. Mayr, S. Hoeppner, and S. Furber. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning. *arXiv preprint arXiv:1911.02385*, 2019.

[108] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[109] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, 5:115133, 1943.

[110] C. Mead. *Analog VLSI and Neural Systems*. AddisonWesley, Reading, MA, 1989.

[111] C. Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):162936, 1990.

[112] T. Miconi, K. Stanley, and J. Clune. Differentiable plasticity: training plastic neural networks with backpropagation. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3559–3568. PMLR, 10–15 Jul 2018.

[113] S. Min, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Noisy channel language model prompting for few-shot text classification. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5316–5330, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[114] S. Min, M. Lewis, L. Zettlemoyer, and H. Hajishirzi. MetaICL: Learning to learn in context. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2791–2809, Seattle, United States, July 2022. Association for Computational Linguistics.

[115] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, Mass, 1969.

[116] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Humanlevel control through deep reinforcement learning. *Nature*, 518(7540):529533, 2015.

[117] S. Moran, B. Gaonkar, W. Whitehead, A. Wolk, L. Macyszyn, and S. S. Iyer. Deep learning for medical image segmentation – using the IBM TrueNorth neurosynaptic system. In J. Zhang and P.-H. Chen, editors, *Medical Imaging 2018: Imaging Informatics for Healthcare, Research, and Applications*, volume 10579, page 1057915. International Society for Optics and Photonics, SPIE, 2018.

[118] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan. Clustergan: Latent space clustering in generative adversarial networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4610–4617, Jul. 2019.

[119] L. K. Muller and G. Indiveri. Rounding methods for neural networks with low resolution synaptic weights. *arXiv preprint arXiv:1504.05767*, 2015.

[120] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka. Statistical power modeling of gpu kernels using performance counters. In *International Conference on Green Computing*, pages 115–122, 2010.

[121] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan. Speech recognition using deep neural networks: A systematic review. *IEEE Access*, 7:19143–19165, 2019.

[122] R. A. Nawrocki, R. M. Voyles, and S. E. Shaheen. A mini review of neuromorphic architectures and implementations. *IEEE Transactions on Electron Devices*, 63(10):38193829, 2016.

[123] E. Neftci, C. Augustine, S. Paul, and G. Detorakis. Eventdriven random backpropagation: Enabling neuromorphic deep learning machines. *Frontiers in Neuroscience*, 11:324, Jun 2017.

[124] E. O. Neftci. Data and power efficient intelligence with neuromorphic learning machines. *iScience*, 5:5268, jul 2018.

[125] E. O. Neftci, H. Mostafa, and F. Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradientbased optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):5163, Nov 2019.

[126] C. Neti. Neuromorphic speech processing for noisy environments. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 7, pages 4425–4430 vol.7, Orlando, FL, USA, 1994. IEEE.

[127] A. Nichol, J. Achiam, and J. Schulman. On firstorder metalearning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[128] S. Nitzsche, B. Pachideh, M. Neher, M. Kreutzer, N. Link, L. Theurer, and J. Becker. Comparison of artificial and spiking neural networks for ambient-assisted living. In *2022 Smart Systems Integration (SSI)*, pages 1–6, 2022.

[129] M. Noroozi, A. Vinjimoor, P. Favaro, and H. Pirsiavash. Boosting self-supervised learning via knowledge transfer. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2018.

[130] OpenAI. Gpt-4 technical report, 2023.

[131] G. Orchard, E. P. Frady, D. B. D. Rubin, S. Sanborn, S. B. Shrestha, F. T. Sommer, and M. Davies. Efficient neuromorphic signal processing with loihi 2. *CoRR*, abs/2111.03746, 2021.

[132] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9, nov 2015.

[133] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:13101318, 2013.

[134] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[135] M. Payvand, M. E. Fouda, F. Kurdahi, A. Eltawil, and E. O. Neftci. Errortriggered threefactor learning dynamics for crossbar arrays. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, page 218222, Aug 2020.

[136] E. Perot, P. de Tournemire, D. Nitti, J. Masci, and A. Sironi. Learning to detect objects with a 1 megapixel event camera. *Advances in Neural Information Processing Systems*, 33:1663916652, 2020.

[137] T. Pfeil, T. C. Potjans, S. Schrader, W. Potjans, J. Schemmel, M. Diesmann, and K. Meier. Is a 4bit synaptic weight resolution enough? constraints on enabling spiketiming dependent plasticity in neuromorphic hardware. *Frontiers in Neuroscience*, 6, 2012.

[138] C. Posch, D. Matolin, and R. Wohlgenannt. A qvga 143 db dynamic range framefree pwm image sensor with lossless pixellevel video compression and timedomain cds. *SolidState Circuits, IEEE Journal of*, 46(1):259275, jan. 2011.

[139] B. M. Posey. What is the akida event domain neural processor?, Jun 2023.

[140] M. Prezioso, M. Mahmoodi, F. M. Bayat, H. Nili, H. Kim, A. Vincent, and D. Strukov. Spiketimingdependent plasticity learning of coincidence detection with passively integrated memristive circuits. *Nature communications*, 9(1):18, 2018.

[141] S. Qiao, C. Liu, W. Shen, and A. L. Yuille. Few-shot image recognition by predicting parameters from activations. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[142] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *International Conference on Learning Representations*, 2020.

[143] F. A. Ran, P. D. Hsu, J. Wright, V. Agarwala, D. A. Scott, and F. Zhang. Genome engineering using the crispr-cas9 system. *Nature Protocols*, 8(11):2281–2308, Nov 2013.

[144] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[145] H. Ren, D. Anicic, and T. Runkler. Tinyol: Tinyml with online-learning on microcontrollers, 2021.

[146] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review,*, 65(6):386408, Nov 1958.

[147] B. Rosenfeld, B. Rajendran, and O. Simeone. Fast on-device adaptation for spiking neural networks via online-within-online meta-learning, 2021.

[148] K. Roy, A. Jaiswal, and P. Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, Nov 2019.

[149] B. Rueckauer, I. Lungu, Y. Hu, M. Pfeiffer, and S. Liu. Conversion of continuousvalued deep networks to efficient eventdriven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017.

[150] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[151] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, page 38563866, 2017.

[152] F. Scherr, C. Stöckl, and W. Maass. Oneshot learning with spiking neural networks. *bioRxiv*, 2020.

[153] T. Scott, K. Ridgeway, and M. C. Mozer. Adapted deep embeddings: A synthesis of methods for k-shot inductive transfer learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 76–85. Curran Associates, Inc., 2018.

[154] S. B. Shrestha and G. Orchard. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, page 14121421, 2018.

[155] Y. Shu, Y. Shi, Y. Wang, T. Huang, and Y. Tian. P-odn: Prototype-based open deep network for open set recognition. *Scientific Reports*, 10(1):7146, Apr 2020.

[156] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[157] J. Simpson and E. Weiner. *Oxford English Dictionary*. Oxford University Press, 1989.

[158] N. S. Singh, S. Hariharan, and M. Gupta. Facial recognition using deep learning. In V. Jain, G. Chaudhary, M. C. Taplamacioglu, and M. S. Agarwal, editors, *Advances in Data Sciences, Security and Applications*, pages 375–382, Singapore, 2020. Springer Singapore.

[159] K. Stewart, A. Danielescu, T. Shea, and E. Neftci. Encoding event-based data with a hybrid snn guided variational auto-encoder in neuromorphic hardware. In *Neuro-Inspired Computational Elements Conference*, NICE 2022, page 88–97, New York, NY, USA, 2022. Association for Computing Machinery.

[160] K. Stewart, A. Danielescu, L. Supic, T. Shea, and E. Neftci. Gesture similarity analysis on event data using a hybrid guided variational auto encoder, 2021.

[161] K. Stewart, G. Orchard, S. B. Shrestha, and E. Neftci. On-chip few-shot learning with surrogate gradient descent on a neuromorphic processor. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 223–227, 2020.

[162] K. Stewart, G. Orchard, S. B. Shrestha, and E. Neftci. Online few-shot gesture learning on a neuromorphic processor. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4):512–521, 2020.

[163] K. Stewart, G. Orchard, S. B. Shrestha, and E. Neftci. Online few-shot gesture learning on a neuromorphic processor. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4):512–521, 2020.

[164] K. Stewart, G. Orchard, S. B. Shrestha, and E. Neftci. Online fewshot gesture learning on a neuromorphic processor. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4):512521, Oct 2020.

[165] K. M. Stewart and E. O. Neftci. Meta-learning spiking neural networks with surrogate gradient descent. *Neuromorphic Computing and Engineering*, 2(4):044002, sep 2022.

[166] K. M. Stewart, T. Shea, N. Pacik-Nelson, E. Gallo, and A. Danielescu. Speech2spikes: Efficient audio encoding pipeline for real-time neuromorphic systems. In *Proceedings of the 2023 Annual Neuro-Inspired Computational Elements Conference*, NICE '23, page 71–78, New York, NY, USA, 2023. Association for Computing Machinery.

[167] E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber, and S. Liu. Robustness of spiking deep belief networks to noise and reduced bit precision of neuroinspired hardware platforms. *Frontiers in neuroscience*, 9, 2015.

[168] S.-H. Sun. Multi-digit mnist for few-shot learning, 2019.

[169] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[170] X. Tao, X. Hong, X. Chang, S. Dong, X. Wei, and Y. Gong. Few-shot class-incremental learning, 2020.

[171] C. S. Thakur, J. L. Molin, G. Cauwenberghs, G. Indiveri, K. Kumar, N. Qiao, J. Schemmel, R. Wang, E. Chicca, J. Olson Hasler, J.-s. Seo, S. Yu, Y. Cao, A. van Schaik, and R. Etienne-Cummings. Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in Neuroscience*, 12, 2018.

[172] J. C. Thiele, O. Bichler, and A. Dupret. Spikegrad: An annequivalent computation model for implementing backpropagation with spikes. *arXiv preprint arXiv:1906.00851*, 2019.

[173] J. Timcheck, S. B. Shrestha, D. B. D. Rubin, A. Kupryjanow, G. Orchard, L. Pindor, T. Shea, and M. Davies. The intel neuromorphic dns challenge, 2023.

[174] A. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(1):230, 1937.

[175] A. Turing. Computing machinery and intelligence. *Mind*, 59(236):433460, 1950.

[176] A. van Schaik and S.-C. Liu. Aer ear: a matched silicon cochlea pair with address event representation interface. In *2005 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 4213–4216 Vol. 5, Kobe, Japan, 2005. IEEE.

[177] A. Vanarse, A. Osseiran, A. Rassau, and P. van der Made. A hardware-deployable neuromorphic solution for encoding and classification of electronic nose data. *Sensors*, 19(22), 2019.

[178] V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[179] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. *arXiv preprint arXiv:1606.04080*, 2016.

[180] C. Viroli and G. J. McLachlan. Deep gaussian mixture models, 2017.

[181] J. von Neumann. *The Computer and the Brain*. Yale University Press, New Haven, CT, USA, 1958.

[182] J. von Neumann. First draft of a report on the edvac. *IEEE Ann. Hist. Comput.*, 15(4):2775, 1993.

[183] J. von Neumann and R. Kurzweil. *The Computer and the Brain*. The Silliman Memorial Lectures Series. Yale University Press, 2012.

[184] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2023.

[185] L. Watts, D. Kerns, R. Lyon, and C. Mead. Improved implementation of the silicon cochlea. *IEEE Journal of Solid-State Circuits*, 27(5):692–700, 1992.

[186] L. Wiskott and T. J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715770, 2002.

[187] A. Wong, M. Famouri, M. Pavlova, and S. Surana. Tinyspeech: Attention condensers for deep speech recognition neural networks on edge devices, 2020.

[188] Y. Wu, R. Zhao, J. Zhu, F. Chen, M. Xu, G. Li, S. Song, L. Deng, G. Wang, H. Zheng, et al. Braininspired globallocal learning incorporated with neuromorphic computing. *Nature Communications*, 13(1):114, 2022.

[189] Y. Wu, R. Zhao, J. Zhu, F. Chen, M. Xu, G. Li, S. Song, L. Deng, G. Wang, H. Zheng, J. Pei, Y. Zhang, M. Zhao, and L. Shi. Brain-inspired global-local hybrid learning towards human-like intelligence, 2020.

[190] B. Xu, Q. Wang, Z. Mao, Y. Lyu, Q. She, and Y. Zhang. $k$nn prompting: Beyond-context learning with calibration-free nearest neighbor inference, 2023.

[191] J.-Q. Yang, R. Wang, Y. Ren, J.-Y. Mao, Z.-P. Wang, Y. Zhou, and S.-T. Han. Neuromorphic engineering: From biological to spike-based hardware nervous systems. *Advanced Materials*, 32(52):2003610, 2020.

[192] S. Yang, B. Linares-Barranco, and B. Chen. Heterogeneous ensemble-based spike-driven few-shot online learning. *Frontiers in Neuroscience*, 16, 2022.

[193] J. Yik, S. H. Ahmed, Z. Ahmed, B. Anderson, A. G. Andreou, C. Bartolozzi, A. Basu, D. den Blanken, P. Bogdan, S. Bohte, Y. Bouhadjar, S. Buckley, G. Cauwenberghs, F. Corradi, G. de Croon, A. Danielescu, A. Daram, M. Davies, Y. Demirag, J. Eshraghian, J. Forest, S. Furber, M. Furlong, A. Gilra, G. Indiveri, S. Joshi, V. Karia, L. Khacef, J. C. Knight, L. Kriener, R. Kubendran, D. Kudithipudi, G. Lenz, R. Manohar, C. Mayr, K. Michmizos, D. Muir, E. Neftci, T. Nowotny, F. Ottati, A. Ozcelikkale, N. Pacik-Nelson, P. Panda, S. Pao-Sheng, M. Payvand, C. Pehle, M. A. Petrovici, C. Posch, A. Renner, Y. Sandamirskaya, C. J. Schaefer, A. van Schaik, J. Schemmel, C. Schuman, J. sun Seo, S. Sheik, S. B. Shrestha, M. Sifalakis, A. Sironi, K. Stewart, T. C. Stewart, P. Stratmann, G. Tang, J. Timcheck, M. Verhelst, C. M. Vineyard, B. Vogginger, A. Yousefzadeh, B. Zhou, F. T. Zohora, C. Frenkel, and V. J. Reddi. Neurobench: Advancing neuromorphic computing through collaborative, fair and representative benchmarking, 2023.

[194] B. Yin, F. Corradi, and S. M. Bohté. Effective and efficient computation with multiple-timescale spiking recurrent neural networks, 2020.

[195] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, page 33203328, 2014.

[196] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow twins: Self-supervised learning via redundancy reduction, 2021.

[197] F. Zenke and S. Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):15141541, 2018.

[198] F. Zenke and E. O. Neftci. Braininspired learning on neuromorphic substrates. *Proceedings of the IEEE*, page 116, 2021.

[199] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3987–3995. PMLR, 06–11 Aug 2017.

[200] F. Zenke, B. Poole, and S. Ganguli. Improved multitask learning through synaptic intelligence. *arXiv preprint arXiv:1703.04200*, 2017.

[201] F. Zenke and T. P. Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *BioRxiv*, 2020.

[202] Y. Zhang, S. Feng, and C. Tan. Active example selection for in-context learning, 2022.

[203] R.-J. Zhu, Q. Zhao, G. Li, and J. K. Eshraghian. Spikegpt: Generative pre-trained language model with spiking neural networks, 2023.