2021-06-15

# A Biologically Inspired Supervised Learning Rule for Audio Classification with Spiking Neural Networks

Peterson, Dylan George

UNIVERSITY OF CALGARY

# A Biologically Inspired Supervised Learning Rule for Audio Classification with Spiking Neural Networks

by

Dylan George Peterson

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING

CALGARY, ALBERTA

JUNE, 2021

## Abstract

Audio classification has many practical applications such as noise pollution monitoring, wildlife monitoring, audio surveillance, speech recognition, and more. For many of these applications, deploying classifiers on low powered devices for persistent monitoring is desirable. Artificial neural networks (ANN) have achieved significant success for audio classification tasks. However, it may not always be feasible to deploy current state of the art ANNs to embedded devices due to their memory footprint and power consumption. Biologically inspired neural networks, also known as spiking neural networks (SNN), have been shown to significantly reduce power consumption during inference when compared with equivalent ANNs. They have also been theoretically proven to be more computationally powerful per unit than ANNs. These two properties make SNNs an attractive solution for machine learning tasks on low powered embedded devices, such as at the edge in an Internet of Things (IoT) sensor network. However, SNNs tend to lag behind in performance when compared to ANNs. This is partially because training SNNs is difficult since the standard backpropagation algorithm is not directly applicable due to the non-differentiable spiking nature of SNNs. Encoding data into spike trains compatible with SNNs is also an unresolved question when applying SNNs. This work compares different spike encoding schemes for audio data, and a learning algorithm for multilayer SNNs inspired by biologically plausible learning rules is developed. The proposed learning rule is then successfully applied to simple pattern recognition and audio classification tasks.

## Acknowledgements

I would like to thank Dr. Henry Leung for his support throughout the duration of my studies. Under his supervision I have been given the opportunity to pursue research in an area I am passionate about, and without his guidance this work would not have been possible. His experience, insight, and dedication to research activities has aided and inspired me to improve as both a student and a researcher.

I would like to thank my committee members Dr. David Westwick, Dr. Gias Uddin, and Dr. Xin Wang for taking the time to read and provide feedback for this thesis. Their insights and suggestions for improving this work are greatly appreciated.

I would also like to thank my research group peers, who have provided compelling conversation and alternative perspectives on problems related to both course work and research tasks. Their support could always be counted on, and for that I am grateful to have been a part of this group.

Finally, special thanks are extended to my close family and friends, who have supported me in countless ways over the course of my studies. I would especially like to thank my parents for their unabating support throughout all my academic pursuits.

*Dedicated to the late Peter Dzus*

# Table of Contents

# List of Tables

# List of Figures

# List of Symbols and Abbreviations

AED – Audio Event Detection

ANN – Artificial Neural Network

BPTT – Backpropagation Through Time

CNN – Convolutional Neural Network

CRNN – Convolutional Recurrent Neural Network

DCASE – Detection and Classification of Acoustic Scenes and Events

DPPM – Differential Pulse Position Modulation

FSDD – Free Spoken Digits Dataset

IF – Integrate and Fire

IoT – Internet of Things

LIF – Leaky Integrate and Fire

LSTM – Long Short-Term Memory Network

MLP – Multilayer Perceptron

PCM – Pulse Code Modulation

PDM – Pulse Density Modulation

PPM – Pulse Position Modulation

ReLU – Rectified Linear Unit

RNN – Recurrent Neural Network

RTRL – Real Time Recurrent Learning

RWCP – Real World Computing Partnership Dataset

SGD – Stochastic Gradient Descent

SNN – Spiking Neural Network

SNU – Spiking Neural Unit

SOM – Self-Organizing Map

STDP – Spike-Timing-Dependant Plasticity

STFT – Short-Time Fourier Transform

SVM – Support Vector Machine

# 1 Introduction

Over the last decade, computer perception has increasingly become a part of everyday life. Prominent examples include speech recognition and computer vision algorithms, which can be found pre-packaged on almost every smartphone today in the form of virtual assistants, automatic image processing, and more. Computer perception is also at the heart of more complicated tasks, like in self-driving car applications which heavily rely on such algorithms to navigate unseen environments [1]. Humans naturally excel at perceptual tasks, and the gap between state-of-the-art computer perceptual algorithms and human performance has been narrowing significantly in recent times. This is primarily due to the adoption of deep neural networks, especially for computer vision tasks [2]. Machine hearing has also greatly benefited from similar deep learning networks, enabling robust speech recognition systems [3]. Somewhat less studied than speech recognition, general sound classification research has also seen an increased interest over the last few years. It could be argued that generalized sound classification is a more difficult task than speech recognition, since language models and other semantic information can't be leveraged to help with model performance. Additionally, the range of sounds audible to the human ear can vary much more than the subset of sounds contained in speech alone, especially in the frequency domain. Due to this, general sound classification relies heavily on the quality of the machine hearing model and could be improved by developing machine hearing models which can capture and process acoustic information similarly to the human auditory system.

Due primarily to recent innovations in machine learning and neural networks, audio classification research has begun to accelerate. The primary objective of audio

classification is to automatically assign one or more labels to an audio clip or live audio feed corresponding to acoustic events of interest, sometimes with an additional goal of detecting onset and offset times of the audio event in the case of acoustic event detection (AED). Audio event classification has many practical applications such as audio surveillance, noise pollution monitoring, wildlife monitoring, speech recognition, and more [4][5][6][7]. As can be seen from competitions such as the annual Detection and Classification of Acoustic Scenes and Events (DCASE) challenge, deep learning methods and artificial neural networks (ANN) currently represent the state-of-the-art for general audio classification tasks [8]. Although successful, often times these deep learning networks are very large in terms of their memory footprint and they also tend to consume a significant amount of power [9]. This can be problematic, as many of the practical applications involving audio classification benefit from edge deployment on embedded devices with non-trivial power constraints, such as in distributed Internet of Things (IoT) sensor networks [10][11]. Due to these power constraints, state-of-the-art ANN classifiers may not always be feasible to deploy on such low power devices. This motivates the use of alternative power and memory efficient approaches to audio classification.

Modern ANNs stem from early biologically inspired networks such as the perceptron [12]. Current deep learning networks are relatively removed from their original biological inspiration, however the underlying ideas of integrating weighted inputs and then “activating” after a certain point still prevail. Spiking neural networks (SNN) have been named “the third generation” of artificial neural networks [13]. They are inspired from biological neurons, and they use mathematical models which more

closely resemble the internal dynamics of biological neurons when compared to traditional ANN networks. SNNs have many potential advantages, however fully exploiting these advantages has remained a challenging task. The primary advantage they bring is a significant reduction in power consumption when compared with equivalent modern ANNs [14]. SNNs communicate between neurons in the network using event-based impulses or spikes, so power is only consumed when a spike is emitted. In contrast, ANNs consume power at every node (and at every timestep if operating over time) to perform floating point mathematical operations. Besides power savings, SNNs have also been theoretically shown to have a higher computational capacity than ANNs; so in addition to consuming less power per node, smaller networks could be used to achieve the same performance as an ANN [13]. However, training SNNs has proven to be a challenging task due to the non-differentiable nature of their spiking output function. This prevents well-known ANN training methods such as the backpropagation algorithm from being applied directly [15]. Additionally, neuromorphic hardware designed to run SNNs may not always support algorithms that are commonly used for training ANNs, especially when considering on-chip or online learning [16]. There are many different approaches to learning in SNNs, however so far there has been no single algorithm that has been adopted for training SNNs as backpropagation was for ANNS. Given the theoretical power and computational advantages over ANNs, SNNs present a promising alternative to ANNs for audio classification tasks.

### 1.1 Problem Outline and Research Question

One of the unique challenges in sound classification is the sequential and temporal nature of the data. In contrast to image data where all of the information required for

classification is generally available at the same time, sound classification may require information separated by many timesteps to successfully characterize and classify a sound. To achieve real time online learning and inference, it would be desirable for a classifier to be able to process information over time as it becomes available. State-of-the-art ANN classifiers can achieve this, however they often have large memory requirements and can consume a significant amount of power during inference which makes their deployment to low powered edge devices challenging [9]. An attractive approach to address both of these problems is to use SNNs. However, SNNs generally lag behind in terms of performance when compared to ANNs, despite their theoretical computational potential [17]. This is due to a few different factors. First and foremost, training multilayer SNNs is still an open research question. There are successful learning algorithms for single layer networks, however they are not easily extended to work in multilayer networks [18][19]. There have been a handful of learning rules that have been shown to work for multilayer SNNs, but they are all slightly different and bring their own sets of trade-offs and constraints. Using multi-layered networks has been shown to increase the computational capabilities of both ANNs and SNNs, so further developing algorithms that effectively train multilayer SNNs is desirable [20]. Approximations to backpropagation through time (BPTT) have been applied to train SNNs, however the BPTT algorithm requires storage of intermediate variables from every timestep and propagates error information backwards through time at the end of a learning trial [21][22] . This is thought to be biologically implausible, and it requires additional memory to store such intermediate variables [23]. Encoding real-valued data into spike trains that are compatible with an SNN is also a relatively open question, as is

the interpretation of the outputs of the SNN. Given the desirable properties of SNNs and the current state of SNN learning algorithms, the following research question is posed: How can multi-layer SNNs be applied to audio classification tasks using biologically inspired, temporally local learning rules?

## 1.2 Thesis Statement and Motivation

This thesis will focus on applying SNNs to audio classification problems such as speech and sound recognition. SNNs operate over time implicitly, so they seem to be naturally suited for temporal classification problems. They could also potentially address some of the power constraints that are often imposed in real world audio classification applications. Their major drawback is that training algorithms for SNNs are relatively underdeveloped and their performance is lacking [17]. Spike train encoding of real-valued data and learning algorithms in SNNs need to be further developed before the power savings and potential performance improvements can be fully realized. To this end, this work will first explore different spike encodings and compare their effect on classification performance using a multilayer SNN classifier. Specifically, rate encoding schemes are compared with temporal encoding schemes on both a non-temporal and a temporal dataset. A learning rule is then developed for multilayer SNNs, which tries to maintain some biological plausibility in an effort to reduce the network's memory footprint and power consumption. The proposed learning rule aims to account directly for spike timings within the SNN during training, using values that are available to the network in a given timestep to avoid BPTT. This is achieved using biologically-inspired update rules that could potentially help with deployment to neuromorphic hardware, which won't necessarily support more established algorithms such as BPTT for on-chip

learning [16]. This thesis shows that effective audio classification can be achieved with multilayer spiking neural networks by combining temporal spike encoding schemes with a biologically inspired learning rule which takes advantage of spike timings, modulated by a supervisory error signal.

## 1.3 Thesis Outline

- Chapter 2 reviews the audio classification problem and current state of the art methods for training spiking neural networks. The leaky integrate and fire (LIF) spiking neuron model is introduced, and various established SNN learning rules are reviewed comparing their advantages and disadvantages.

- Chapter 3 introduces and compares different spike encoding schemes. A multi-layer SNN trained via BPTT and an SVM classifier are used to evaluate and compare the effect of different spike encodings on classification performance.

- Chapter 4 introduces the proposed learning algorithm for multilayer SNNs. This learning rule is then demonstrated on some simple toy problems, namely a fixed spike pattern association and a XOR problem which has been modified to work with spiking patterns.

- Chapter 5 shows the proposed learning rule applied to some machine learning datasets. The IRIS dataset is first used to show that the learning rule can work on non-temporal data [24]. Then the Free Spoken Digits Dataset (FSDD) and the RWCP dataset is used to show the proposed algorithm's performance on speech

and non-speech audio data [25][26]. Finally, the UrbanSound8k dataset is used as a challenging audio dataset to test the learning rule [27].

- Chapter 6 concludes this thesis. A summary of this work is presented, and future directions for this research are discussed.

# 2 Background and Literature Review

Training networks of spiking neurons has proven to be a challenging task. There are many different approaches, and each approach has unique tradeoffs to consider when training an SNN. This chapter will review the audio classification problem setup, the operation of SNNs, and finally introduce and compare current approaches to learning in SNNs. The learning methods will be divided into three categories: machine learning approaches, biological approaches, and “hybrid approaches”. The machine learning approaches involve techniques that attempt to train SNNs exclusively for performance, whether they are biologically plausible or not. The biological approaches are training methods that are based on biological models for synaptic plasticity, and they try to maintain biological plausibility for every stage of the learning rule. Finally, the “hybrid” approaches are methods which apply biologically inspired ideas for certain portions of the learning rule, but there are also elements which cannot fully be described by biological mechanisms.

## 2.1 Audio Classification

Audio classification endeavors to assign semantic labels to audio clips which contain sound events of interest. Using supervised machine learning methods, a classifier can be trained to identify such sounds of interest given a sufficiently large and representative labeled dataset. A labeled dataset can be described by data examples with associated ground truth labels, forming dataset $D = \{(s, L)\}$ where $L$ is an integer representing a class label out of $K$ possible classes and $s$ is a vector of sampled digital audio. Ideally, a classifier trained to properly label the provided dataset should also be

able to generalize and correctly classify unseen audio examples belonging to the relevant classes. One view of classification is as a two-step process: a feature extraction step followed by a final decision/classification step which assigns a predicted label to the input data. The feature extraction step serves to transform the input data into a feature space where the various classes can be easily identified by the following classification step. In the case of audio data, it is common to also have a preprocessing step to transform the time domain audio signal into a time-frequency representation [28][29][30]. This is done primarily for dimensionality reduction, however such transformations are often times biologically inspired to mimic human perception or the

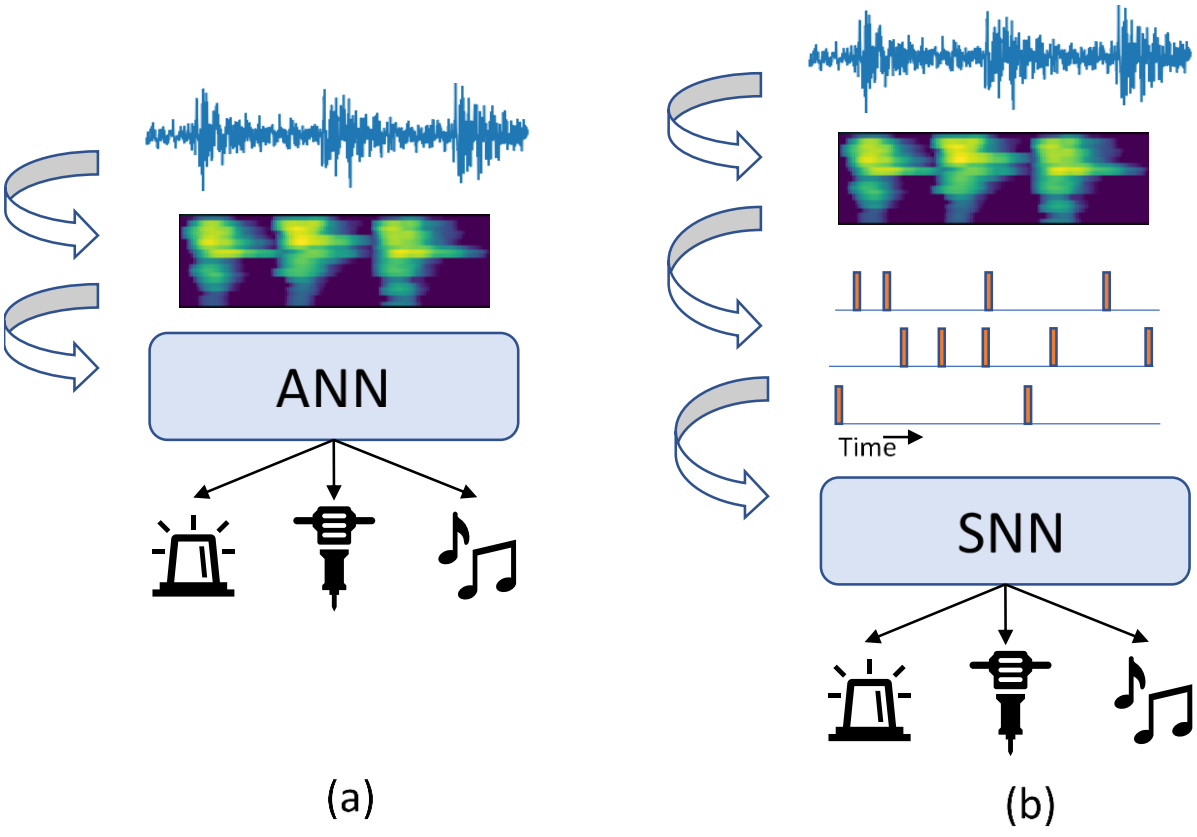



*Figure 2.1: Audio classification pipelines. (a) The audio classification pipeline for ANNS. Audio data is first transformed into a time-frequency representation, then fed into an ANN for classification. (b) The audio classification pipeline for SNNs. An additional spike encoding stage is required for SNNs to operate on the input data.*

frequency decomposition performed by the human cochlea [31][32]. After pre-processing, either an ANN or an SNN can be applied to the pre-processed audio data which performs further feature extraction via learnable transformations before a final output classification. Since SNNs operate on spike trains, there is an additional spike encoding step which will need to be considered alongside the initial preprocessing step, which is the main subject of Chapter 3. Figure 2.1 shows flow charts for standard ANN and SNN approaches to audio classification.

Using ANNs, the classification problem is framed as an estimation problem, with the goal of estimating class labels for each audio clip as a probability [33]:

$$y_k = p(k \mid x;\ \theta) \tag{2.1}$$

Where $y$ is the classification model output vector and each $y_k$ is the element representing the class membership to class $k$ with $k$=1,2,…,$K$. The class probabilities are conditioned on the input to the classification model, $x$, and the classification model is parameterized by θ. A loss function is then defined between the ANN's outputs and the ground truth labels, and θ are optimized to minimize the loss function via stochastic gradient descent (SGD) and the backpropagation algorithm [15]:

$$\Delta\theta = -\eta \nabla_\theta E \tag{2.2}$$

Here, $E$ is the loss function or objective function, $\theta$ are the trainable parameters, and $\eta$ is a learning rate. By training an ANN this way, feature extraction and classification are trained concurrently to minimize the objective function. In practice for classification, the outputs of the ANN are usually set to be softmax values, which forces the output into the range [0,1]. The softmax activation function is described by:

$$y_k = p(k|x;\Theta) = softmax(z_k) = \frac{e^{z_k}}{\sum_{i=1}^{K} e^{z_i}} \tag{2.3}$$

$$\sum_{k=1}^{k} y_k = 1 \ \ and \ y_k \geq 0 \tag{2.4}$$

The inputs to the softmax function are the values of a feature mapping, *z*, of the input data, *x,* parameterized by θ which is to be learned by the ANN. This approach assumes each possible output class is mutually exclusive, so the input data may only contain one class per sample. This is the classification scenario considered in this work, specifically referred to as multi-class, single label classification. The most common loss function applied to this type of classification problem is the cross-entropy loss:

$$E_{softmax}(x, L) = -\sum_{k=1}^{K} q_k \, log(y_k) = -log(y_L) \tag{2.5}$$

Where *q* is the one-hot class distribution of the training example (equals 1 at index k=L, zero otherwise), and $L$ is the ground truth target label index. With the objective function chosen this way, the log likelihood function is maximized by minimizing the cross entropy [34].

In the case of SNNs, the above framework is more difficult to apply directly. The first major hurdle is that the spiking activation of neurons in the network is non-differentiable, which prevents the backpropagation algorithm from being applied directly and in turn makes SGD difficult. Furthermore, applying a soft-max over the network outputs is not always possible since SNNs produce temporal outputs which aren't

always summarized as single values compatible with a softmax function. To achieve a classification from an SNN, there are a few different readout schemes that can be applied. Some works assign labels based on the first output neuron to spike [35]. Others assign labels based on the neuron which emits the most spikes during the pattern presentation; and others assign specific target output spike trains to represent different classes [36][19]. Some methods even use the membrane potential of the output neurons as a real-valued output, ignoring or preventing output spikes altogether [37]. These schemes usually lead to all or none classification, where there isn't an explicit class probability distribution. It is possible to obtain softmax probabilities, for example by applying the softmax function over a vector of the sum of spikes output per neuron during the pattern presentation. However, this approach is only applicable if an approximation to the BPTT algorithm is used, as it requires waiting until the end of the sequence and then assigning temporal credit retroactively. More commonly, a loss function is defined as a summation over the timesteps which can then be decomposed into a per-timestep loss. To this end, the squared error between a target spike train and the output spike train can be calculated at each timestep, and used to update parameter values:

$$E = \sum_{t} \sum_{k} \frac{1}{2} \left( y_k^d(t) - y_k(t) \right)^2 \tag{2.6}$$

Where $y_k^d(t)$ is the value of the target spike train for output neuron $y_k$ at time $t$. Choosing target spike train $y_k^d$ depends on the desired classification method. It can be set to specific spike timings that change depending on the class, or it could be set to fire in every timestep when the corresponding input class is presented and remain silent

otherwise. Defining the loss function this way allows temporally local learning rules to be applied in an attempt to avoid BPTT and achieve online learning. Methods to update the weights in an SNN to minimize such loss functions are discussed later in this chapter.

## 2.2 Spiking Neural Networks

Spiking neural networks are comprised of interconnected stateful nodes where each node models the dynamics of a biological neuron. Similar to ANNs, the specific interconnectivity of the nodes can be chosen by the network designer, and the chosen connectivity is generally referred to as the network's "architecture". Common ANN architectures such as feedforward and recurrently connected networks can be applied to SNNs. A major difference between ANNs and SNNs is that each node in an SNN maintains a state variable, referred to as the neuron's "membrane potential", which is updated over time. Due to this, all inputs and outputs of the network must have a time dimension as do the signals transmitted between nodes. These time domain signals are also generally restricted to have a value of either 0 or 1 which represents the spiking or non-spiking of a signal in a given timestep in discrete time. A popular spiking neuron model is the leaky integrate and fire (LIF) neuron model [38]. This model is widely used in SNN research because it approximates the primary functions of a biological neuron while still being computationally simple for training and running large networks. It is possible to use more accurate biological neuron models, such as the classical Hodgkin–Huxley neuron model; however, these models are much more computationally complex and become restrictive when SNNs increase in size [39]. There are other potential neuron models that could be used in SNNs such as Izhikevich spiking neurons which

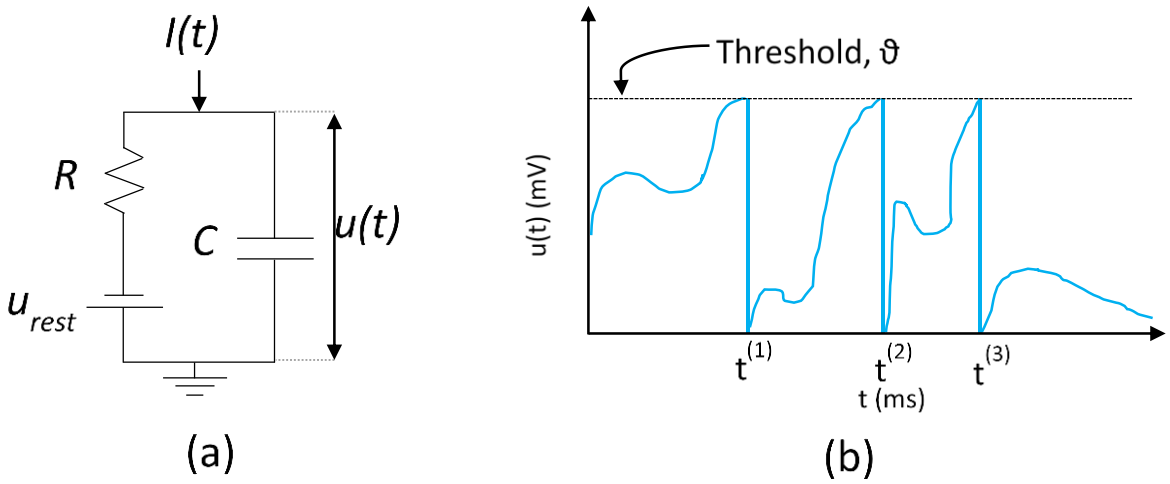



*Figure 2.2: LIF neuron model. (a) The RC circuit model for the LIF neuron. (b) An example membrane potential time course showing the threshold reset behaviour of the LIF neuron.*

are less computationally complex than the Hodgkin–Huxley model while still being more biologically realistic than the LIF neuron model [40]. However, the majority of SNN research has focused on LIF based neurons due to their simplicity, and they will also be the neuron model chosen for this work. The dynamics of the LIF neuron can be modelled as a simple RC circuit shown in Figure 2.2(a) with an additional reset condition. The LIF neuron dynamics can be described by the following equations [38]:

$$\tau_m \frac{du}{dt} = -[u(t) - u_{rest}] + RI(t) \tag{2.7}$$

$$\tau_m = RC \tag{2.8}$$

$$t^{(f)} = \{t | u(t) = \vartheta\} \tag{2.9}$$

$$\lim_{\delta \to 0; \delta > 0} u\left(t^{(f)} + \delta\right) = u_r \tag{2.10}$$

In the above equations, $u(t)$ is the neuron's membrane potential which is modelled as the voltage across a capacitor, $I(t)$ is the input current, and $t^{(f)}$ is the neuron's output firing time. Without any external input current, the membrane potential decays exponentially with time constant $\tau_m$ towards the neuron's resting potential $u_{rest}$. The value of $\vartheta$ determines the neuron's firing threshold. When the membrane potential exceeds $\vartheta$, the neuron emits a spike and the membrane potential is reset to a reset value $u_r$, which is often times set to the neuron's resting potential. An example of this behavior is shown in Figure 2.2(b). A solution to the above differential equation is given by:

$$u(t) = u_{rest} + \sum_{f} (u_r - \vartheta) e^{-\left(\frac{t - t^{(f)}}{\tau_m}\right)} + \frac{R}{\tau_m} \int_0^{\infty} e^{-\frac{s}{\tau_m}} I(t - s)\, ds \qquad (2.11)$$

Where the summation over *f* accounts for the reset dynamics at firing times and the integral accounts for the input current's effect on the membrane potential.

Generally, in the case of SNNs, the input current is abstracted and can be represented by a sum of weighted incoming spikes to the neuron:

$$I(t) = \sum_{i} w_i S_i(t) \qquad (2.12)$$

Where *i* corresponds the $i^{th}$ incoming spike train and $S_i(t)$ is the value of that spike train (either 0 or 1) at time *t*. These incoming spikes are then weighted by their corresponding weight, $w_i$. To illustrate, an example of a neuron with three inputs is shown in Figure 2.3. In SNNs, the weights are treated as trainable parameters. Since the weights are

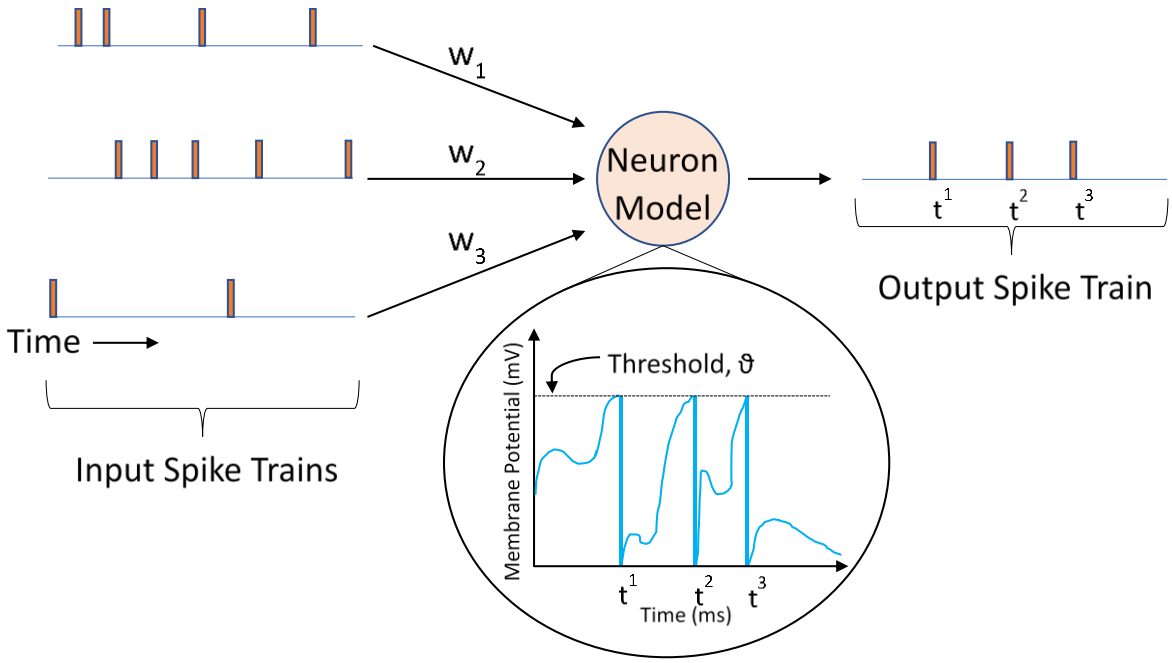



*Figure 2.3: An example of a LIF neuron with three input spike trains.*

trainable, generally the constant terms such as $\frac{R}{\tau_m}$ are absorbed into the weight value. In discrete time, a simple LIF neuron can be expressed recursively as [37]:

$$u_j(t) = u_j(t-1)e^{-\frac{\delta t}{\tau_m}} - S_j(t-1)\vartheta + \sum_i w_{ji}S_i(t) \tag{2.13}$$

$$S_j(t) = H\big(u_j(t) - \vartheta\big) \tag{2.14}$$

Where $\delta t$ is the length of the discrete timestep and *H(·)* is the Heaviside step function. In the above formulation the resting potential is set to 0 and refractoriness isn't explicitly modelled, however in simulations it can be implemented by ignoring input spikes for a given duration after a reset. It should be noted that this formulation isn't the only discrete time formulation for LIF neurons, and there many variations which can allow for different time-courses for the input currents or more precise reset dynamics for

example. The formulation presented here is chosen for its simplicity and ease of implementation in simulations. Given the neuron model, the goal for training an SNN is then to update the weight values to achieve the desired output behavior from the network.

### *2.2.1 Difficulties with Training SNNs*

A major challenge for training SNNs is the non-differentiable nature of their spiking output function. In Equation (2.14) this spiking function is modelled explicitly as a Heaviside step function, which has a derivative of 0 everywhere except at the location where the step from 0 to 1 occurs, in which case it is non-differentiable. Other spiking functions are possible, however in general they all share the same issue of non-differentiability due to the all-or-none spiking behavior of the neuron. This prevents gradient descent via backpropagation from being applied directly to SNNs. Specifically, ignoring temporal dependencies for now, the following term appears from the chain rule in the backpropagation algorithm:

$$\frac{dE(t)}{dw_{ji}} = \frac{dE(t)}{dS_j(t)} \frac{dS_j(t)}{du_j(t)} \frac{du_j(t)}{dw_{ji}} \tag{2.15}$$

Where *E(t)* is the value of the objective function at time *t*, *S(t)* is the output spike train, *u(t)* is the membrane potential and $w_{ji}$ are the trainable weights. The problematic term is $\frac{dS_j(t)}{du_j(t)}$, as it requires taking the derivative of the Heaviside step function. This issue is at the center of most learning algorithms for SNNs, and it motivates alternative approaches to learning than strict backpropagation. Most learning rules for SNNs focus on this term and ways to approximate it or avoid it altogether, and often times these

approximations require specific neuron models or spike encoding schemes. In an effort to overcome the non-differentiable nature of the spiking activation function, learning rules such as SpikeProp were developed by restricting various aspects of the SNN, such as only allowing 1 spike per neuron to occur in the network in all layers [41]. Similarly, the Tempotron is a gradient-based rule for classification that was developed for a single layer SNN which only allowed one output spike per neuron indicating a desired class [18]. Restrictions such as these are undesirable, as it is not clear if these networks could be placed in larger spiking systems or if single spikes are sufficient to capture the necessary information for more complicated tasks.

Since all inputs and outputs in SNNs are modelled as spike trains over time, encoding methods used to convert real valued data into spike trains and the interpretation of the SNN output spike trains also affect learning [42]. Some learning rules that have been developed for SNNs depend on specific spike encoding schemes; for example, ANN-SNN conversion methods tend to depend on rate encodings throughout the network. However, the best encoding scheme may be task dependent. Ideally a learning rule could be applied to SNNs without strictly requiring a specific encoding scheme or neuron model. Spike encoding schemes for audio classification are investigated in Chapter 3 of this thesis.

## 2.3 Machine Learning Based Learning Rules

Currently, the best performing learning rules in SNNs tend to be inspired directly from machine learning approaches. These approaches do not attempt to include any form of biological plausibility, and algorithm complexity is generally not a limiting factor. Due to this, these approaches may not be suitable for deployment to embedded devices

for the same reasons that normal ANNs may not be suitable, especially when considering on-chip learning. Furthermore, these approaches still underperform when compared to equivalent ANNs for the most part, however the difference in performance is smaller when compared with some of the more biologically inspired learning rules. Of the machine learning approaches, there are two major categories: ANN-SNN conversion methods and methods which attempt to apply backpropagation through time directly to SNNs.

### *2.3.1 ANN-SNN Conversion Methods*

The learning methods in this category endeavor to train an ANN for a given task and then apply a method which converts the ANN to an equivalent spiking version of the same network, hoping to maintain the same performance that was initially achieved with the ANN. These approaches map the activations of rectified linear units (ReLU) in ANNS to the firing rate of spiking neurons [43]. Generally, the ANN design is restricted so that the network can be successfully converted into an SNN. In [44], these restrictions included taking the absolute value of the input data and restricting the activation functions in the network to only ReLU. They also set every bias in the network to 0 and replaced max-pooling operations with spatial linear subsampling. Then, a convolutional neural network (CNN) was modified using these restrictions and after training the learned weights were mapped onto an SNN with the same architecture as the CNN. This approach showed good performance on image recognition datasets, with a slight decrease in accuracy when the SNN was compared with the original CNN. Several other approaches improved upon this method [45][14][46]; in all cases, the ANN-SNN conversion depends on the relationship between neuron firing rate and the

ReLU activation function. Due to this, the information is stored and transmitted solely in the firing rate of a neuron as opposed to relative spike timings. This was shown to work well for image data where the static pixel values are converted into an input firing rate, however it is unclear whether this will still work for temporal input data. Furthermore, spike timing has been thought to play an important role in biological neuron computations, so enforcing a strict rate code throughout the network may be limiting [38]. These approaches also require training a full, separate ANN which could prevent them from being able to perform on-chip learning. These methods may not fully exploit the computational power of spiking neurons, and the maximum performance could be limited to the peak performance of the ANN. Due to these limitations it is worthwhile to explore methods to train SNNs directly, especially in the case of audio or general temporal classification where the input data can fluctuate on every timestep.

### *2.3.2 Backpropagation Methods*

In contrast to the ANN-SNN conversion methods, there are also works which attempt to apply backpropagation or backpropagation through time (BPTT) directly to SNNs. Since the activation function for a spiking neuron is non-differentiable, these methods generally employ a pseudo-derivative in place of the non-differentiable term in the backwards pass during BPTT [21][47]. This idea is related to the straight-through estimator presented by Bengio et al. in [48] where the derivative of a binary activation function is replaced with an approximate derivative, such as the identity function or the derivative of the log-sigmoid function. After defining a pseudo-derivative for the non-differentiable terms, these approaches simply apply BPTT as normal. In [21], a spiking neural unit (SNU) was defined using TensorFlow building blocks and was trained using

TensorFlow's built in auto-differentiation framework [49]. The SNU was designed to mimic the LIF neuron, with a few minor restrictions. It achieved high performance on the MNIST, Penn Treebank (PTB), and Johan Sebastian Bach's chorales (JSB) datasets [50][51][52]. It was found that after training via BPTT, the weights learned using the SNU network were able to be mapped onto simulated LIF neurons and achieve similar dynamics. The SNU cell and an example pseudo-derivative is shown in Figure 2.4.This method is promising, since as opposed to the ANN-SNN conversion methods it doesn't require any particular spike encoding/communication scheme, and it seems to perform quite well. However, it also relies on the BPTT algorithm which could be difficult to perform on-chip learning with since it is usually an offline algorithm, and it requires the storage of many intermediate values from every timestep to then perform a

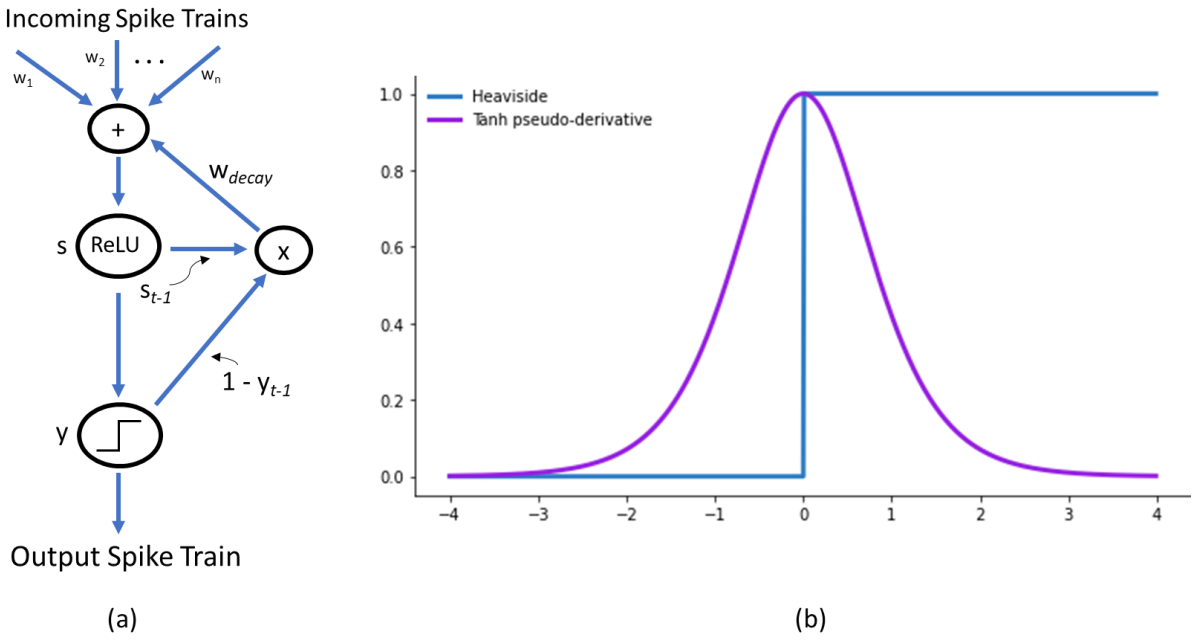



*Figure 2.4: SNU model. (a) The architecture of the SNU cell to be implemented in a deep learning framework such as TensorFlow. (b) An example pseudo-derivative used to train a SNU network which replaces the derivative of the non-differentiable Heaviside function with the derivative of the tanh function during backpropagation. Other pseudo-derivatives are possible.*

backpropagation through time after the end of the input sequence. Due to this method's generality, ease of implementation, and high performance, the SNU network will be used as a benchmark method to compare proposed methods against throughout this thesis.

There have been other approaches to training SNNs where the backpropagation algorithm is modified specifically for spiking neurons, to varying degrees of success and with their own limitations. SpikeProp is one of the earliest approaches to applying backpropagation directly to SNNs [41]. SpikeProp uses backpropagating error signals to adjust weights, and it avoids taking the derivative of the spiking function by making a linear assumption about the membrane potential in a small region around the neuron's firing time. However, this method is limited as it only allows one spike per neuron in all layers. Various works have improved certain aspects of SpikeProp, to varying degrees of performance improvement [53][54][55]. A more recent approach by Lee et al. developed a backpropagation algorithm for spiking neural networks by using the neuron's membrane potential to form a continuous differentiable signal as a surrogate gradient for the spiking function and treating the discontinuities due to resetting/spiking as noise [56]. This method used very precise weight initialization and regularization schemes, and it achieved results on MNIST competitive with state of the art ANNS using a convolutional architecture. All of these methods (and similar methods) make approximations and assumptions specifically to allow the backpropagation algorithm to be applied. These methods tend to be effective, however the assumptions that are made are often not biologically motivated and are computationally complex.

## 2.4 Biologically Inspired Learning Rules

As opposed to the machine learning inspired approaches to learning in SNNs, there are also biologically inspired approaches which make weight updates using biologically plausible processes. Most of the biologically inspired approaches implement some form of spike-timing-dependent plasticity (STDP) . STDP is a model for synaptic plasticity (weight update) based on biological observations [57][58]. There are many different variations of STDP that can be used, however the most common form is one that increases the weight if the pre-synaptic neuron fires shortly before the post-synaptic neuron fires, and it decreases the weight if the post-synaptic neuron fires before the pre-synaptic neuron. A simple STDP model is shown in Figure 2.5, and can be expressed by the following equations [58]:

$$\Delta w = STDP(\Delta t) = A_+ e^{-\left(\frac{\Delta t}{\tau_+}\right)}, \Delta t > 0 \tag{2.16}$$

$$\Delta w = STDP(\Delta t) = A_- e^{-\left(\frac{\Delta t}{\tau_-}\right)}, \Delta t < 0 \tag{2.17}$$

$$\Delta t = t_{post} - t_{pre} \tag{2.18}$$

Where $t_{post}$ and $t_{pre}$ correspond to the post and pre-synaptic neuron firing times. The time constants $\tau_+$ and $\tau_-$ control the length of the learning window where coincident spikes cause a weight update, and $A_+$ / $A_-$ control the sign and amplitude of the STDP function. Note that the exponential learning windows don't need to be symmetrical, and the signs of the weight update can also change allowing for different STDP models. Models similar to the one shown in Figure 2.5 are extremely common however, as they reinforce causality between spikes and can be considered as an extension of Hebb's

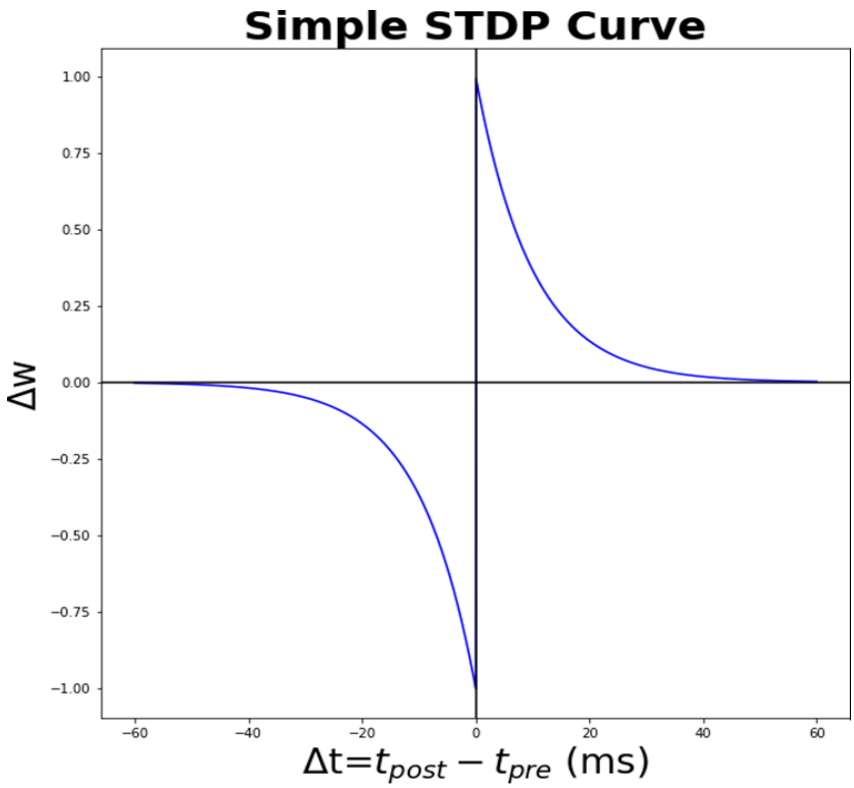



*Figure 2.5: A simple STDP curve.*

postulate [59] which is popularly summarized as “Cells that wire together fire together.” [60]. STDP rules on their own can act as an unsupervised learning rule, where the weight updates are made solely based on pre/post synaptic spike timings without any additional error signals. One issue that arises from using STDP based methods is the so-called “silent neuron” problem where no weight updates can occur if a neuron never outputs any spikes. This is usually addressed via careful weight initialization, or by adding some small non-Hebbian terms to the weight update [61]. In [62], a network of spiking neurons was trained in an unsupervised manner using only STDP to achieve 95% classification accuracy on the MNIST dataset [50]. However, a specific architecture with dedicated inhibitory/excitatory neurons had to be designed, and this approach isn’t necessarily generalizable to arbitrary network architectures.

STDP weight updates can be combined with an external error signal to form so-called “three-factor” learning rules [63]. These rules implement STDP updates local to the synapses within a network, however they allow the magnitude and sign of the weight change to be modulated by a third signal. Biologically, this third signal is thought to represent the effect of neuromodulators such as dopamine on synaptic plasticity. In general, three factor rules are defined using decaying eligibility traces which capture the pre/post synaptic coincidences at each synapse in a network. Three factor learning rules take the following general form [64]:

$$\frac{d}{dt}w_{ji} = e_{ji}M^{3rd}(t) \tag{2.19}$$

$$\frac{\partial}{\partial t}e_{ji} = \eta f_i(x_j)g_j(y_j) - \frac{e_{ji}}{\tau_e} \tag{2.20}$$

Where $e_{ji}$ is the eligibility trace at the synapse from pre-synaptic neuron *i* to post-synaptic neuron *j* which decays with time constant $\tau_e$. The functions $f_i(x_j)$ and $g_j(y_j)$ are some functions of a presynaptic factor $x_j$ and a postsynaptic factor $y_j$, which in the case of STDP could be pre- and post-synaptic spike timings. These functions along with learning rate $\eta$ work to capture correlations between the inputs and outputs of the synapse and store this information in the eligibility trace. In Equation (2.19), the eligibility traces are then modulated by some third-factor signal $M^{3rd}(t)$ which is required to induce a weight change. This third signal can take different forms, such as a global reward signal broadcast to every synapse or a targeted synapse-specific error signal. It was shown in [65] how using a global reward signal to modulate STDP updates in an SNN leads to a form of reinforcement learning. This approach was shown

to successfully solve both a rate coded and a temporally coded XOR problem. In [66], it was shown how reinforcement learning could be achieved using STDP with reward signals using biological mechanisms and constraints. In [67], reward modulated STDP (R-STDP) was used for digit recognition achieving 97.2% accuracy on the MNIST dataset. However, in this network the spiking neurons were limited to one spike per neuron and the reward was only applied to the last few layers, using unsupervised STDP and handcrafted features in the lower layers. A similar approach where R-STDP was only applied to the final layer and unsupervised STDP was applied to the lower layers was used for ECG classification in [68]. In both cases the network architecture was non-arbitrary, and the lower layers were assigned heuristic task-specific functions by the authors.

## 2.5 Hybrid Learning Rules

In this work, “hybrid approaches” will refer to learning rules which attempt to combine machine learning approaches with biologically inspired weight updates or constraints. An example of this type of learning rule is the Multi-ReSuMe learning rule [20], which is an extension of the well-known ReSuMe learning algorithm [61]. This learning rule uses backpropagated errors similar to the regular backpropagation algorithm, except it replaces ill-defined products of spike trains with STDP processes. The algorithm is derived for a feedforward network with an input, hidden, and output layer, indexed by *i, h,* and *o* in the following equations. The multi-ReSuMe update equations are as follows:

$$E(t) = \frac{1}{2}\sum \left(R_o(t) - R_o^d(t)\right)^2 \tag{2.21}$$

$$\Delta w_{ji}(t) = -\eta \frac{\partial E(t)}{\partial w_{ji}} \tag{2.22}$$

Here, the objective function is described in terms of output firing rates $R_o(t)$ and desired output firing rate $R_o^d(t)$. Then, they apply the backpropagation algorithm in terms of firing rates. By doing this, taking the derivative of the spiking function is avoided altogether. These firing rates are later replaced with the actual spike trains as a best-estimate to the instantaneous firing rate. After this substitution, the following expression is obtained for the weight updates from the last hidden to output layer:

$$\Delta w_{oh}(t) = \frac{1}{n_h}[S_o^d(t) - S_o(t)]S_h(t) \tag{2.23}$$

Where $S_o(t)$ is the output spike train and $S_h(t)$ is the hidden layer output spike trains, both composed of Dirac δ functions. The term $n_h$ is the number of neurons in the hidden layer. Since the products of Dirac δ are ill-defined, the authors substitute this operation with and STDP process. After this substitution, the final weight update for the hidden-to-output layer weights is:

$$\Delta w_{oh}(t) = \frac{1}{n_h} S_h(t)[\int_0^\infty a^{pre}[S_o^d(t) - S_o(t)]\, ds]$$

$$+ \frac{1}{n_h}[S_o^d(t) - S_o(t)][\, a + \int_0^\infty a^{post}(s) S_h(t-s)\, ds] \tag{2.24}$$

The $a^{pre}(*)$ and $a^{post}(*)$ functions correspond to STDP functions similar to (2.16) and (2.17) respectively, and s corresponds to $\Delta t$. The constant $a$ term is a non-Hebbian term

added to guarantee a weight change in the correct direction. After continuing their derivation and substituting products of spike trains with STDP processes, the following equation was obtained for the input-to-hidden layer weight updates:

$$\Delta w_{hi}(t) = \frac{1}{n_i n_h} S_i(t) \sum_{o \in O} \left[ \int_0^\infty a^{pre} [S_o^d(t) - S_o(t)] \right] w_{oh} + \frac{1}{n_i n_h} \sum_{o \in O} [S_o^d(t) - S_o(t)] \left[ a + \int_0^\infty a^{post}(s) S_i(t-s) ds \right] w_{oh} \quad (2.25)$$

It is noted by the authors in this approach that the STDP process for input-to-hidden layer weights is between input layer spikes and the output layer's spikes; this is generally thought to be biologically implausible since the output layer's spikes aren't thought to be available to layers which aren't directly connected. This also means that the firing/ non-firing of hidden layer neurons are not considered when making weight updates in the first layer. However, this algorithm is advantageous as it can be performed online without storing state values from every timestep as in BPTT, and it shows how weight updates can be implemented using biologically inspired rules in a supervised learning scenario. The authors stated that the normalization by the number of input neurons ($n_i$ and $n_h$) helped with learning stability and allowed for a single learning rate to be used for all layers. This algorithm was tested on a XOR problem, the IRIS dataset, and some simple spike pattern recognition problems.

Another approach which combines error backpropagation with STDP-like rules is the BP-STDP algorithm [36]. This algorithm is derived using integrate and fire (IF) neurons where the neuron is a perfect integrator and the membrane potential doesn't

decay over time. It is shown that the membrane potential of the IF neuron approximates the activation of a ReLU in an ANN, and then a gradient descent algorithm is derived using this fact to overcome the non-differentiable spiking function. The BP-STDP weight update equations are:

$$E(t) = \frac{1}{2}(S_o^d(t) - S_o(t))^2 \tag{2.26}$$

$$\Delta w_{ji} \propto \frac{\partial E}{\partial w_{ji}} \tag{2.27}$$

$$\Delta w_{oh}(t) = \eta \cdot \xi_o(t) \cdot \sum_{t'=t-\in}^{t} S_h(t') \tag{2.28}$$

$$\xi_o(t) = \begin{cases} 1, S_o^d(t) = 1, S_o \neq 1 \; in \; [t-\in, t] \\ -1, S_o^d(t) = 0, S_o = 1 \; in \; [t-\in, t] \\ 0, otherwise \end{cases} \tag{2.29}$$

$$\Delta w_{hi}(t) = \begin{cases} \eta \cdot \sum_o \xi_0(t) w_{oh}(t) \cdot \sum_{t'=t-\in}^{t} S_i(t') \;\; , \;\; S_h(t) = 1 \; in \; [t-\in, t] \\ 0 \;\; , \;\; otherwise \end{cases} \tag{2.30}$$

Here, $\in$ is a small window before time t where coincident spikes can cause a weight update (similar to the learning window in STDP). The derivative used for the spike activations during the backwards pass is set to 1 if the hidden neuron spiked recently within a short interval, and 0 otherwise. This followed from their comparison between IF neurons and the ReLU function. Then, the input values are set to either 0 or 1 in a similar fashion depending on if they have spiked in a short time window. This causes an

STDP-like interaction where the weight update now depends on both the incoming spike timings and the output spike timings. The error signal, $\xi_o(t)$, causes a weight increase (potentiation) if the corresponding output neuron didn't spike when the target was 1, and it causes a weight decrease (depression) when the output neuron spikes if the target was 0. If the output neuron spikes follow the target signal, the update will be 0. Compared to the Multi-ReSuMe rule, the BP-STDP rule gates the backpropagating error signal by the activations of the hidden layer neurons. However, this rule was implemented for IF neurons whereas the Multi-ReSuMe algorithm was derived for the more general LIF neurons. Furthermore, the BP-STDP algorithm exclusively used rate encoded spiking schemes whereas the Multi-ReSuMe learning rule was shown to work for both rate and temporal spike encoding schemes. This learning rule was shown to work well for a multilayer SNN on a XOR problem, the IRIS dataset, and the MNIST dataset.

In the e-prop learning rule, a biologically plausible online alternative to BPTT is derived [37] for a single recurrently connected layer of spiking neurons with a feedforward readout layer. They show mathematically that the chain of derivatives through time can be approximated by decaying eligibility traces of spike activations and use those values to update the weights online instead of backpropagating through time as in BPTT. To overcome the non-existent derivative of the spiking function, they use a heuristic voltage-dependent pseudo-derivative to replace the spiking function derivative in the backwards pass. This approach is attractive since it can operate fully online and uses quantities (besides the modulating error signal) available locally at a synapse to update synaptic weights. Furthermore, all of the required quantities are available in a

given timestep without needing to store values from previous timesteps. In terms of biological plausibility, the use of synaptic eligibility traces to track information regarding pre/post spike time firing is very closely related to STDP and three-factor synaptic plasticity rules [64]. The e-prop learning rule is defined for a single layer of recurrently connected LIF neurons by the following equations (bias values and updates omitted for brevity):

$$y_k(t) = \sigma y_k(t-1) + \sum_j w_{kj}^{out} S_j(t) \tag{2.31}$$

$$E = \frac{1}{2}\sum_{t,k}\left( y_k(t) - y_k^d(t)\right)^2 \tag{2.32}$$

$$\Delta w_{kj}^{out} = -\eta \frac{dE}{dw_{kj}^{out}} = -\eta \sum_t \left( y_k(t) - y_k^d(t)\right) \bar{S}_j(t) \tag{2.33}$$

$$\Delta w_{ji} = -\eta \frac{dE}{dw_{ji}} \tag{2.34}$$

$$\frac{dE}{dw_{ji}} = \sum_t L_j(t) \cdot \overline{e}_{ji}(t)$$
$$where\ i,j = pre\ and\ post\ synaptic\ indices \tag{2.35}$$

$$L_j(t) = \sum_k w_{kj}^{out} \left( y_k(t) - y_k^d(t)\right) \tag{2.36}$$

$$\psi_j(t) = \frac{\partial S_j(t)}{\partial u_j(t)} = \frac{1}{\vartheta}\gamma_{\rho d}\, max\left(0{,}1 - \left|\frac{u_j(t) - \vartheta}{\vartheta}\right|\right) \tag{2.37}$$

$$e_{ji}(t) = \psi_j(t) \cdot \bar{S}_i(t) \tag{2.38}$$

As can be seen in equation (2.31), the membrane potential of the output neurons (with membrane decay constant $\sigma$) is used as a real-valued network output, as opposed to using output spikes directly. This non-recurrent readout layer is connected to each node in a recurrent layer via weights $w_{kj}^{out}$. The connections from network inputs and connections within the recurrent layer are indexed generically by *i* and *j,* corresponding to a given pre- and post-synaptic neuron. In this approach, equation (2.37) is the pseudo-derivative chosen by the authors which is based on the neuron's membrane potential, where $\gamma_{\rho d}$ is a constant. The author's notation indicates a low-pass filtered version of a given signal by the bar above it: e.g., $\bar{x}(t)$. In equation (2.38), $\bar{S}_i$(t) is a low-pass filtered pre-synaptic spike from neuron *i*. The main idea behind this algorithm is to apply temporally local weight updates by capturing the temporal derivative information in eligibility traces $e_{ji}(t)$, and multiplying by an error signal $L_j(t)$. By utilizing eligibility traces in this way, BPTT can be avoided and temporal credit is assigned via the decaying eligibility traces. This approach is significant in that it shows how temporal dependencies can successfully be captured online using eligibility traces which are local to the neurons in the network, and a good approximation to gradient descent can be achieved in a network of LIF neurons by combining said traces with a backpropagated supervisory error signal.

The approaches in this section share many similarities in implementation and goals to the real time recurrent learning (RTRL) algorithm, which is an algorithm for recurrent ANNs that attempts to convert BPTT to an online algorithm that updates

weights using values which are fed forward through time [22]. All three of these learning rules aim to merge supervisory backpropagated error signals with biologically inspired weight updates. By doing this, they share the advantages of both the machine learning inspired approaches as well as the biologically inspired methods. These approaches are the most closely related to the method proposed in chapter 4 of this thesis.

## 2.6 Looking at Issues with BPTT Applied to SNNs

BPTT is the most widespread algorithm to train recurrent neural networks (RNN). Although it is extremely effective for training RNNs, this algorithm still has some drawbacks which are exacerbated when switching to SNNs. It accumulates and stores the state values from units in the neural network at each timestep, then propagates error backwards through time after the last time step of the input pattern [34]. It is a highly effective algorithm however it is relatively complex and requires the storage of many intermediate variables. It also waits until after the last time step to apply weight changes, which means it is generally an offline algorithm and thought to be biologically implausible [23]. Furthermore, it already suffers from vanishing/exploding gradient problems due to unrolling the computational graph arbitrarily far back through time (LSTMs address this to some extent) [69]. Since spiking neurons use a non-differentiable activation function, these issues could be compounded when using non-exact function derivatives or pseudo-derivatives to approximate the spiking function derivative, which is fairly standard among supervised SNN learning algorithms. For these reasons, this work will focus on temporally local update rules to avoid BPTT to improve both computational efficiency and biological plausibility.

## 2.7 Looking at Issues with SNN Learning Algorithms

Supervised learning algorithms used to train SNNs directly generally make use of some form of approximated backpropagation. The more successful approaches use a pseudo-derivative to approximate the gradient of the non-differentiable spike activation function and then apply the standard backpropagation algorithm [47][56]. However, the error signal is often propagated independently of the actual spiking/non spiking of the neurons in the network. By doing this, the error signal that arrives at the lower layers represents the error as if each neuron in the upper layers had contributed to the output error in a given timestep. Since not every single neuron spikes in every timestep, it may be incorrect to allow non-spiking neurons to be involved in assigning error credit. Some algorithms partially address this by using a voltage-based pseudo-derivative which increases in magnitude the closer a neuron's membrane potential is to the firing threshold. This does link the error propagation to the neuron's spiking activity; however, these pseudo-derivatives still allow large error signals to be propagated by neurons that haven't spiked recently and therefore couldn't have contributed to the current output error. Using an example to illustrate, here is the pseudo-derivative used in the e-prop learning rule again [37]:

$$\frac{\partial S_j(t)}{\partial u_j(t)} = \frac{1}{\vartheta}\gamma_{\rho d}\, max\left(0{,}1 - \left|\frac{u_j(t) - \vartheta}{\vartheta}\right|\right) \tag{2.37}$$

The above equation for the pseudo-derivative reveals how the error signal is proportional to the membrane potential's proximity to the error threshold. Such voltage-based pseudo-derivatives do not make the distinction of whether the neuron has actually fired or not. So the error signals are gated based on the membrane potential's

proximity to the firing threshold rather than the actual output spikes which directly contribute to the neurons in the next layer. Therefore, it is proposed that output errors should only be credited to neurons that have spiked recently and directly contributed to the output error. The BP-STPD learning rule does propagate the error signal based on the recent firing/non firing of neurons, however it was derived specifically for IF neurons using rate encodings. To address these issues, a learning rule is proposed in Chapter 4 of this thesis which aims to adapt the BP-STDP rule to LIF neurons. This is achieved by using eligibility traces to update the weight values similarly to the e-prop rule, however the voltage-based pseudo-derivative is also replaced by an eligibility trace value. This causes the error signal to be gated by the spiking/non spiking of hidden layer neurons as in BP-STDP and also causes an STDP-like interaction for weight updates.

# 3 Spike Encoding

A unique challenge posed by SNNs is the requirement for signals to be encoded as temporal spike events. Real-valued datasets must first be converted into time-domain spike trains before they can be used as inputs to an SNN. Neural encoding is a very active field of research in neuroscience, and it can be difficult to explain biologically observed spike encoding observations with a single encoding model [38]. That being said, there are two prominent models for neural encoding: rate-based encoding, and temporal encoding. Rate-based encodings suggest that information is encoded solely in a neuron's output firing rate. However, there is evidence that suggests that this cannot be the only way information is encoded in the brain, since low firing rates would require long latencies before the rate could be identified but humans are able to process information at faster timescales than could be achieved with only a rate code [70]. This constraint motivates the idea of temporal encoding, where the information is thought to be stored and transmitted in the inter-spike intervals of a single neuron or in the relative spike timings of a population of neurons [71]. A related temporal code is phase-coding, where the information is stored in the spike's phase relative to a global oscillatory signal [72]. In reality, the true neural code is most likely to be some combination of the above schemes, and the difference between rate and temporal codes can break down under certain conditions which suggest they may be special cases of each other [73].

For the purposes of this thesis, different encoding schemes will be tested and the chosen scheme will be selected based on classification performance. To test the encoding schemes, a two-layer SNU network (i.e. a network with an input, hidden, and output layer) with 300 hidden neurons trained via BPTT will be used. The number of

neurons in the final layer will be set equal to the number of classes in a dataset, where each neuron is responsible for detecting one of the given classes. The SNU network will be trained up to 100 epochs using the Adam optimizer [74]. Classification will be based on which neuron outputs the most spikes over the duration of the input pattern, and the loss function will be categorical cross entropy using a softmax over the spike counts at the end of the pattern presentation. To test the spike encoding schemes, the IRIS flower dataset and the Free Spoken Digits Dataset (FSDD) will be used [25][24].

The IRIS flower dataset is a relatively small dataset consisting of three classes of IRIS flower. For each measured flower, four measurements were taken: sepal length and width, and petal length and width. There are 50 sets of measurements for each of the three classes of flower, giving 150 total samples. Many of the encoding schemes considered in this chapter require normalized data; in these cases, the four length measurements are normalized to the range [0,1] using min-max scaling per-feature across the entire dataset.

The FSDD is a publicly available open audio dataset consisting of six different speakers saying the digits 0-9 fifty times each. Since this an open dataset, it may grow in the future as more users add recordings; the version used for this research is v1.0.10. The audio recordings in this dataset are one second long sampled at 8kHz, and there are a total of 3000 recordings. For the experiments in this chapter, the training-test split outlined by the authors of the dataset will be used which sets 90% of the data for training and 10% for testing. Audio preprocessing for this data is described later in this chapter.

The FSDD and IRIS dataset were chosen for their simplicity as well as to test the differences in encoding schemes when the input data is initially non-temporal versus naturally temporal data. A review of temporal spike encoding schemes was presented in [42] which used a support vector machine (SVM) and a tempotron spiking classifier to test latency, phase, population, and threshold encoding schemes [18]. In this chapter some of those encoding schemes will be tested again, however the multilayer SNU classifier will be used in place of the single layer tempotron classifier to test the effects of the different encoding schemes in a multilayer spiking network. A linear SVM classifier will also be used as a baseline for comparison, similar to the approach in [42]. The SVM is chosen for comparison since it works well with high dimensional data, and it is a good representation for how linear classifiers perform under the tested encoding schemes. Since SVMs don't process their inputs temporally, the vectors from each timestep of the spike-encoded data will be concatenated to form one large input vector. This should be noted when comparing the results of this chapter, since the SNU approach processes the data temporally whereas the SVM approach requires a fixed input size and doesn't specifically account for the temporal component of the encoded input data. Some rate-based encoding schemes and a pulse density modulation (PDM) scheme will also be tested which were absent in [42]. Additionally, results for a non-temporal dataset (the IRIS dataset) will be presented to highlight the difference in encoding requirements when moving to a temporal dataset. For the IRIS dataset, the average validation accuracy from a 5-fold cross validation are reported. For the FSDD, the reported results are the peak validation accuracy achieved on the test split defined by the dataset authors, where the train/test split is 90%/10%.

## 3.1 Audio Preprocessing

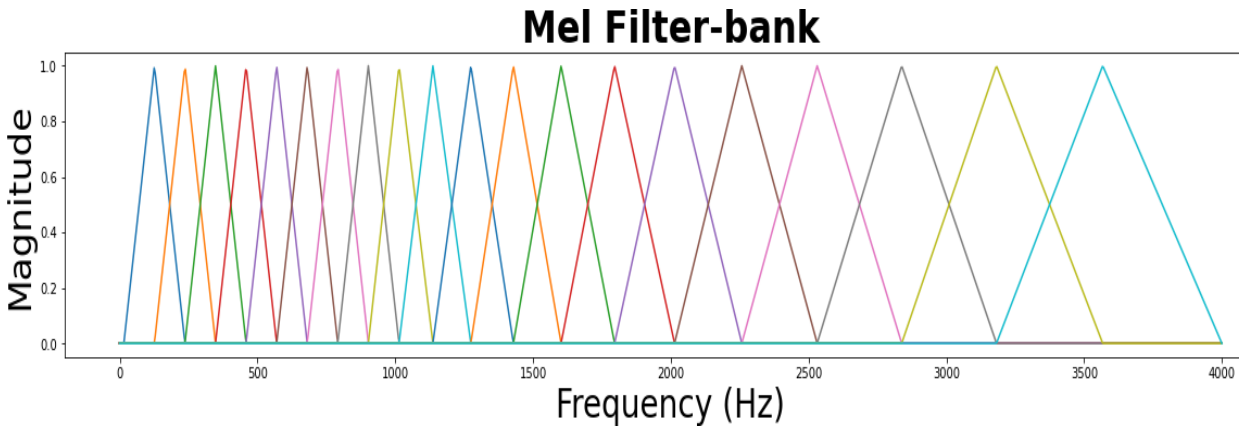



*Figure 3.1: A Mel Filter-bank. 20 Mel-spaced filters are shown in the frequency domain.*

For the FSDD and the audio datasets in Chapter 5, a preprocessing step is performed before the spike encoding method is applied. Instead of operating directly on the raw audio sequence, log-Mel spectrograms are computed as a preliminary feature extraction step. This serves two purposes: first, it is a biologically inspired approximation to the frequency decomposition performed by the human cochlea [31]. Secondly, this preprocessing step helps reduce the time dimension significantly by summarizing the log power for the given frequency bands over a short period of time. The Mel-spaced filter bank approximates the frequency tuning of the inner ear, where there are many narrow filters close together at lower frequencies, and they increase in center frequency spacing and bandwidth as the frequency increases. The Mel filter bank is similar to constant-Q filters banks and gammatone filter banks, however the Mel scale is based on human perception of pitch [28][75]. An example Mel filter-bank with 20 filters between 0 and 4kHz is shown in Figure 3.1. For these experiments, the audio is divided into 20ms frames with 50% overlap, then a short-time Fourier transform (STFT) is

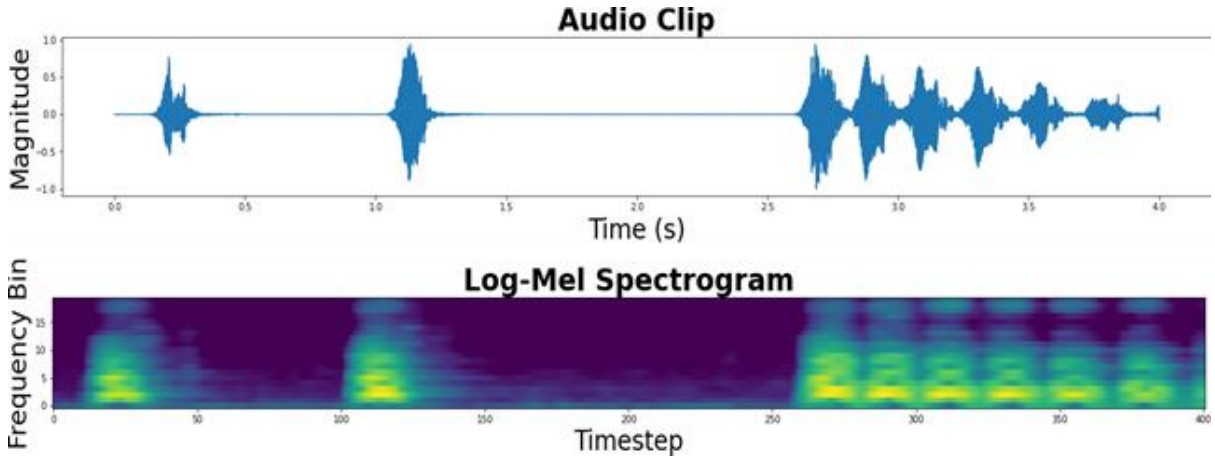



*Figure 3.2: Audio clip with its corresponding Log-Mel Spectrogram. This audio clip is of a dog barking from the UrbanSound8k dataset.*

applied. A Mel filter-bank is applied to the magnitude spectrum creating 20 Mel-spaced frequency bands between 0Hz and half the audio sampling frequency. The log power is then calculated giving a log-Mel spectrogram in dB. Alternatively, a similar process can be implemented using a time domain Mel filter-bank where the log energy of the time domain signals after filtering are calculated online, as in [32]. However, for simplicity and simulation speed, the transform will be conducted using the STFT. The result is a frequency-time representation of the audio data where the number of frequency bins matches the number of Mel-scaled filters, and there are timesteps equal to the length of the original audio data in seconds divided by 10ms (due to 20ms framing with 10ms overlap). An example audio clip and its corresponding log-Mel spectrogram is shown in Figure 3.2. These log-Mel spectrograms are then normalized per-sample to the range [0,1] using min-max scaling. For the encoding schemes considered in this chapter, each spike encoding scheme will be applied per frequency band of the log-Mel spectrogram.

## 3.2 Spike Encoding Scheme Classification Performance

### *3.2.1 No Spike Encoding*

Unlike SNNs, SVM classifiers do not require spike-encoded inputs. In this section, the performance of the SVM classifier on the IRIS and FSDD without any spike encoding is reported as a baseline. For each dataset, two versions will be classified by the SVM classifier: one with normalized inputs to the range [0, 1] and one where the data is not normalized at all. This is done because many of the encoding schemes in this chapter require normalized input data, so the best comparison will be against the normalized classification performance in this section. The results from these experiments are shown in Table 3.1.

*Table 3.1: SVM Classification Results*

| **Classifier** | **Classification Validation Accuracy** | | | |
|---|---|---|---|---|
| | **IRIS dataset** | | **FSDD** | |
| | **Normalized** | **No Normalization** | **Normalized** | **No Normalization** |
| Linear SVM | 92.0% | 95.6% | 32.0% | 95.4% |

When the input data is normalized to the range [0,1], the SVM classification performance decreases for both datasets. In the case of the FSDD, the decrease in performance is much more severe which could indicate that the datasets become less linearly separable when normalized. For the encoding schemes in the following sections, SVM performance will be included to see if the normalized results in Table 3.1 are improved at all when moving to a spiking representation. Furthermore, if classification accuracy can match or exceed the non-normalized SVM performance, it is

a good indication that the given encoding scheme is accurately representing the input data (at least for the purpose of classification).

### *3.2.2 Rate Encoding*

The first spike encoding scheme to be tested is a direct rate encoding scheme. This encoding method transforms the input data into a spike train where the spikes occur at a rate proportional to the magnitude of the input data. To this end, an encoding window with T timesteps will need to be established where the maximum firing rate is 1 spike per timestep. The input data will be normalized to the range [0,1], where a value of 1 in the input data will result in approximately the maximum firing rate, and 0 will result in no firing during the encoding window. To achieve this, spikes will be generated according to a Poisson process where the mean firing rate will be proportional to the magnitude of the normalized input data. For the given input sample and encoding window, the spikes within that window can be generated according to [76]:

$$P[spike\ during\ \delta t] \approx r\delta t \tag{3.1}$$

$$Encoded\ Signal[j] = 1\ \ if\ \ x[j] \leq r[i]\delta t \tag{3.2}$$

Where $r[i]$ is the normalized input data at timestep *i* of the original data sequence and $\delta t$ is the simulation timestep length which is set to 1ms. The value $x[j]$ is an element of a sequence of random values sampled from the uniform distribution between 0 and 1. The index *i* corresponds to the original data sequence and ranges from [0,S] where S is the total number of samples, and the index *j* corresponds to the encoded sequence window and ranges from [0,T] where T is the encoding window length. A new random sequence $x[j]$ is generated for each input data point $r[i]$. It should be noted that this

encoding scheme greatly increases the number of timesteps if the input is already in the time domain; since each input value requires an encoding window of length T, the number of timesteps becomes TxS. For audio data, this drastically slows down simulation speed since the data could contain many timesteps even before spike encoding.

Two versions of this rate encoding will be tested: one with an encoding window of T=20 timesteps, and one with an encoding window of T=1 timestep which will try to capture an instantaneous firing rate. Note that the case of T=1 only will apply to the audio dataset since this would result in a single vector with no time dimension in the case of the IRIS dataset. The classification results of the rate encoding scheme are summarized in Table 3.2.

*Table 3.2: Rate Encoding Classification Results*

| **Classifier** | **Classification Validation Accuracy** | | | |
|---|---|---|---|---|
| | **IRIS dataset** | | **FSDD** | |
| | **T=1** | **T=20** | **T=1** | **T=20** |
| Linear SVM | - | 63.3% | 37.67% | 71.0% |
| 2 Layer SNU | - | 80.7% | 47.0% | 57.0% |

As can be seen in the above table, the rate encoding scheme seems to result in relatively low accuracies for both the SVM and SNU classifier. On the IRIS dataset, the SNU classifier significantly outperformed the SVM classifier; however on the FSDD the SVM classifier outperformed the SNU classifier at an encoding window length of 20. For both classifiers on the FSDD, increasing T from 1 to 20 improved classification accuracy

although the absolute accuracy was still poor. This indicates that increasing the window length could increase accuracy. The effect of the encoding window length on classification accuracy is tested later in this chapter.

### *3.2.3 PDM Encoding*

A pulse density modulation (PDM) scheme for spike encoding is tested in this section. Under PDM, an analog signal is converted to a binary digital signal where the density of 1's increases as the analog signal's magnitude increases, and decrease when the input signal decreases [77]. This can be thought of as another form of rate coding, where a constant valued input would correspond to a constant spike rate. However, this scheme allows the output spike trains to match the number of timesteps of the input temporal data. A pulse-code modulated signal (PCM), such as audio data, can be converted into a PDM signal using the following equations:

$$y[n] = sign(x[n] - e[n-1]) \tag{3.3}$$

$$e[n] = y[n] - (x[n] - e[n-1]) \tag{3.4}$$

Where y[n] is the PDM signal at timestep n, x[n] is the input signal, and e[n] is a running quantization error with $e[0] = 0$. For the IRIS dataset, the input values x[n] will be the

*Table 3.3: PDM Encoding Classification Results*

| **Classifier** | **Classification Validation Accuracy** | |
|---|---|---|
| | **Iris Dataset (T=20)** | **FSDD** |
| Linear SVM | 86.0% | 81.7% |
| 2 Layer SNU | 90.7% | 78.7% |

feature vector values repeated T times where T is the length of the desired encoding window (set to 20 timesteps). The results on the two datasets using a PDM spike encoding scheme are shown in Table 3.3.

Compared to the previous rate encoding scheme the performance for both classifiers under the PDM encoding scheme is greatly improved, although they still underperform compared to the baseline SVM. On the FSDD, there is no requirement to have an additional time encoding window; so compared to the T=1 case under the rate encoding scheme, the performance almost doubled for both classifiers when switching to PDM encoding. However, the PDM encoding was developed for digital signal processing and is not necessarily biologically plausible. A drawback from this type of encoding is that it has a fixed dimension, and there's no way to increase data representation by extending the encoding window (or population size like some of the later encoding schemes). Increasing the fidelity under the PDM encoding scheme would require re-sampling the input signals at a higher rate, which may not always be possible.

### *3.2.4 Latency Encoding and Population Latency Encoding*

In contrast to the rate encoding schemes, latency encoding schemes encode input values to the relative timing between spikes. An encoding window of length T will be required, which is set to 20 timesteps for these experiments to match the encoding window length used in the previous experiments. The idea behind latency encoding is to let larger values of the input data cause a spike earlier in time relative to the encoding window, and values with a lower magnitude will cause a spike later in time within the

encoding window. In discrete time, with input data normalized to [0,1], the spike timing within the encoding window for a given value can be described by:

$$t_{spike} = T - \lfloor xT \rfloor, x > 0 \tag{3.5}$$

Where *x* is the normalized input value, $\lfloor \cdot \rfloor$ is the floor operator, and $t_{spike}$ is an integer timestep spike time relative to the encoding window of length T. This encoding scheme is similar to a pulse position modulation (PPM) scheme, where each input value is converted to a single time shift per encoding window [78]. However, instead of encoding 0-valued input as a specific offset, an input of 0 will result in no spikes during the encoding window. This causes the encoding scheme to be event based, where spikes are only generated when there is non-zero input data. Results using a PPM scheme where 0 is directly encoded as an offset are included for comparison. Additionally, a differential pulse position modulation (DPPM) scheme is tested where the trailing zeros after a spike are removed [78]. For the IRIS dataset, this will result in the same encoding as the PPM encoding since there is only one encoding window. Results for the PPM and DPPM encoding schemes are shown in Table 3.4.

*Table 3.4: PPM and DPPM Encoding Classification Results*

| **Classifier** | **Classification Validation Accuracy** | | | |
|---|---|---|---|---|
| | **IRIS dataset** | | **FSDD** | |
| | **PPM** | **DPPM** | **PPM** | **DPPM** |
| Linear SVM | 94.0% | - | 95.7% | 58.3% |
| 2 Layer SNU | 57.3% | - | 83.7% | 31.3% |

The above table clearly shows that the PPM encoding scheme seems to work well for the SVM classifier, however the performance is quite poor for the SNU classifier. This could indicate that the data becomes more linearly separable when the entire encoded signal is available at the same time as is the case for the SVM classifier, but when it is presented sequentially the classification problem remains difficult. When moving from PPM to DPPM encoding on the FSDD, both classifiers saw a significant decrease in accuracy. Since the encoding method is being applied per frequency band of the log-Mel spectrogram, removing the trailing zeros from each individual encoding window de-synchronizes the time domain between the frequency bands which could possibly destroy some of the information contained in the time-frequency representation. Under the PPM encoding scheme, the SVM classifier outperformed the baseline non spike-encoded performance with normalized input data. This indicates that the PPM encoding scheme may actually increase the linear separability of the input data which in turn helps the linear SVM performance. Furthermore, the SVM slightly outperformed even the non-normalized, non-spiking baseline on the FSDD under the PPM encoding scheme. So although the performance using the SNU classifier is still poor under the PPM encoding scheme, it does seem to help improve classification accuracy for the linear SVM.

An extension of the event-based latency encoding scheme is to distribute the input value over a population of neurons with overlapping gaussian receptive fields. Under a population latency encoding scheme, depending on where the input value falls within a given neuron's receptive fields, a set of spikes will be generated with each neuron spiking at a different time offset. This encoding scheme is illustrated in Figure

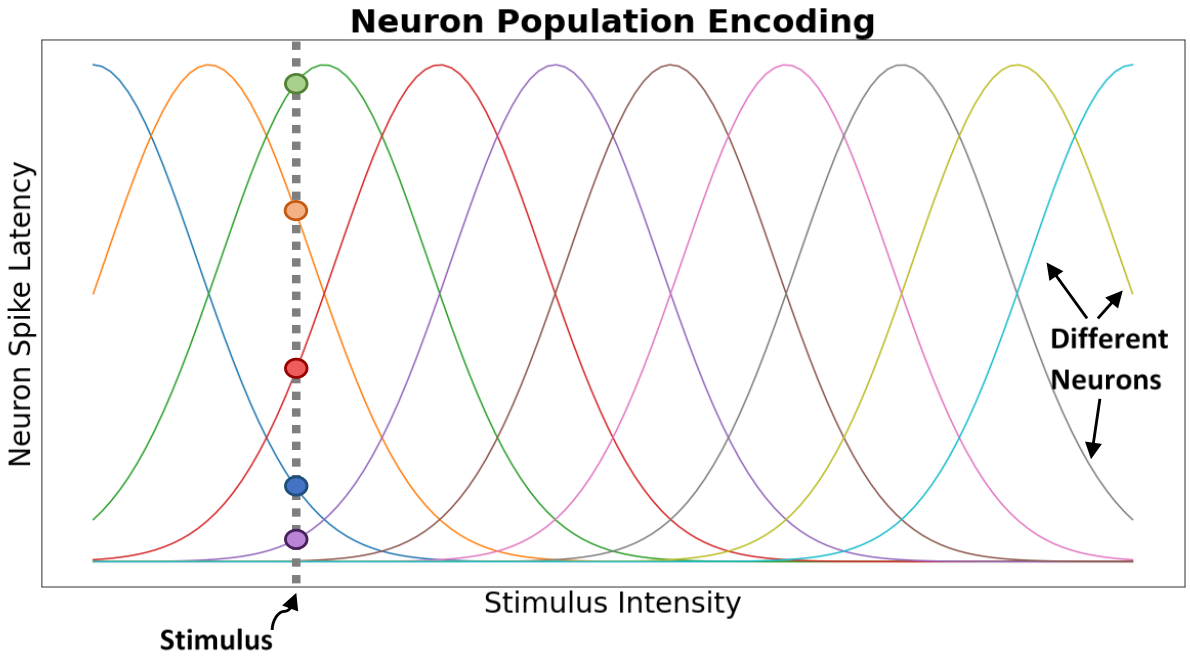



*Figure 3.3: Illustration of a population latency encoding. Each gaussian receptive field belongs to a corresponding neuron, and depending on where the input value falls in each neuron's receptive field, a spike will be triggered at a temporal offset corresponding to the value of that neuron's gaussian function at that point.*

3.3. The gaussian receptive fields are spaced evenly between 0 and 1, the peaks are normalized to 1, the standard deviation is set to the distance between consecutive gaussian centers, and the values of these functions at the stimulus value are used to replace *x* in Equation (3.5). For the experiments using latency encoding, the value P will

*Table 3.5: Latency Encoding Classification Results*

| **Classifier** | **Classification Validation Accuracy** | | | |
|---|---|---|---|---|
| | **IRIS dataset** | | **FSDD** | |
| | **P=1** | **P=20** | **P=1** | **P=20** |
| Linear SVM | 92.7% | 91.7% | 95.3% | 90.3% |
| 2 Layer SNU | 44.7% | 95.6% | 83.7% | 98.0% |

refer to the population number of encoding neurons; P=1 will use a single encoding neuron with the formula in Equation (3.5) directly and P=20 will be used to test a population of 20 encoding neurons with overlapping gaussian receptive fields as in Figure 3.3. The results from these experiments are shown in Table 3.5.

Table 3.5 shows the benefit of moving to a population encoding scheme as opposed to using a single neuron to encode the input value for the spiking classifier. With P=20, the SNU classifier outperforms the SVM on both datasets and achieves a higher accuracy than was achieved under any encoding scheme tested so far. The population encoding scheme does not seem to benefit the SVM at all, and leads to a ~5% decrease in accuracy on the FSDD. For the standard latency encoding scheme (P=1) the SVM performed similarly to the PPM encoding scheme for both datasets. On the FSDD, the SNU classifier performed similarly to the PPM encoding scheme when P=1 which is expected. However, at P=1 for the IRIS dataset, the SNU classifier underperforms when compared to the PPM encoding scheme by ~10%. This could be due to the normalization scheme used in these experiments, where each input feature is scaled to the range [0 1] with 0 being the minimum feature value. Due to this, the smallest features of the IRIS dataset will result in no spikes under the standard latency encoding scheme, even though their true values are non-zero. In the PPM encoding scheme, the smallest features will result in a maximally offset spike. For the FSDD, explicitly encoding the 0 valued inputs may not be required since there are significantly more features and timesteps of the original data being encoded which produces many more spikes overall to help inform classification. Compared to the non-spiking SVM baseline, the SNU classifier with the population latency encoding scheme matched the

non-normalized input SVM performance on the IRIS dataset and exceeded the non-normalized input SVM performance on the FSDD. This is very promising, since under this encoding scheme the spiking neural network begins to outperform even the non-normalized, non-spike encoded SVM.

### *3.2.5 Threshold Population Encoding*

The final encoding scheme that will be evaluated is referred to as threshold population encoding [42][79]. Under this encoding scheme, a population of neurons are assigned unique threshold values, and a spike is generated if a time-varying input crosses that neuron's threshold. There are two neurons assigned per threshold value, one for onset and one for offset crossings. Since this scheme requires time-varying data, it is not applicable to the non-temporal IRIS dataset. A major advantage of this encoding scheme compared to the previous encoding schemes is that an extra encoding window is not required. The time dimension of the threshold encoding matches the number of timesteps of the input data. This encoding scheme essentially

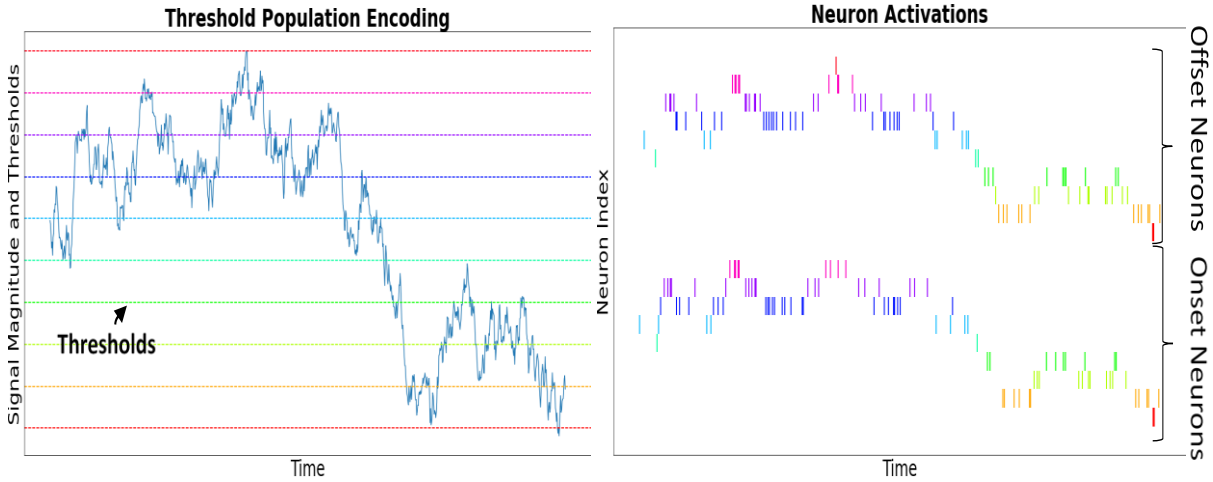



*Figure 3.4: Threshold encoding for a randomly generated signal. There are ten thresholds in this image, which corresponds to 20 encoding neurons with 10 onset and 10 offset neurons.*

encodes the signal information in the spatial domain as opposed to completely in the temporal domain, as was done with the rate/latency codes. This is beneficial as it is easier to parallelize spatial signals than sequential signals, since the values from a given timestep can be computed in parallel. An example signal and the corresponding threshold encoded spike trains is shown in Figure 3.4. To test this encoding scheme on the FSDD, 15 thresholds spaced evenly between 0 and 1 are assigned per frequency bin of the normalized log-Mel spectrogram. The results from these experiments are shown in Table 3.6.

*Table 3.6: Threshold Population Encoding Classification Results*

| **Classifier** | **Classification Validation Accuracy** |
|---|---|
| | **FSDD** |
| Linear SVM | 71.0% |
| 2 Layer SNU | 99.3% |

Similar to the population latency encoding scheme, the threshold encoding scheme shows the benefit of encoding input values with a population of neurons as opposed to a single neuron. Under this encoding scheme, the highest classification accuracy on the FSDD was achieved among the considered encoding schemes when using a spiking classifier. The SVM classifier saw a significant drop in performance when compared to some of the previous encoding schemes. So once again, the population encoding seems to significantly benefit the SNU classifier while decreasing the SVM's performance. Furthermore, the SNN classifier is now significantly outperforming the baseline SVMs performance on the FSDD under this encoding scheme.

## 3.3 Additional Encoding Scheme Comparisons

### *3.3.1 Average Number of Spikes*

In addition to their effect on classification performance, the encoding schemes presented in this chapter can also affect the power requirements of the SNN. If implemented on neuromorphic hardware, power is primarily consumed when spikes are emitted within the SNN. Due to this, encoding schemes can be evaluated and compared by looking at their total spike output over a given dataset. Schemes which result in more spikes will result in a higher power consumption, so minimizing the average amount of generated spikes would be beneficial. To this end, the considered encoding schemes are compared by averaging the total number of spikes emitted per data sample under a given encoding scheme for the IRIS and FSDD. These values are then normalized by dividing the average number of emitted spikes per-sample by the number of features and the number of timesteps of the original data sample. This is done to see the effect of the spike encodings independent of the input data shape.

Figures 3.5 and 3.6 show the average number of spikes per input value over the IRIS dataset and FSDD respectively. Compared to the temporal encoding schemes, the standard rate encoding scheme produces significantly more spikes on average. The PDM encoding scheme produces a relatively low number of spikes for the FSDD, and a high number of spikes for the IRIS dataset. This is due to the nature of the input data. The IRIS dataset maintains a fixed input value for each timestep under the PDM scheme, which results in a fixed spike density over the duration of the input sample. The signals from the FSDD change over time, and there are many timesteps where the input values could be close to zero. This results in much lower spike density over the course

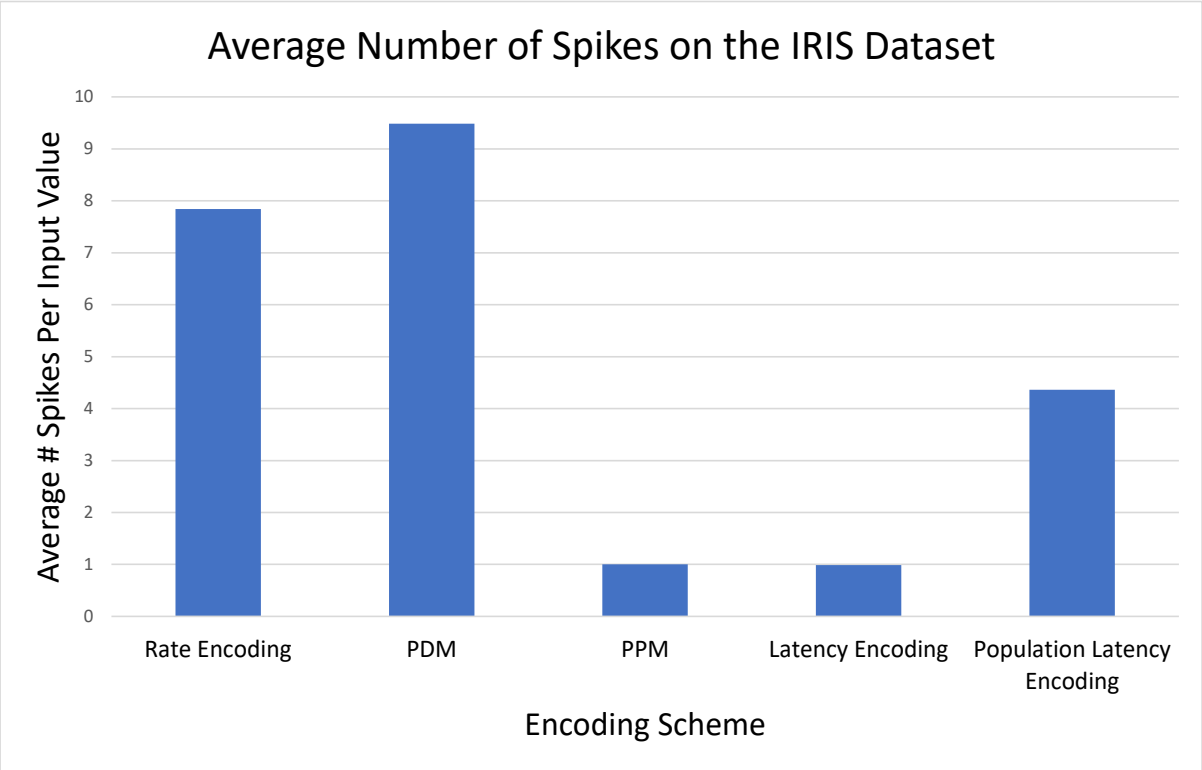



*Figure 3.5: Average number of spikes on the IRIS dataset.*

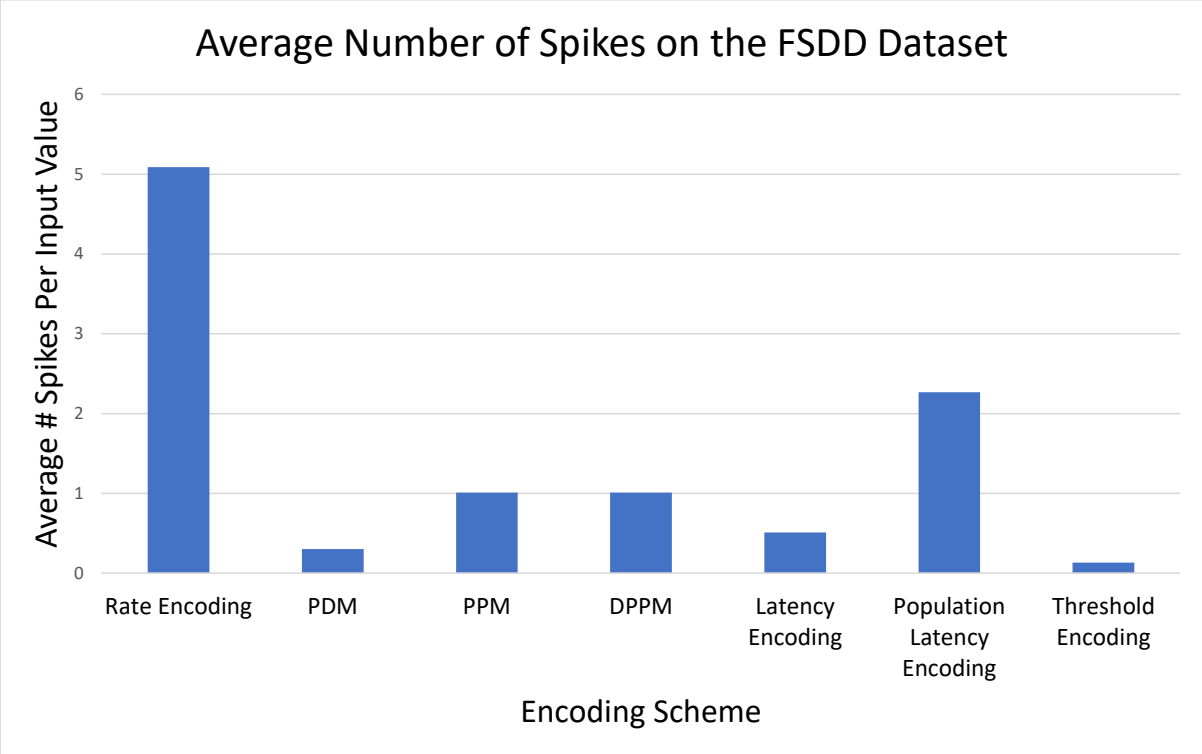



*Figure 3.6: Average number of spikes on the FSDD.*

of the input sample for the PDM scheme on the FSDD. Since both the rate encoding and PDM encoding schemes can lead to a high number of spikes, and they achieved relatively lower classification performance, they are less suitable for the considered classification tasks via SNNs. The PPM/DPPM and latency encoding schemes performed similarly in terms of average number of spikes on both datasets. Since the PPM/DPPM encoding schemes assign exactly one spike per input value (including zero values), they expectedly averaged to produce 1 spike per input value. For the IRIS dataset, this results in no difference when compared with the latency encoding scheme since almost all feature values of the IRIS dataset are non-zero, which results in a single spike per value. However, on the FSDD, it can be seen how the latency encoding scheme can save power by not emitting spikes for zero-valued input. Under the population latency encoding scheme, the number of emitted spikes increases significantly. This is expected, since multiple spikes are emitted per input value due to the population of encoding neurons. However, the number of spikes is still significantly lower than under the rate encoding scheme. Finally, it can be seen that the threshold encoding scheme achieves the lowest number of average spikes for the FSDD. This is very promising since it also achieved the highest accuracy with the SNN classifier.

### *3.3.2 Input Dimensions*

Another consideration for the encoding schemes is to look at their spatial and temporal dimensions. Keeping consistent with the notation introduced earlier in this chapter, let P represent the number of encoding neurons and T represent the number of timesteps used to encode a single input value. The requirements for the various encoding schemes are summarized in Table 3.7.

*Table 3.7: Encoding Scheme Dimensions*

| Encoding Scheme | P | T | Encoded Dimension |
|---|---|---|---|
| Rate Encoding | =1 | >1 | T |
| PDM | =1 | =1 | 1 |
| PPM/DPPM | =1 | >1 | T |
| Latency Encoding | =1 | >1 | T |
| Population Latency Encoding | >1 | >1 | PxT |
| Threshold Encoding | >1 | =1 | P |

In this table, the “Encoded Dimensions” column indicates the encoding size required per input data value, whether distributed in time (T) or space (P). In the case of temporal input data, this would be the dimension of the encoded output per timestep of the original data (number of timesteps of original data defined as S). As mentioned earlier in this chapter, the encoding schemes which encode the data in time inflate the time dimension of the incoming data by a factor T, which drastically slows down simulation speed (and would require faster hardware implementations to maintain the same data rate). Increasing the P dimension implies an increase in the number of neurons required to encode the input data, however this does not drastically affect the system speed since these computations can be carried out in parallel in the same timestep. Due to this, the encoding schemes which don’t encode information over the time dimension are preferable for time-varying input signals. This further motivates the use of the threshold encoding scheme as it encodes the input data solely over the spatial dimension. The PDM encoding scheme also avoids inflating the time dimension, however it is limited in that it only uses one encoding neuron. This could be part of the reason that it’s performance is lower than the temporal encoding schemes; it may not able to fully capture the information in the input data because it is limited to a single binary 1 or 0 in a given timestep. The threshold encoding, in contrast, can increase its

resolution by increasing the number of encoding neurons (P) which has a smaller effect on computation speed than increasing T. In the case of non-temporal data, the population latency code could be effective by using a chosen T value to add a time dimension and increasing P to better represent the input data.

To test the effect of the encoding dimensions for the various encoding schemes, the SNU network will be applied to the IRIS dataset under different values for P and T. The population size for the threshold encoding scheme will be tested using the FSDD, since it requires time-varying inputs and is not applicable to the IRIS dataset. The IRIS dataset is used for all other encoding schemes because larger T quickly becomes impractical when working with the FSDD; simulation times increase drastically. Beginning with the rate encoding scheme, encoding window lengths of T=10, 20, 50, 100, and 200 were tested. The results from this experiment are shown in Figure 3.7. It is

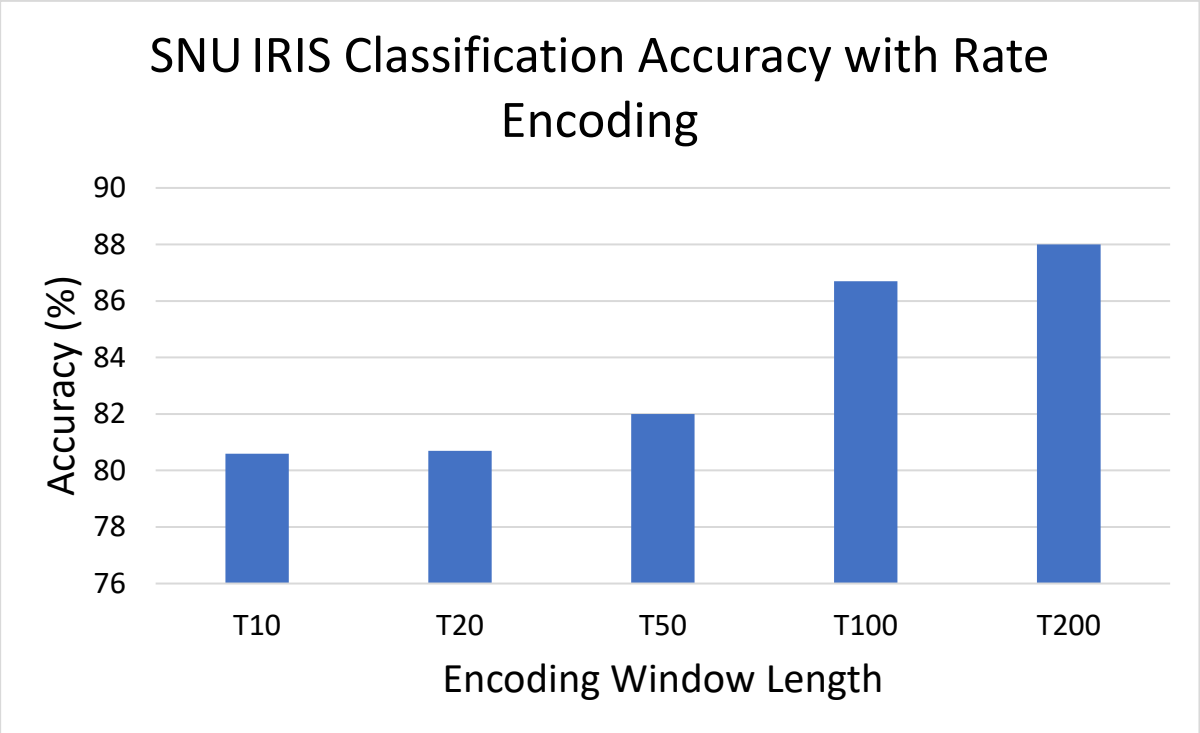



*Figure 3.7: Effect of rate encoding window length on IRIS classification accuracy.*

clear that as the encoding window length is increased, the accuracy also increases. This makes sense since values can be represented more accurately by averaging the random rate-based spikes over a longer time frame. However, to achieve the higher accuracies in Figure 3.7, the encoding window had to be increased significantly to the point where the latency required to encode a single value becomes impractical. Furthermore, the gain in accuracy is small even when the encoding window is doubled. From T=50 to T=100, there was only a ~5% increase in accuracy, and when it is doubled again to T=200, there was only ~2% increase in accuracy. This encoding scheme could be reasonable for a static dataset such as image data or the IRIS dataset given a sufficiently long encoding window, however it is not suitable for temporal data due to the long encoding window requirements.

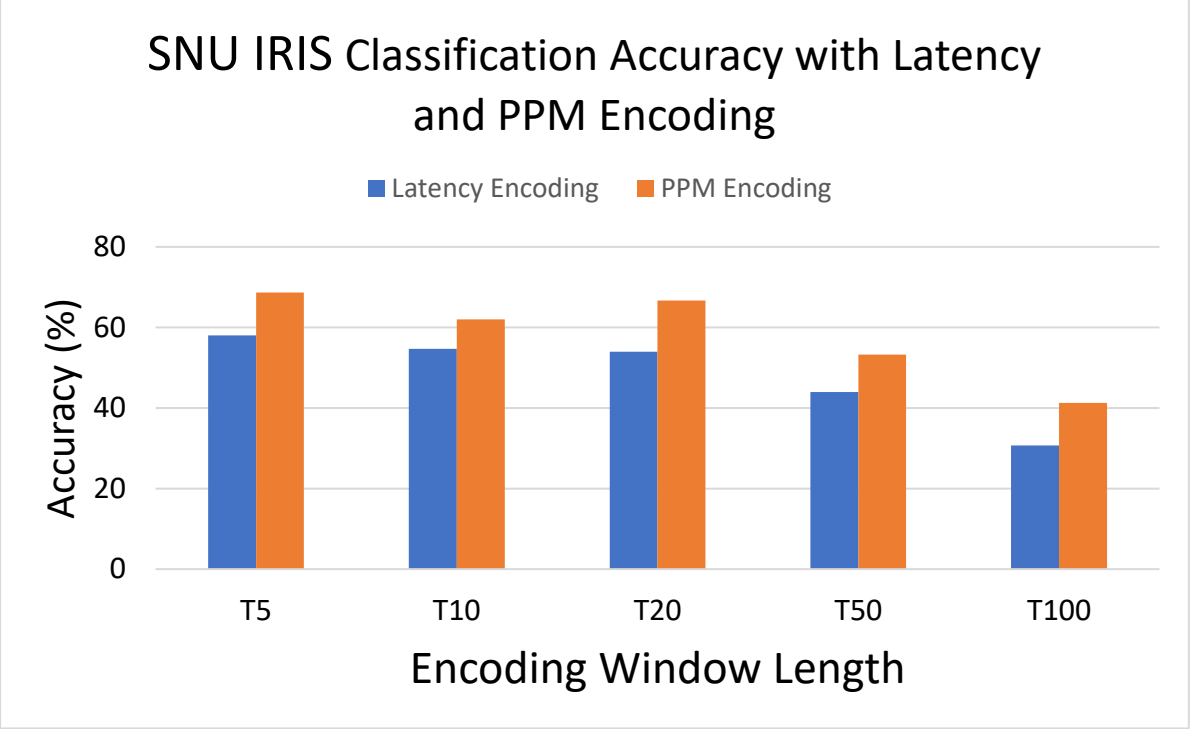



*Figure 3.8: Effect of Latency and PPM encoding window length on IRIS classification accuracy.*

Next, the effect of the encoding window length for the PPM and latency encoding schemes are examined. Interestingly, in Figure 3.8 it can be seen that for both schemes the accuracy is relatively stable for T5-T20. However, when it is extended significantly to T50 or T100, the accuracy begins to decrease. This seems counterintuitive, since longer encoding windows allow for more positions in which a spike could occur which should improve the encoding scheme's ability to represent the input data values. This decrease in accuracy may be due to the increased distance in time between spikes; at longer values of T the relative time between spikes for the different input features increases, but the total number of spikes remains constant. Compared with the rate encoding, the absolute accuracy for both of these schemes is significantly lower, at least for the SNU network applied to the IRIS dataset.

The population latency encoding scheme has two parameters to test: population size (P) and encoding window length (T). A surface plot showing the performance of the SNU classifier on the IRIS dataset over various P and T is shown in Figure 3.9. Similar to the single neuron latency encoding scheme, the accuracy seems to decrease as the encoding window length is increased. However, this drop in performance can now be offset by increasing the neuron population size. Even at T=100, accuracies greater than 94% can now be achieved given a sufficiently large population size. Even in the worst tested case in this experiment, the accuracy was still ~80% with a population of 5 neurons and an encoding window length of 100. This is a major increase in accuracy compared to the ~65% accuracy achieved with the single neuron PPM and latency encoding schemes. Since an accuracy increase can be obtained by fixing the encoding

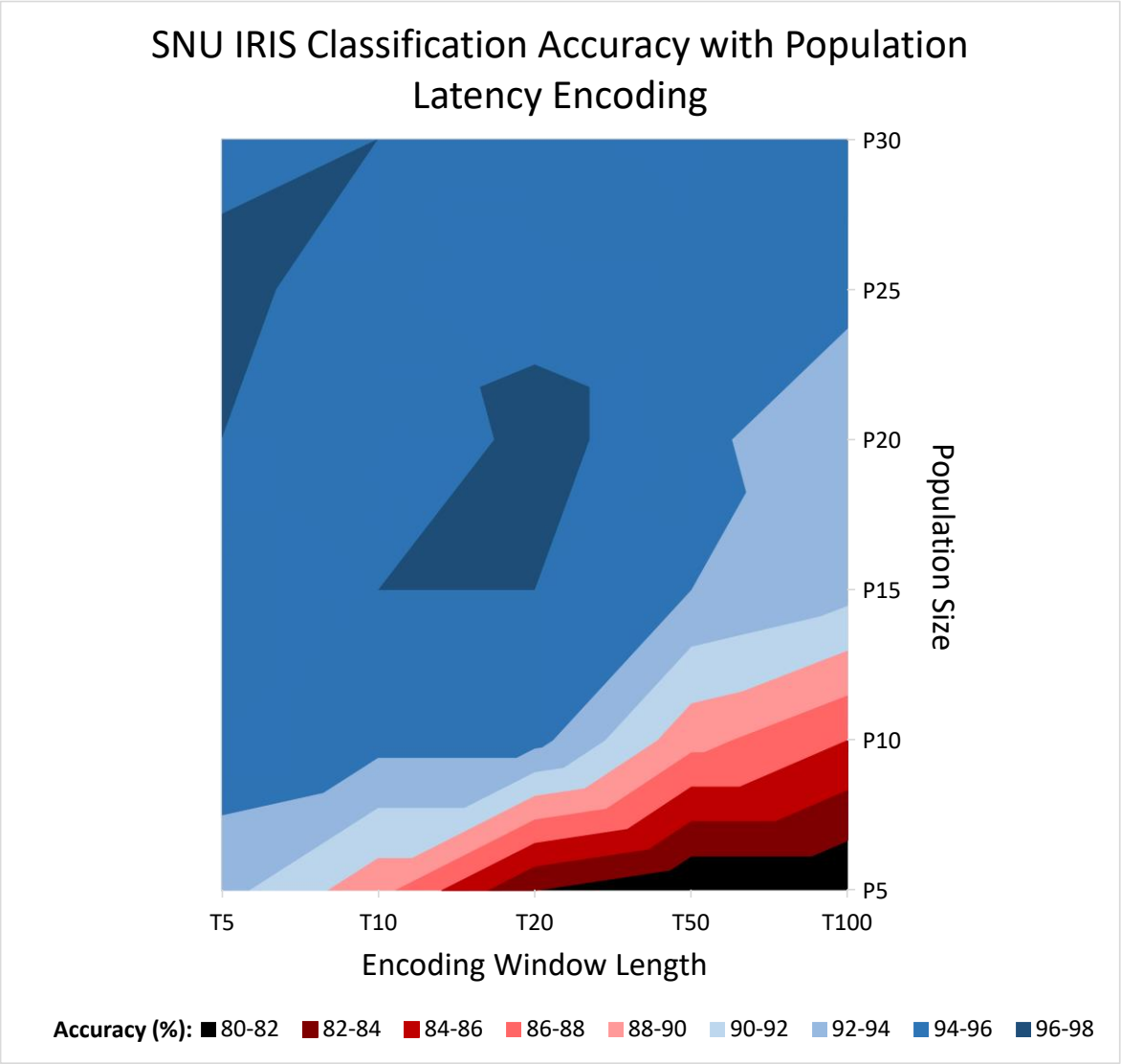



*Figure 3.9: Effect of population latency encoding window length and population size on IRIS classification accuracy.*

window length and increasing P, this scheme is promising for both temporal and non-temporal datasets since the temporal dimension can be kept relatively small.

Finally, the threshold encoding scheme will be tested by varying the number of threshold-encoding neurons over the FSDD. The results from this experiment are shown in Figure 3.10. The P values in Figure 3.10 refer to the number of thresholds used to encode the signal, but the number of encoding neurons is be 2x this value; one

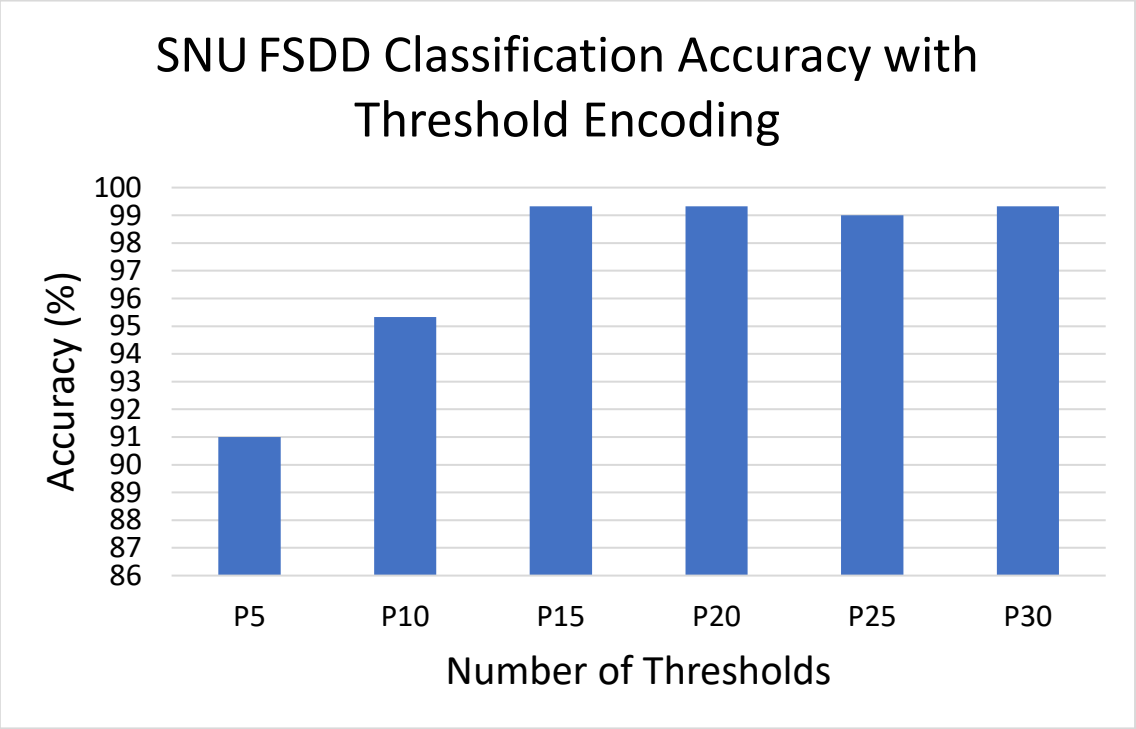



*Figure 3.10: Effect of threshold encoding population size on FSDD classification accuracy.*

for onsets and one for offsets per threshold. Accuracy increases up until P=15, where it seems to stabilize around 99%. Since this encoding scheme is able to achieve high accuracy without needing a temporal encoding window, it seems to be the most suitable for temporal data when using an SNN. Furthermore, it has the ability to increase the population size as needed depending on the given dataset to improve performance.

## 3.4 Discussion

Of the spike encoding schemes considered in this chapter, the population latency and the population threshold encoding schemes performed the best when combined with a multilayer SNN classifier. Under the population latency and threshold encoding schemes, the SNN classifier outperformed a linear SVM on both datasets across all encoding schemes, achieving 95.6% and 99.3% accuracy on the IRIS and FSDD

respectively. The rate encoding and PDM encodings seemed to perform relatively worse overall for both the SVM and the SNU classifier. The rate encoding performance could be improved by increasing T, however it quickly becomes impractical when applying long T to temporal input data. Furthermore, the rate encoding scheme emits the most spikes per input value which would directly lead to increased power consumption if implemented on neuromorphic hardware. The PPM encoding resulted in similar accuracies as the latency encoding scheme for both classifiers on both datasets. Since the latency encoding scheme only produces spikes when there is non-zero input data, it is preferred for its event-based encoding and lower average spike rate; however as was seen in the case of the IRIS dataset, if there are few input features encoding 0 directly may be beneficial. In any case, the biggest accuracy improvement for the SNU classifier came when moving to the population latency and threshold encoding schemes. The threshold encoding scheme is advantageous over the latency encoding schemes as it does not inflate the number of timesteps by requiring an encoding window of length T, and it also emits the lowest amount of spikes on average compared to the other encoding schemes. However, the threshold encoding scheme cannot easily be adapted for non-temporal inputs since it requires threshold crossings over time. So for non-temporal input data, a population latency encoding scheme seems to perform the best when using a spiking classifier. As further support for the two population encoding schemes, the performance of the SNU classifier matched the non spike-encoded SVM performance on the IRIS dataset and it significantly exceeded the SVM's performance on the FSDD under the population latency and threshold population schemes. The relative spike encoding performance observed in this chapter are in

agreement with the study conducted in [42], specifically that the population encoding schemes seem to perform best when using an SNN classifier. These experiments also show that applying a classifier which processes data temporally can be advantageous, since the highest accuracies on both datasets were achieved with the multilayer SNU under the appropriate spike encoding scheme.

# 4 Proposed SNN Learning Rule

Finding appropriate learning algorithms for directly training SNNs is an open research question. To achieve the power savings possible with SNNs, neuromorphic hardware which is designed for event-based spike computation needs to be used [16]. A limitation of this type of hardware is that algorithms such as BPTT may not be possible, since it is generally an offline algorithm and it requires storage of many intermediate variables that aren't necessarily available at a given time step [34]. The BPTT approach also reduces the biological plausibility since it requires signal quantities from many timesteps in the past to make weight updates. Biologically inspired learning rules based on spike-time dependant plasticity (STDP) make weight updates using only quantities thought to be available to a synapse at a given time, namely through eligibility traces of neuron spike timings which could be thought of as the residual charge remaining on a capacitor or as biochemical quantities in the brain [64]. However, these rules tend to be unsupervised and error credit assignment is not straightforward for supervised learning scenarios. In the interest of supervised learning the proposed approach is to use updates similar to STDP at a synapse based on eligibility traces, while still using backpropagated error information for targeted error credit assignment. Backpropagation in and of itself is not a biologically plausible learning rule, however the proposed approach attempts to combine biologically inspired synaptic weight updates with error signals provided by a backpropagation-like algorithm. This is similar to the approach taken in the Multi-ReSuMe, BP-STDP, and e-prop learning rules [37][20][36]. However, these learning rules either don't directly account for the spike timings of hidden layer neurons, or they were derived for IF neurons as opposed to LIF neurons.

To better assign error credit, the proposed rule suggests scaling the error signal in the backwards pass by the spike time eligibility traces of the incoming neurons to the given layer. This is done with the goal of more directly assigning temporal error credit since only neurons which contributed recently to the error at a given timestep will propagate the error signal to affect the weight updates in lower layers. This approach is beneficial because it only uses signals that exist at a given time step, it uses STDP-like weight updates, it can operate online, and it could more easily be implemented on neuromorphic hardware than BPTT as it avoids propagating error signals retroactively through time.

### 4.1 Proposed Learning Rule

The proposed learning rule will be defined for multi-layer feedforward networks of LIF spiking neurons. An example feedforward architecture with a single hidden layer is shown in Figure 4.1. Nodes in the network will be generically indexed by *i* and *j,* where *i* will represent the pre-synaptic neuron and *j* will represent the post-synaptic neuron for a given connection weighted by $w_{ji}$. In the special case of the output layer, nodes will be indexed by *k*. The LIF neuron dynamics will be modelled by the following equations, which were introduced in Chapter 2:

$$u_j(t) = u_j(t-1)e^{-\frac{\delta t}{\tau_m}} - S_j(t-1)\vartheta + \sum_i w_{ji}S_i(t) \tag{2.13}$$

$$S_j(t) = H\big(u_j(t) - \vartheta\big) \tag{2.14}$$

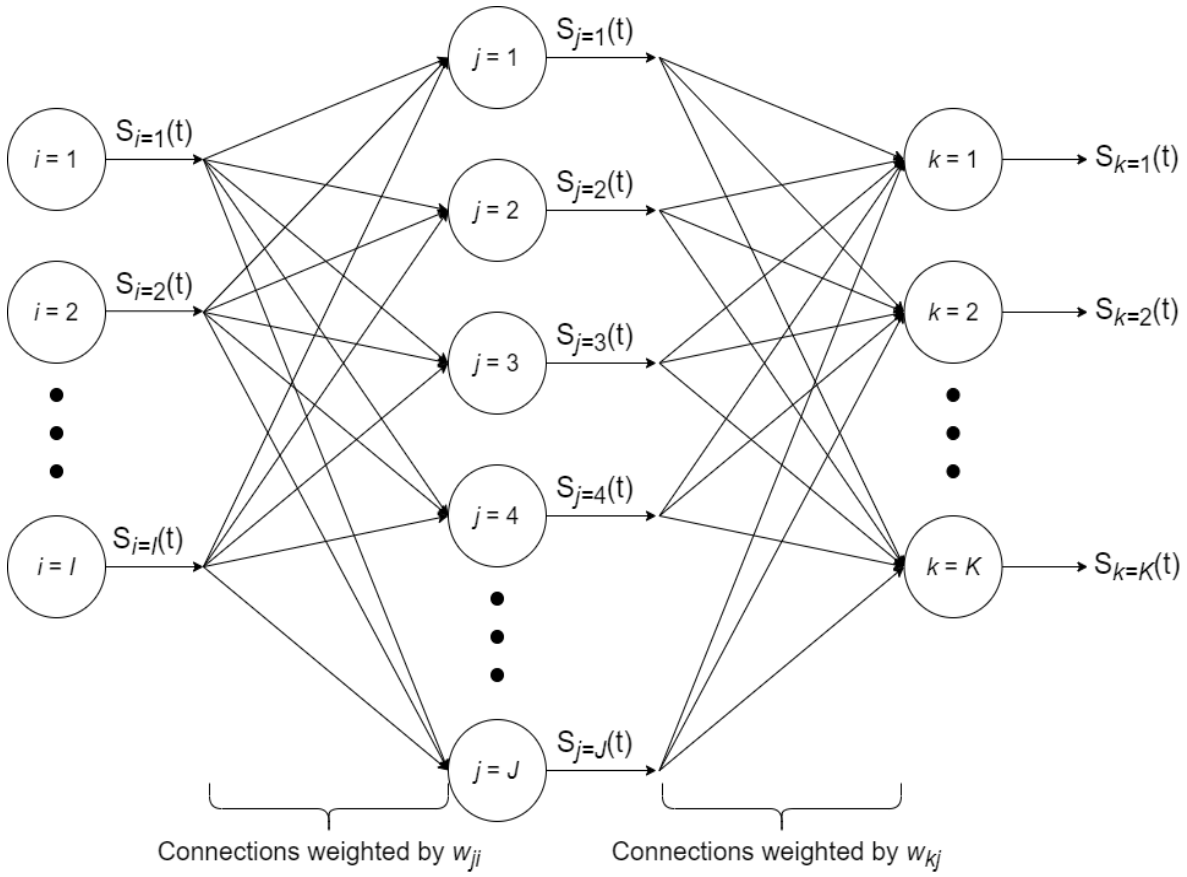



*Figure 4.1: Feedforward SNN architecture with a single hidden layer.*

The first portion of the proposed learning rule will be to use STDP-like updates to assign temporal credit in the network, which is done in the interest of avoiding BPTT while also introducing a biologically plausible weight update. To achieve this, eligibility traces will be maintained at each neuron which track how recently a given neuron spiked. Eligibility traces will decay exponentially with a time constant $\tau_\varepsilon$, which controls the length of the STDP learning window. A neuron's eligibility trace will be defined as:

$$\varepsilon_i(t) = \begin{cases} 1 & ,if\ S_i(t) = 1 \\ \varepsilon_i(t-1) \cdot e^{-\frac{\delta t}{\tau_\varepsilon}} & ,otherwise \end{cases} \tag{4.1}$$

In the above equation, $\varepsilon_i(t)$ is neuron *i*'s eligibility trace at time *t*, $S_i(t)$ is the spike train produced by neuron *i*, and $\delta t$ is the discrete timestep which is set to 1ms. Now, with the eligibility traces defined this way, STDP can be implemented by the following weight update for pre-synaptic neuron *i* and post-synaptic neuron *j*:

$$\Delta w_{ji}(t) = \begin{cases} +\varepsilon_i(t) & ,if\ S_j(t) = 1 \\ -\varepsilon_j(t) & ,if\ S_i(t) = 1 \end{cases} \tag{4.2}$$

The above weight update is an unsupervised form of STDP where each pre-synaptic/post-synaptic spike would trigger a weight change and anti-causal spikes would cause a negative weight change. However, under a supervised learning scenario, this update rule will be modified to allow a supervisory signal to determine the sign of the weight change and further modulate the amplitude of the weight change. This approach falls under the framework of a 3-factor neo-Hebbian learning rule described in [64]:

$$\frac{d}{dt} w_{ji}(t) = \varepsilon_{ji}(t) M_j^{3rd}(t) \tag{4.3}$$

In the general formulation of these 3-factor rules, the eligibility trace is defined as:

$$\frac{d}{dt} \varepsilon_{ji} = \eta f_i(x_i) g_j(y_j) - \frac{\varepsilon_{ji}}{\tau_e} \tag{4.4}$$

Where $\eta$ is a learning rate, $\tau_e$ is the eligibility trace time constant, $f_i(x_i)$ is some function if the presynaptic activity and $g_j(y_j)$ is some function of the post synaptic activity. So concretely in the case of the proposed learning rule, the simple eligibility traces defined in Equation (4.1) will be used to represent the joint pre- and post-

synaptic activity, giving:

$$\varepsilon_{ji}(t) = \eta\varepsilon_i(t)\varepsilon_j(t) \tag{4.5}$$

Under this framework, weight updates are only applied under the influence of the third factor modulatory signal $M_j^{3rd}(t)$. By choosing $\varepsilon_{ji}$ this way, the proposed STDP updates will be tracked in $\varepsilon_{ji}$ but only confirmed and applied when $M_j^{3rd}(t)$ is non-zero. The next step in the proposed learning rule is to find a way to assign structural error credit in a multi-layer SNN and combine it with the above eligibility traces which are responsible for temporal credit assignment via their STDP-like weight updates. Since the backpropagation algorithm is the most well-known algorithm for assigning structural error credit in ANNs, it will be adopted in the proposed algorithm. Specifically, $M_j^{3rd}(t)$ will be defined as a back-propagating error signal. Since the goal is supervised learning, an objective function will be defined as the squared difference between a desired output spike trains and the actual observed output spike trains:

$$E = \frac{1}{2}\sum_t\sum_k \left(S_k^d(t) - S_k(t)\right)^2 = \sum_t\sum_k E_k(t) \tag{4.6}$$

Here, the network outputs are $S_k(t)$ , and the desired (target) spike trains are $S_k^d(t)$ for neuron *k* in the output layer. With the objective function defined this way, it can then be split up over time and per output neuron giving a temporally local error signal for each output neuron:

$$E_k(t) = \frac{1}{2}\left(S_k^d(t) - S_k(t)\right)^2 \tag{4.7}$$

Now, the backpropagation algorithm can be applied per-timestep to assign structural error credit while the eligibility traces will account for temporal dependencies. Applying gradient descent:

$$\Delta w_{kj}(t) = -\eta \frac{\partial E_k(t)}{\partial w_{kj}} \tag{4.8}$$

$$\frac{\partial E_k(t)}{\partial w_{kj}} = \frac{dE_k(t)}{dS_k(t)} \frac{dS_k(t)}{du_k(t)} \frac{\partial u_k(t)}{\partial w_{kj}} \tag{4.9}$$

As mentioned in Chapter 2, $\frac{dS_k(t)}{du_k(t)}$ is problematic since it involves taking the derivative of the Heaviside step function which is non-differentiable. To overcome this, an approach similar to the straight-through estimator will be used as in [48] and [80]. In this case, the standard straight-through estimator would be given by:

$$\frac{dE_k(t)}{du_k(t)} = \frac{dE_k(t)}{dS_k(t)} \tag{4.10}$$

Which is essentially setting $\frac{dS_k(t)}{du_k(t)} = \mathbb{1}$**.** Other estimators were suggested in [48] such as using the derivative of the sigmoid function as an approximation, but they reported that the identity function seemed to work better. For the proposed learning rule, instead of applying the identity function to estimate this derivative, this value will be set to represent the post-synaptic spike timings. This will be achieved by setting $\frac{dS_k(t)}{du_k(t)} = \varepsilon_k(t)$. Since the eligibility trace is strictly positive, this won't change the sign of the gradient, but it will scale the magnitude of the error signal in proportion to the time since that neuron's last spike. Effectively, this transforms the algorithm to assign temporal error

credit based solely on relative spike timings without needing access to the instantaneous membrane potential of individual neurons in the network as would be required if the pseudo-derivative from e-prop was used. This modification was inspired by the BP-STDP algorithm, which set the derivative to 0 if a neuron hadn't spiked within a short time window $[t-\epsilon, t]$, and set it 1 one if it had [36]. In the BP-STDP algorithm, this was done as an approximation to the derivative of the ReLU function in a network of IF neurons where the neuron's membrane potential did not decay over time. However, in the case of the proposed algorithm, instead of setting the derivative value to 1 if the neuron spiked anywhere within a time window $[t-\epsilon, t]$, the eligibility trace will be used which will more precisely keep track of the time since last spike. This allows for a similar algorithm to the BP-STDP algorithm to be applied to a feedforward network of LIF neurons, and it also removes the requirement to look back in time $t-\epsilon$; the eligibility traces are maintained online so their value at time $t$ can be used directly. Furthermore, this approach gates the backpropagating error signal such that the structural error credit at a given timestep is assigned proportionally to the more recently spiking neurons, which would be responsible for causing erroneous output spikes. Although not as rigorously equivalent as in the case of the BP-STDP algorithm for IF neurons, this could still be considered to implement a ReLU-like derivative in the sense that the gradient is only propagated to lower layers if the neuron was activated (spiked) recently. The effect of this modification is tested under the ablation test section in Chapter 5, where it is compared with the standard straight-through estimator in Equation (4.10). The first two derivative terms in Equation (4.9) will be defined as follows:

$$\frac{dE_k(t)}{dS_k(t)} = -\left(S_k^d(t) - S_k(t)\right) \tag{4.11}$$

$$\frac{dS_k(t)}{du_k(t)} \approx \varepsilon_k(t) \tag{4.12}$$

The last term in Equation (4.9) would result in the following:

$$\frac{du_k(t)}{dw_{kj}} = S_j(t) + \frac{d(u_k(t-1)e^{-\frac{\delta t}{\tau_m}})}{dw_{kj}} - \frac{d(S_k(t-1)\vartheta)}{dS_k(t-1)} \cdot \frac{dS_k(t-1)}{du_k(t-1)} \cdot \frac{du_k(t-1)}{dw_{kj}} \tag{4.13}$$

$$\frac{du_k(t)}{dw_{kj}} = S_j(t) + e^{-\frac{\delta t}{\tau_m}} \cdot \frac{du_k(t-1)}{dw_{kj}} - \vartheta \cdot \varepsilon_k(t-1) \cdot \frac{du_k(t-1)}{dw_{kj}} \tag{4.14}$$

$$\frac{du_k(t)}{dw_{kj}} = S_j(t) + \left(e^{-\frac{\delta t}{\tau_m}} - \vartheta \cdot \varepsilon_k(t-1)\right) \cdot \frac{du_k(t-1)}{dw_{kj}} \tag{4.15}$$

This expression would normally be further unrolled through time under BPTT due to the recurrent dependencies of the current membrane potential on previous membrane potential values. Since the goal of the proposed approach is to capture temporal dependencies via simple STDP eligibility traces, this term will also be replaced with its corresponding eligibility trace defined in Equation (4.1). This is not a gradient-exact replacement, however it may be a reasonable approximation. A similar approximation was made in the e-prop learning rule, where simple exponentially decaying eligibility traces were used to approximate this term; their work showed that this approximation ignores the temporal dependencies from the reset dynamics, however they found that using a more exact expression for this derivative did not seem to improve performance

[37]. To show that using a simple eligibility trace could be a reasonable approximation (ignoring the reset dynamics), consider the case where there is an input spike from neuron *j* sometime in the past, say at $t_{spike}$. In this case, the only time $S_j(t) = 1$ is when $t = t_{spike}$. A short time later, this derivative term would now be:

$$\frac{\partial u_k(t)}{\partial w_{kj}} = \frac{\partial u_k(t)}{\partial u_k(t-1)} \cdot \frac{\partial u_k(t-1)}{\partial u_k(t-2)} \cdot \ldots \cdot \frac{\partial u_k(t_{spike}+1)}{\partial u_k(t_{spike})} \cdot \frac{\partial u_k(t_{spike})}{\partial w_{kj}} \quad (4.16)$$

Now, it is noted that:

$$\frac{\partial u_k(t)}{\partial u_k(t-1)} = e^{-\frac{\delta t}{\tau_m}} = \sigma \quad (4.17)$$

Where $\sigma$ is a decay constant. The derivative through time will terminate at the last spike time, $t_{spike}$, giving $\frac{du_k(t_{spike})}{dw_{kj}} = 1$. This gives an exponentially decaying eligibility trace,

$$\frac{\partial u_k(t)}{\partial w_{kj}} = \sigma^{N_T} \cdot 1 \quad (4.18)$$

$$N_T = \ number\ of\ timesteps\ since\ t_{spike} \quad (4.19)$$

This shows that it could be a reasonable approximation to use eligibility traces in place of the full temporal derivative term. In the interest of more flexible STDP models, this term will be approximated by the STDP eligibility trace, which allows the decay time constant to differ from $\tau_m$ :

$$\frac{\partial u_k(t)}{\partial w_{kj}} \approx \ \varepsilon_j(t) \quad (4.20)$$

Another difference between the proposed eligibility traces and the ones used in the e-prop learning rule is that the proposed algorithm uses saturating eligibility traces. This is done to invoke a nearest neighbor STDP interaction, where the weight update is determined primarily by the interaction between the last input spike and the most recent output spike [58]. By defining $\frac{dS_k(t)}{du_k(t)}$ and $\frac{\partial u_k(t)}{\partial w_{kj}}$ as above, an STDP interaction now governs the weight update modulated by the error value $\frac{dE_k(t)}{dS_k(t)}$, which can be expressed as a 3-factor rule as in Equation (4.3). This allows temporal error credit to be captured by the STDP interaction between $\varepsilon_k(t)$ and $\varepsilon_j(t)$, while the spatial error credit is determined by $\frac{dE_k(t)}{dS_k(t)}$. Analogously to the backpropagation algorithm, this error signal can be propagated to lower layers as follows:

$$\frac{\partial E_k(t)}{\partial w_{ji}} = \frac{\partial E_k(t)}{\partial u_j(t)} \frac{\partial u_j(t)}{\partial w_{ji}} \approx \frac{\partial E_k(t)}{\partial u_j(t)} \cdot \varepsilon_i(t) \tag{4.21}$$

$$\frac{\partial E_k(t)}{\partial u_j(t)} = \frac{\partial E_k(t)}{\partial S_j(t)} \frac{dS_j(t)}{du_j(t)} \approx \left( \sum_k w_{kj} \cdot \frac{\partial E_k(t)}{\partial u_k(t)} \right) \cdot \varepsilon_j(t) \tag{4.22}$$

Or, expressing this algorithm as a three-factor learning rule:

$$\Delta w_{ji} = \eta e_i(t) e_j(t) M_j^{3rd}(t) \tag{4.23}$$

$$M_k^{3rd}(t) = \ target\ spike - actual\ spike\ ,\ \ k = output\ layer\ index \tag{4.24}$$

$$M_i^{3rd}(t) = \sum_j w_{ji} e_j(t) M_j^{3rd}(t)\ , \qquad otherwise \tag{4.25}$$

So in addition to the STDP process being implemented by the neuron eligibility traces, the backpropagating error signal is also gated by the eligibility traces of the neurons in upper layers. This is the main idea behind the proposed algorithm, which was inspired from the BP-STDP algorithm. However, by allowing the temporal dependencies to be approximated by the eligibility traces in a network of LIF nodes, the proposed algorithm may be better equipped to handle temporally encoded spike inputs whereas the BP-STDP algorithm was designed for rate-encoded inputs and outputs. Another motivation for the proposed algorithm was to combine biologically plausible local weight updates with a backpropagating error signal. However, this algorithm has many similarities to the gradient-based e-prop learning rule since their algorithm used simple decaying eligibility traces to assign temporal credit as well. The biggest difference between the proposed algorithm and the e-prop learning rule is the pseudo-derivative used to approximate the derivative of the spiking function. The e-prop learning rule uses a heuristic voltage-based approximation (Equation (2.37) in Chapter 2), whereas the proposed learning rule attempts to maintain equivalence to the STDP update rule. Another difference was the use of saturating eligibility traces in the proposed algorithm, which was done to maintain simple pair-based STDP interactions. It is also noted that the e-prop learning rule was defined for a single layer of recurrently connected LIF neurons, where the proposed algorithm is for non-recurrently connected feedforward SNNs with LIF neurons. So the proposed algorithm could also be seen as a minor extension of the e-prop learning rule to multilayer feedforward networks. A version of the proposed network which uses the e-prop pseudo-derivative (Equation (2.37) in Chapter 2) is tested and compared to the proposed algorithm under the

ablation tests section in Chapter 5. Another important difference between the proposed algorithm and the e-prop learning rule is that the proposed algorithm uses spike train targets directly whereas the e-prop learning rule was defined for real-valued targets and network outputs by using a readout layer where the membrane potential was interpreted as the real valued network output. By using spike train targets and outputs, the proposed SNN could potentially be embedded in a larger spiking system where the output spike trains could be passed on and used as input for another SNN.

To improve learning stability, two additional modifications were made to the proposed rule. The first issue that needed to be addressed was the silent neuron problem. This is an issue that also appears in other SNN learning algorithms, which occurs when neurons within the network spike infrequently or not at all. In the proposed learning rule, this would prevent the error signals from being propagated to lower layers since neurons which don't spike never increase their eligibility trace above 0. To remedy this, the eligibility trace used to approximate the derivative of the spiking function in the backwards pass was clipped to a small minimum value, which was set to 0.01 times the eligibility trace's maximum value. This allows small error signals to be propagated downstream even if their corresponding neurons did not spike; this change is similar to the small non-Hebbian constant '*a*' (Equation (2.25) in Chapter 2) which is used in the Multi-ReSuMe learning rule [20]. Alternatively, this could be viewed as a spiking version of the leaky ReLU function used in ANNs which also adds a small positive term where the standard ReLU derivative would be zero [81]. At the output layer, the difference between actual spikes and target spikes will be positive if an output neuron didn't spike when it should have, negative if it spikes when the target was 0, and 0 if it matches the

target spikes. Thus, positive errors lead to an increase (potentiation) of weight values which will increase the chances of a spike occurring next time a similar input pattern is presented, and negative errors will cause a decrease of weight values (depression) if the neuron spikes when it shouldn't. Due to this, $\varepsilon_k(t) = 1$ whenever that neuron receives a negative error. However, since positive errors indicate a lack of spiking at a desired spike time, $\varepsilon_k(t) < 1$ under positive errors. To balance the error signal magnitudes, when the neuron receives a positive error in the output layer (indicating that it didn't spike at the correct time and $\varepsilon_k(t) < 1$), the eligibility trace will be treated as if it did spike so that the full positive error can be communicated to the rest of the network. This is equivalent to using the standard straight-through estimator (Equation (4.10)) for the output layer only. This change makes the proposed learning rule directly analogous to the BP-STDP algorithm adapted to work in a network of LIF neurons; in

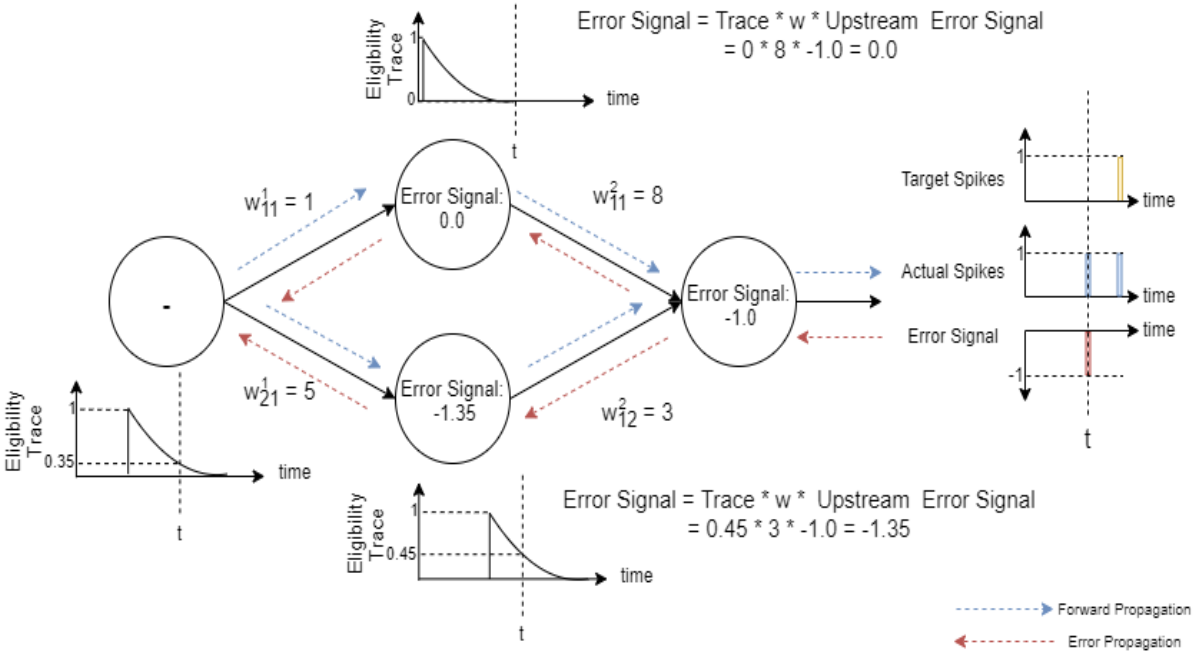



*Figure 4.2: Example error propagation under the proposed learning rule. Scaling by $n_l$ and $\varepsilon_i(t)$ clamping omitted for illustrative purposes.*

the BP-STDP learning rule they treated the output layer as having linear activation derivatives and the rest of the layers used their ReLU-equivalent derivative. Another change is inspired by the Multi-ReSuMe learning rule which was to scale the backpropagating error signal and the weight update by the number of neurons in the incoming layer, $n_I$; this was found to improve learning stability in the Multi-ReSuMe learning rule, and it also seemed to help the proposed algorithm. The proposed learning rule with these modifications included can be summarized as follows, and an example of the error propagation is shown in Figure 4.2:

$$E_k(t) = \frac{1}{2}\left(S_k^d(t) - S_k(t)\right)^2 \tag{4.26}$$

$$\Delta w_{ji}(t) = -\eta \frac{\partial E_k(t)}{\partial w_{ji}} \tag{4.27}$$

$$\frac{\partial E_k(t)}{\partial w_{ji}} \approx \frac{1}{n_I} \frac{\partial E_k(t)}{\partial u_j(t)} \varepsilon_i(t) \tag{4.28}$$

$$\frac{\partial E_k(t)}{\partial u_k(t)} = -(S_k^*(t) - S_k(t)), \quad if\ k = output\ layer\ neuron\ index \tag{4.29}$$

$$\frac{\partial E_k(t)}{\partial u_i(t)} = \frac{1}{n_I} \cdot \frac{dS_i(t)}{du_i(t)} \sum_j w_{ji} \frac{\partial E_k(t)}{\partial u_j(t)} \quad , for\ i, j = pre/post\ synaptic\ indices \tag{4.30}$$

$$\frac{dS_i(t)}{du_i(t)} \approx \max\left(\epsilon, \varepsilon_i(t)\right) \qquad , \epsilon = 0.01 \tag{4.31}$$

$$\varepsilon_i(t) = \begin{cases} 1 & ,if\ S_i(t) = 1 \\ \varepsilon_i(t-1) \cdot e^{-\frac{\delta t}{\tau_\varepsilon}} & ,otherwise \end{cases} \quad (4.32)$$

## 4.2 Testing the Proposed Learning Rule

To test the proposed learning rule, three experiments were conducted. First, an SNN was trained to output a specific target spike pattern in response to inputs where there was only 1 spike per input. Then, a similar experiment was conducted where each input spiked randomly multiple times throughout the pattern presentation. Finally, an SNN using the proposed learning rule was applied to solve a spike encoded XOR problem. For each experiment, a network with one hidden layer and a network with 2 hidden layers were trained for the respective task. The effect of learning rate and hidden layer size are also explored for the simple pattern recognition problems. These experiments show that the proposed learning algorithm can successfully train multilayer SNNs to output specific target spike trains in response to simple repeating input patterns, as well as solve non-linear problems such as the XOR problem.

### *4.2.1 Single-spike Input Pattern Association*

For this experiment, an input pattern was generated for a population of 1000 input neurons. Each of the 1000 neurons were assigned a single random spike time between 1 and 50ms. An SNN with 10 hidden neurons and 1 output neuron was then trained to output a target spike train of 1 spike every 10ms. Both the eligibility trace time constant and the membrane potential time constant were set to 10ms for this experiment. The input, target, and final output spike trains are shown in Figure 4.3. After approximately 50 repetitions of this frozen pattern presentation, the network was able to

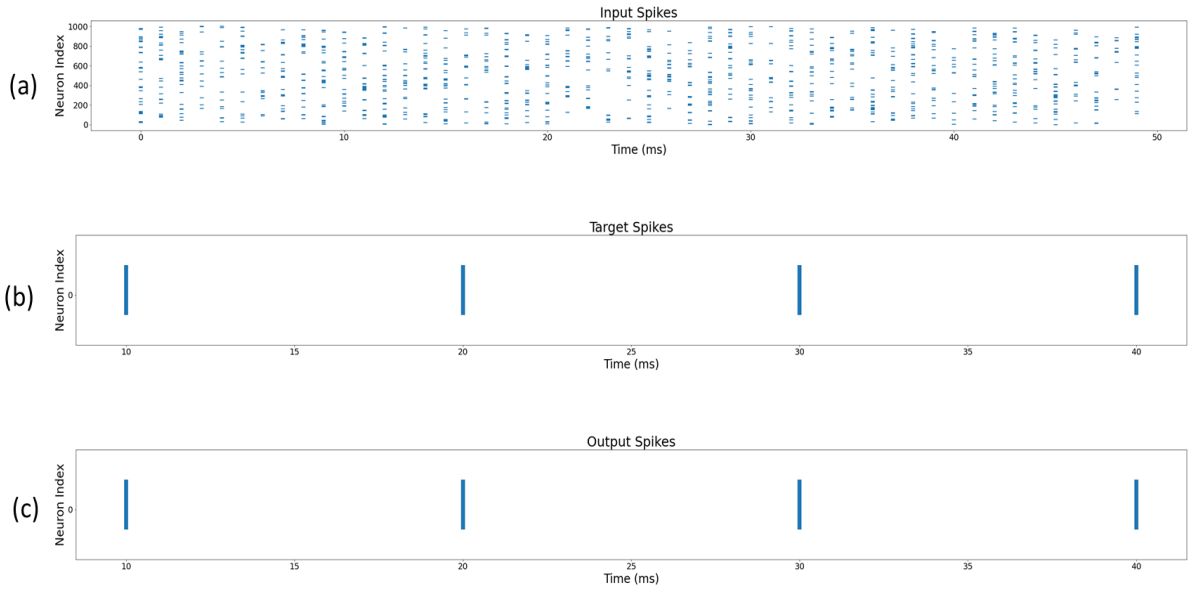



*Figure 4.3: Single spike pattern association. An SNN trained with proposed algorithm successfully outputs the target spike train for an input pattern where each input neuron is only allowed to spike once. Given the inputs shown in (a) and the target signal shown in (b), the network successfully output the spike train in (c) after training.*

exactly match the target pattern when presented with the input pattern. The parameters used for this experiment are summarized in Table 4.1.

*Table 4.1: SNN Parameters used in spike pattern association experiments.*

| **Parameter** | **Value** |
|---|---|
| $\tau_m$ | 10.0ms |
| $\tau_e$ | 10.0ms |
| $\vartheta$ | 10mV |
| $u(t)$ lower bound | $-\vartheta$ |
| $\eta$ | 0.1 |

To test the effect of learning rate and hidden layer size, the above experiment was repeated with varying learning rates and hidden layer neurons. A two-hidden layer

network with 10 neurons in both hidden layers was also tested to see if the proposed learning rule can be applied to deeper SNNs. The results of these experiments are summarized in Table 4.2. The results reported in this table are the number of pattern presentations required for the network to converge to the target spike train averaged over 5 trials under the given hidden layer configuration and learning rate. In most cases, the network was able to successfully converge to the desired output spike train; however, depending on the learning rate and hidden layer size, the number of epochs required could vary drastically. If at least one trial did not converge to the target spike train within 3000 epochs, it is represented by '-' in the table below.

*Table 4.2: Single spike association parameter experiments. Table values are the number of pattern presentations to converge to target spike pattern under the given learning rate and hidden layer configuration for single-spike input.*

| **Hidden Layer Size** | **Average # Epochs for Convergence** | | |
|---|---|---|---|
| | **η = 0.001** | **η = 0.01** | **η = 0.1** |
| **2** | - | 219.0 | 45.8 |
| **10** | 773.8 | 83.8 | 37.0 |
| **100** | 153.2 | 40.0 | - |
| **10-10** | 966.4 | 209.8 | 76.4 |

As can be seen in the above table, the proposed learning rule follows similar patterns to ANNs with respect to hidden layer size and learning rate. Faster convergence is seen with higher learning rates, however if it is set too high the network can fail to converge. Faster training is also seen with increased layer size and slower training is observed with increased number of layers. This experiment also shows that the proposed learning rule can be successful for deeper SNNs, where the 10-10

network contains two hidden layers of 10 neurons each. Both hidden layer size and learning rate will need to be tuned depending on the given task.

### *4.2.2 Multi-spike Input Pattern Association*

For this experiment, an input pattern was also generated for a population of 1000 input neurons. Each of the 1000 neurons were assigned to spike at a random rate in a 50ms time window. As in the previous experiment, an SNN with 10 hidden neurons and 1 output neuron was then trained to output a target spike train of 1 spike every 10ms. The other network parameters used in this experiment are the same as in Table 4.1. The input, target, and final output spike trains are shown in Figure 4.4. After approximately 85 repetitions of this frozen random pattern, the network was able to exactly match the target output

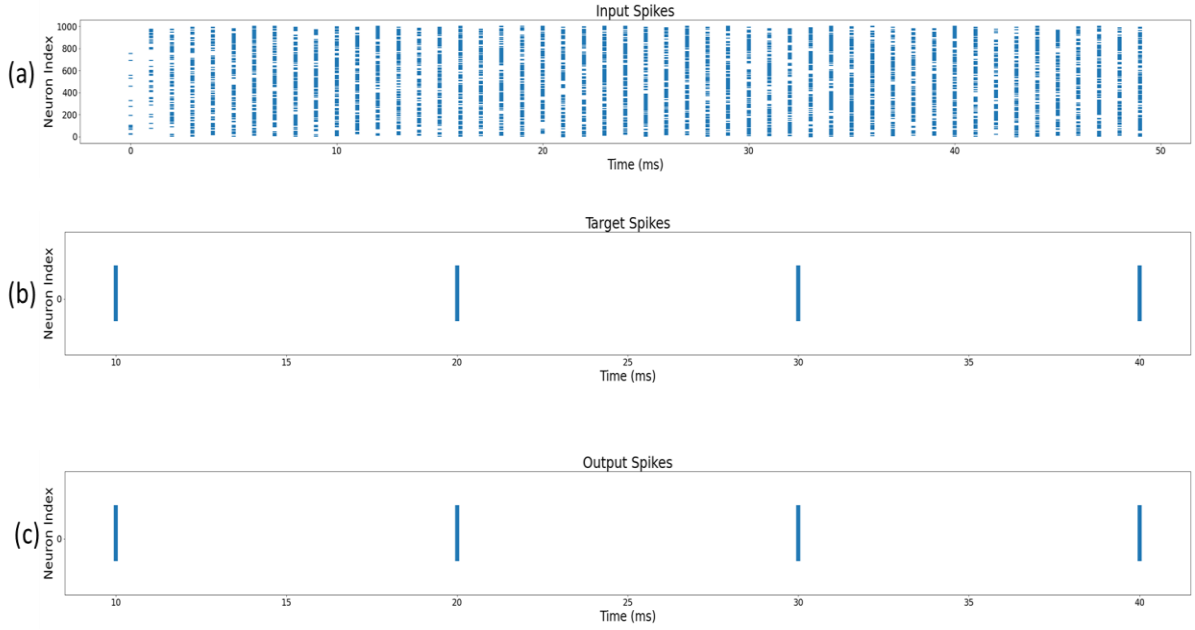



*Figure 4.4: Multi-spike pattern association. An SNN trained with proposed algorithm successfully outputs the target spike train for an input pattern where each input neuron randomly spikes multiple times. Given the inputs shown in (a) and the target signal shown in (b), the network successfully output the spike train in (c) after training.*

spike train, the network took longer to train compared to the single spike per input neuron case. This seems to indicate that high frequency input spiking can slow down training, and sparse input spikes may be desirable when possible. Similar to the single spike per input experiment, the proposed rule is tested using various learning rates and hidden layer configurations with the multi-input spike pattern. These results are summarized in Table 4.3. It was found that this task took longer to converge than the single input spike case, so the maximum number of epochs before considered to have not converged was set to 5000.

*Table 4.3: Multi-spike association parameter experiments. Table values are the number of pattern presentations to converge to target spike pattern under the given learning rate and hidden layer configuration for multi-spike input.*

| **Hidden Layer Size** | **Average # Epochs for Convergence** | | |
|---|---|---|---|
| | **η = 0.001** | **η = 0.01** | **η = 0.1** |
| **2** | - | 1292.8 | 235.0 |
| **10** | 2577.6 | 417.8 | 85.8 |
| **100** | - | - | 85.4 |
| **10-10** | - | 373.8 | 295.8 |

As in the single spike input case, the choice of hidden layer size and learning rate can drastically affect convergence speed. It can be seen that under the multiple input spike scenario, there are more configurations where the proposed learning rule fails to correctly train the SNN. This could indicate that this task is more difficult to learn than the single spike input scenario. However, under the correct choice of hidden layer configuration and learning rate, the proposed algorithm was still able to train the SNN to output the desired spike train. Overall, convergence was slower than in the single input

spike case. These two experiments suggest that sparsely spiking inputs to the SNN can simplify the learning task and increase convergence speed. This implication seems to also be reflected in the encoding experiments from chapter 3; the population and threshold encoding schemes led to the highest performance and those encoding schemes tend to spike sparsely in time (for a given neuron) when compared with the rate-based encoding schemes.

### *4.2.3 XOR Problem*

To represent the inputs for this XOR experiment, a low-rate spike train was chosen to represent the value 0 and a maximum rate spike train was chosen to represent the value 1 over 50ms, similar to the XOR problem that the BP-STDP

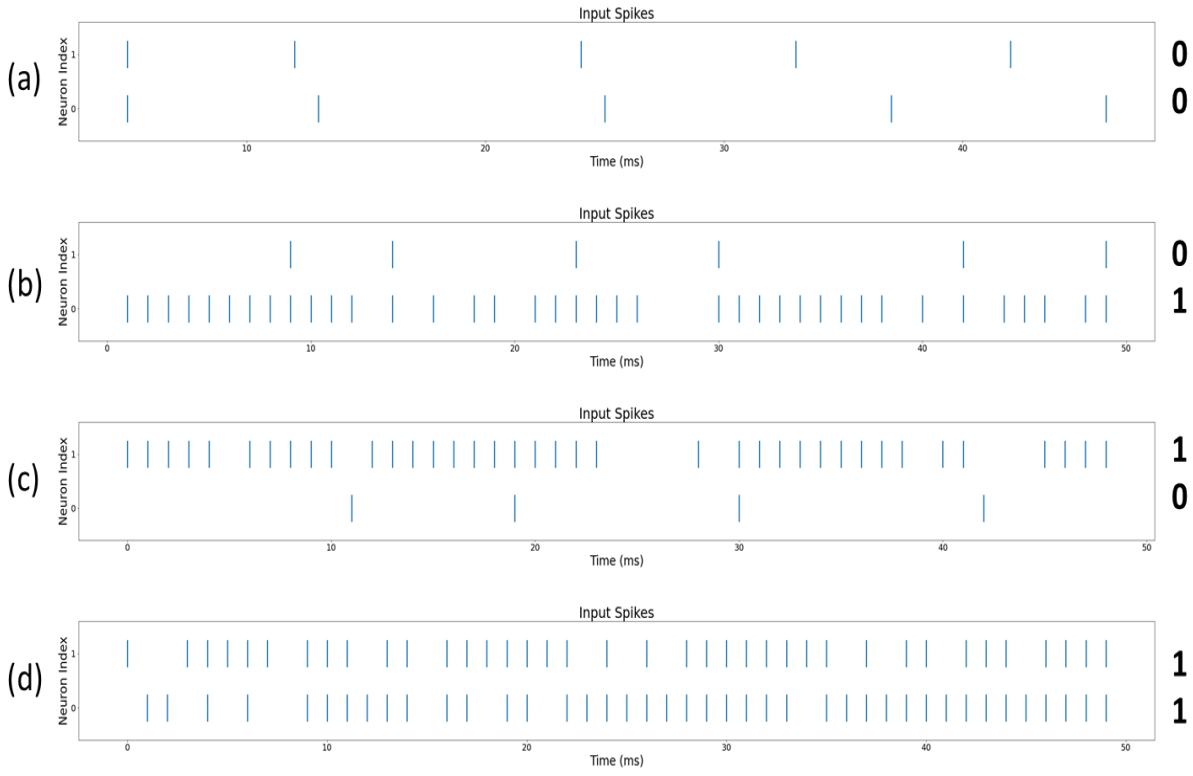



*Figure 4.5: XOR problem inputs. Example input spike trains used to train an SNN with the proposed learning algorithm are shown. (a),(b),(c), and (d) represent the four possible binary input combinations for the XOR problem.*

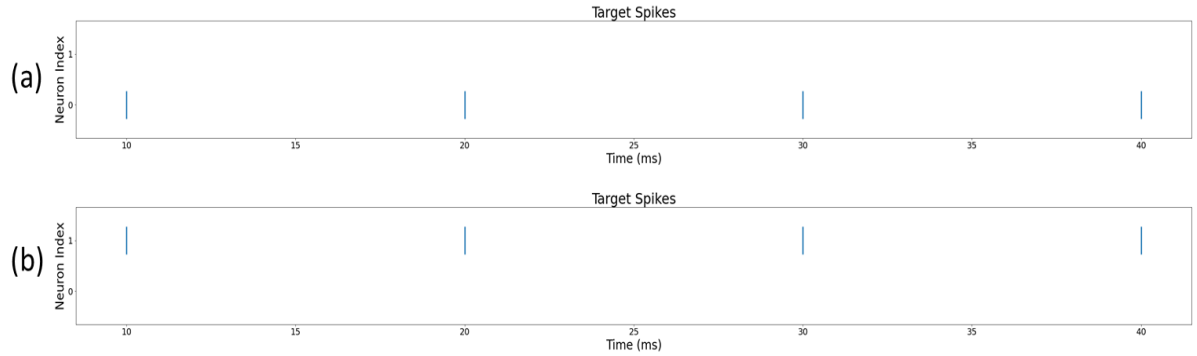



*Figure 4.6: Target output spike trains for the XOR problem. Neuron 0 is assigned to represent the output value '0', and neuron 1 is assigned to represent output value '1'. The output value is determined by which neuron is spiking more frequently, so when the output value is '0' the spike train in (a) is used as the target and (b) is used when the desired value is '1'.*

algorithm was tested on [36]. To represent '0', the input spike train was set to spike approximately every 10 timesteps and the representation for '1' was set to spike approximately every timestep. The input spike trains were generated using a random Poisson spiking process, so each input presentation was slightly different. An example set of input spike trains is shown in Figure 4.5. An SNN with 20 hidden neurons and 2 output neurons was trained to solve the XOR problem. Two output neurons were chosen such that one would be responsible to fire to represent a '0' output and the other is chosen to fire to represent a '1' output. The target spike trains used in this experiment are shown in Figure 4.6. Since the spike train representing a value of '0' spikes approximately once every 10 timesteps, the target training signal was also set to spike every 10 timesteps. This was done so that the '0' values would be able to contribute to every target spike. If the target train was set to fire faster than the maximum inter-spike interval for the '0' representation, there could be many timesteps where the network would adapt its weights solely to the high firing input. A set of output spikes after training is shown in Figure 4.7. It was found that the network was not able to exactly match the

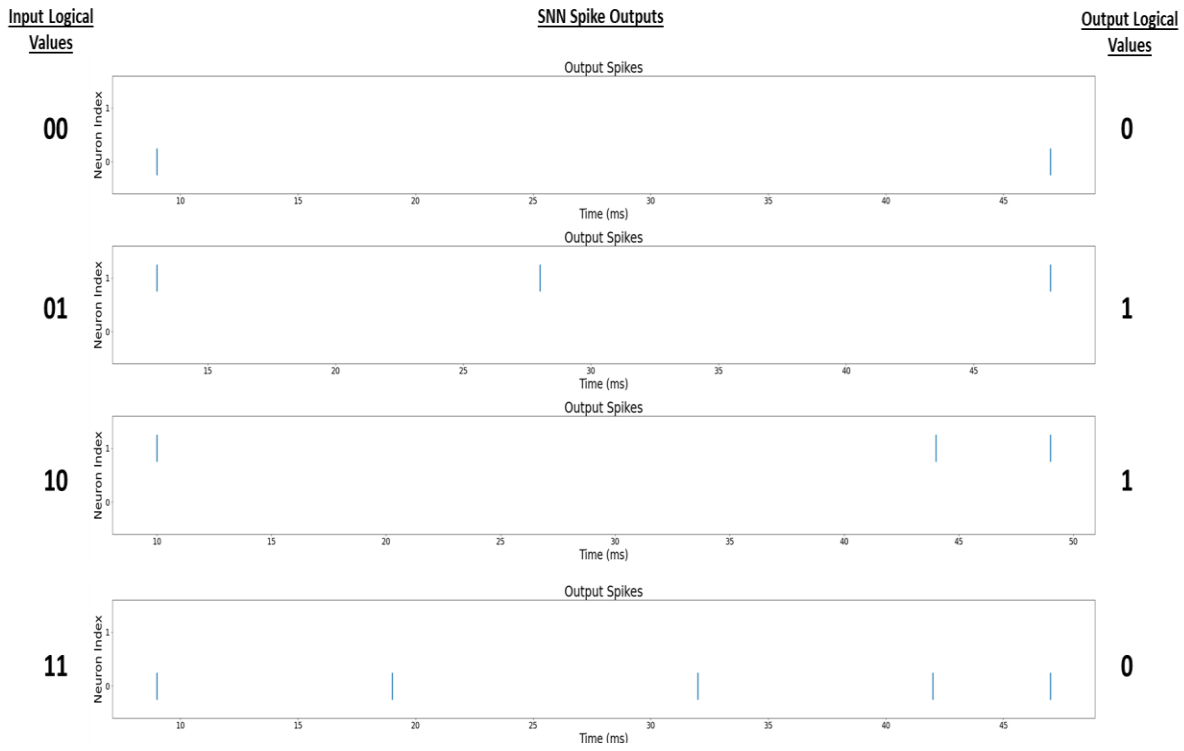



*Figure 4.7: XOR problem outputs after training. SNN output spikes given the four logical XOR inputs encoded as spike trains (not shown). For each of the four inputs, the SNN was successfully able to spike appropriately corresponding to the correct output logical values.*

target signals, however the correct neurons did learn to spike at a similar rate to the target signals; this could be due to the randomness of the input spike trains. After training for approximately 50 epochs, the two neurons could differentiate the input spike patterns based on output spike counts (which is sufficient for classification or identifying the ‘0’ or ‘1’ output correctly). However, there were still many spikes in the off-class neuron. After approximately 300 epochs, the off-class neurons were mostly silent during patterns corresponding to the other output value. Depending on the randomness in the input spike trains, there could be a few off-class spikes however the neuron representing the correct class still reliably spiked more than the off-class neuron after training. This could indicate that while learning specific target spike train patterns may

take a long time, classifications can be learned relatively fast based on spike counts. This is the approach taken for the classification tasks in Chapter 5 of this thesis, where classification decisions are made based on the neuron which fires most during pattern presentation, while still allowing off-class neurons to spike. In addition to simplifying the classification task for the SNN, this learning scheme could be useful if the classes of the input data share certain features which might appear identical over small time windows to the SNN (for example phonemes shared between spoken words).

# 5 Sound Classification

In this chapter, the proposed learning rule introduced in Chapter 4 will be applied to four different datasets. Specifically, the IRIS, FSDD, RWCP, and UrbanSound8k datasets will be used to test the proposed learning rule. The IRIS dataset is not an audio dataset, however it is included as it has been used as a benchmark for similar SNN learning rules. The other three datasets consist of different types of audio data. The FSDD contains spoken digits, the RWCP dataset is made up of isolated non-speech sounds, and the UrbanSound8k dataset contains raw audio consisting of urban environmental sound classes [27]. For comparisons, a SNU network of the same architecture that was used to test the proposed learning rule will also be applied to these datasets [21]. Since the SNU network allows for neuron threshold and membrane time constants to be trained via BPTT, two versions will be used for comparison: a default network where all parameters are trainable, and a restricted network where the threshold and membrane time constant are fixed to more closely match the proposed learning rule's network. This is done because the proposed learning rule does not have temporally local method to update the threshold or membrane potential time constant, and those values are set manually. A few SNN learning rules have been tested on the IRIS and the RWCP datasets, so the results reported in those approaches will be included as well for comparison. Finally, an ablation study is conducted using the FSDD to test the effect of various components of the proposed learning rule on final classification accuracy.

## 5.1 SNN Classification Set-Up

For the purposes of classification, the target spike train used to train an SNN with the proposed algorithm will be set to fire at a high rate. The output layer of the SNN classifier will be set to have as many neurons as there are classes for the given dataset, and each neuron will be responsible for firing when it's corresponding class is presented as an input to the SNN. Then, the classification will be made by assigning a label corresponding to whichever neuron fired the most during the sample presentation. Since not every single timestep will contain useful information for classification, it is difficult to train a given output neuron to exactly fire in every timestep as would be intended with the target signal chosen this way. Alternatively, choosing other target spike trains would also be difficult without in depth knowledge of each individual sample (for example, assigning the targets to spike in a certain region where there is high input activity). To relax this condition for the purposes of classification, at the end of a trial if the correct neuron spiked most the network is considered to be "trained" correctly for that sample and the weight updates accumulated over the trial are not applied since it is already correctly classifying the input. However, this can lead to situations where the given class may only be winning by a small number of spikes. To remedy this, the weight updates will stop being applied once the desired output neuron spikes a certain amount more than any other output neuron. For the experiments in this chapter weight updates will stop being applied once the target neuron spikes at least 1.5x more than any other output neuron. This simplifies the learning problem for the network, and it also allows off-class neurons to spike as long as they spike less than the neuron responsible for the true class. This could help in cases where the input samples have shared

features, for example if common phonemes are shared between different speech recognition classes. Unless otherwise stated, the experiments using the proposed learning rule in this chapter use a small weight decay of $10^{-4}$ which is applied at the end of a pattern presentation and Adam optimization was used [74]; these changes helped improve learning stability and convergence speed. Similar to the RTRL algorithm, the proposed learning rule can be applied online where there is a weight update every timestep, or the proposed weight updates can be accumulated per-timestep and applied after the pattern presentation is over [22]. The second scheme will be used for the experiments in this chapter, as it was found to converge to a better solution in less time. Furthermore, as outlined in [18], this training scheme is gradient-equivalent to BPTT if the network is using a gradient-exact algorithm; the proposed learning rule is not an exact gradient descent algorithm due to various approximations, however maintaining gradient-equivalence where possible is desirable.

## 5.2 Evaluation Metrics

To evaluate the classification performance on the audio datasets in this chapter, four metrics will be used. Overall classification accuracy will be the primary metric used for comparison, which is simply the number of correctly labelled samples divided by the total number of tested samples. This is a valid choice since the datasets used in this chapter have relatively balanced support between their classes. Precision, recall, and F1-score will also be calculated per class, and a confusion matrix will be generated to visualize classification performance [82]. Precision and recall can be calculated as follows:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{5.1}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{5.2}$$

Precision can be thought of as a value that represents how reliable the classifier's output predictions are, and recall can be thought of as a value that represents how well a given class is predicted by the classifier. The F1-score is the harmonic mean of precision and recall, and can be calculated as follows:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.3}$$

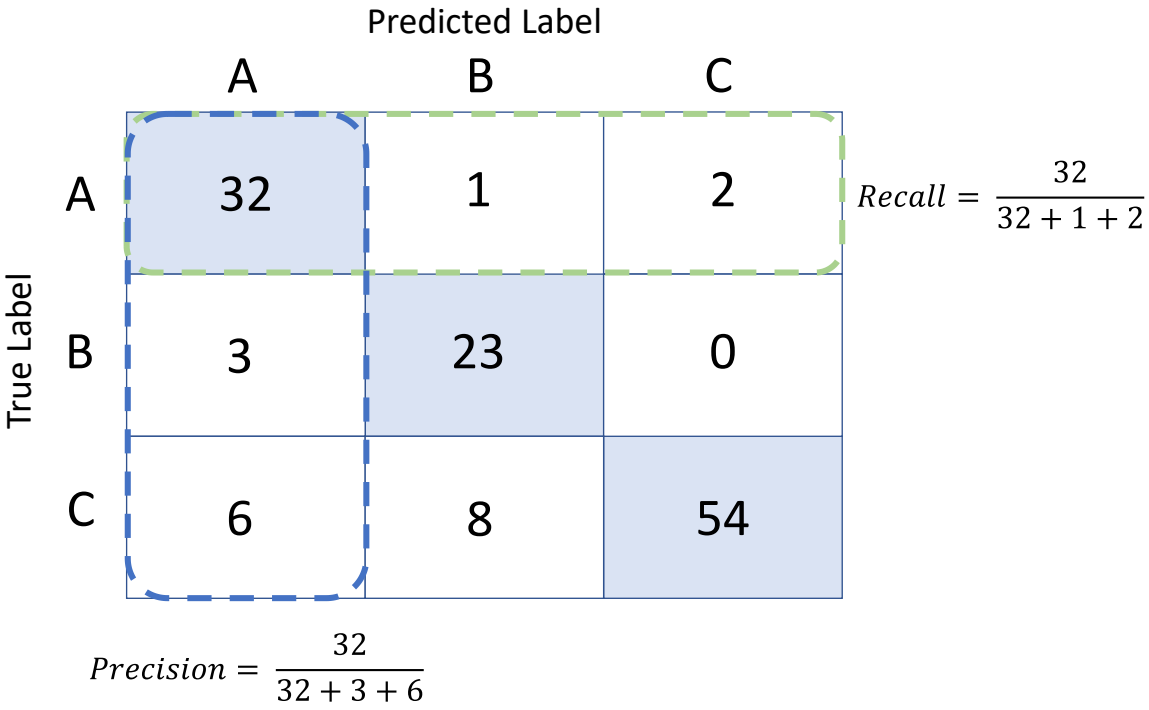



*Figure 5.1: Precision and recall example calculation. Showing calculations for example class 'A'.*

The F1-score is useful when datasets have significantly unbalanced data, in which case it is preferable to accuracy measurements since it accounts for both precision and recall quality whereas accuracy could be artificially high, for example, by simply predicting the majority classes every time. In the case of multi-class classification, precision and recall will be calculated per-class where all other classes are considered the “off” class. In this case, false positives will be the sum of all other classes incorrectly classified as the class under question and false negatives will be the sum of all the instances when the class under question is misclassified as a different class. An example confusion matrix with corresponding precision/recall calculations for multi-class classification is shown in Figure 5.1. In this figure, an example calculation of precision and recall for the class ‘A’ is shown.

## 5.3 IRIS Dataset Experiments

Fisher’s IRIS dataset is a classical dataset used to benchmark various classifiers [24]. Although it is relatively small and simple by today’s standards, it is still useful for testing SNN classifiers since only recently have SNN learning methods begun to reach the performance of ANN classifiers on this dataset. Furthermore, it will act as a test to see the performance of the proposed learning rule on a non-temporal dataset which has been spike encoded. As described in Chapter 3, the IRIS dataset is made up of 3 different classes of IRIS flowers which are represented by four length measurements, and each class has 50 samples. Based on the observations in the Chapter 3 encoding experiments, a population latency spike encoding scheme will be used for these experiments. A time window of 20 timesteps is used and a population of 20 neurons with overlapping gaussian receptive fields will be applied to convert each individual

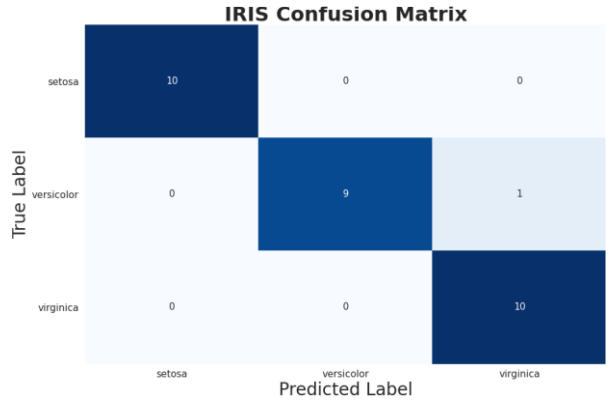
input value into spike trains. Since there are four input values per sample, this gives a total input dimension of 80 input neurons. An SNN with a 50-unit hidden layer and a 3-unit output layer will be trained. For training via the proposed learning rule, the target train was set to spike every 5 timesteps, Ƞ = 0.5, and $\tau_e$ = $\tau_m$ = 20ms. The time constants were chosen this way since the entire 20 timestep window is used to encode the input values under the population latency encoding scheme. Since the population latency encoding scheme leads to relatively sparse spiking in time, it was found that lowering the target spike train rate and increasing the time constants such that the learning window was able to span multiple input spikes was necessary. To evaluate classification performance over this dataset, a 5-fold cross-validation is repeated 5 times where each fold used 120 samples for training and 30 samples for validation. The reported results in Table 5.1 are the average validation accuracy over the five repetitions of the 5-fold cross validation after 100 epochs. Results from [36] for the BP-STDP and an ANN are included for comparison, and so are the results from the Multi-ReSuMe algorithm [20].

*Table 5.1: Classification performance on the IRIS dataset.*

| **Classifier** | **Validation Accuracy** |
|---|---|
| ANN (30 hidden neurons) [36] | 96.7% |
| Multi-ReSuMe [20] | 94.0% |
| BP-STDP [36] | 96.0% |
| Proposed Learning Rule | 95.6% |
| Matched SNU | 95.1% |
| Full SNU | 95.6% |

As can be seen from Table 5.1, the proposed learning algorithm was able to out-perform the Multi-ReSuMe algorithm. Since part of the motivation for the proposed rule was to account for hidden-layer neuron activations directly in contrast to the Multi-ReSuMe, this is a promising result. However, it underperforms when compared to the ANN and BP-STDP approaches. The BP-STDP approach applied rate encoding schemes in their experiments, which seem to work well when paired with IF neurons which can act as spike counting neurons. The proposed learning rule and the SNU networks used here are applied to temporally encoded spike patterns, so although they underperformed slightly when compared with the BP-STDP rule combined with rate encoded input, they were still able to show that a competitive classification can be achieved using temporally encoded data with multilayer SNNs composed of LIF neurons. When comparing the proposed rule with the two SNU approaches, it can be seen that when the SNU network is matched to the proposed algorithm that the proposed algorithm performs slightly better. On this dataset, the proposed algorithm was able to match the performance of the fully trainable SNU. Since both SNU networks are trained using full BPTT, this indicates that the proposed learning rule can be successful in approximating BPTT in multilayer SNNs for classification tasks. An example confusion matrix and detailed evaluation metrics from one fold for the proposed learning rule are shown in Figure 5.2.

The performance seen in Figure 5.2 is similar to the performance achieved with standard ANN approaches, where the versicolor and virginica classes are commonly mis-classified [83][84]. However, with the proposed approach when the cross-validation




| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Setosa | 1.0 | 1.0 | 1.0 | 10 |
| Versicolor | 1.0 | 0.9 | 0.95 | 10 |
| Virginica | 0.91 | 1.0 | 0.95 | 10 |

*Figure 5.2: Confusion matrix and detailed classification performance for the proposed algorithm on the IRIS dataset.*

is performed multiple times with randomized folds, there are more splits where 2 or (rarely) 3 validation samples are mis-classified at the end of the cross-validation which leads to the slightly lower average accuracy reported in Table 5.1. So although the SNN under the proposed learning rule is outperformed by the ANN approach on average, it is worth noting that the trained SNN makes similar errors to the ANN approach (confusing virginica/ versicolor) even though it works on a temporal spike encoded version of the dataset.

## 5.4 FSDD Experiments

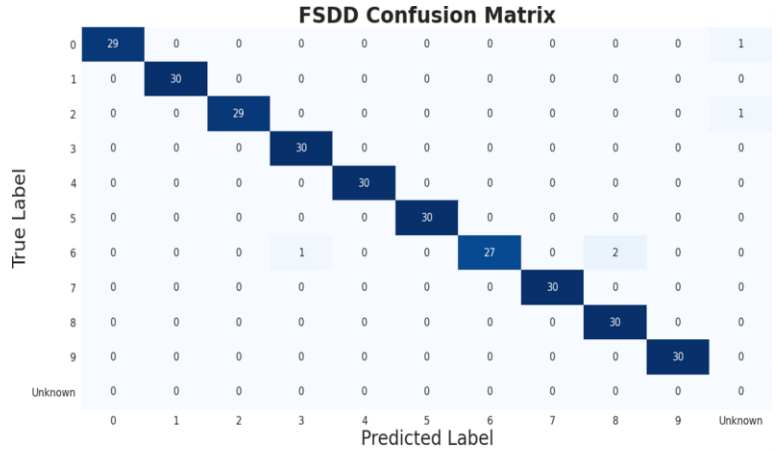
As outlined in Chapter 3, the Free Spoken Digit Dataset (FSDD) is a publicly available open audio dataset consisting of six different speakers saying the digits 0-9 fifty times each for a total of 3000 recordings [25]. For the experiments in this section, the training-test split outlined by the authors of the dataset is used which sets 90% of the data for training and 10% for testing. All audio recordings are converted to log-Mel spectrograms as described in Chapter 3, after which a threshold spike encoding scheme is applied with 15 thresholds per frequency band. A feedforward SNN with 300

hidden units and 10 output units was then trained to classify the 10 spoken digits using the proposed algorithm as well as the SNU. The reported results are the average of three runs of 100 epochs each. For the proposed algorithm, the target spike train was set to spike in every timestep. As opposed to the IRIS dataset with the population latency encoding scheme, the threshold encoding scheme converts the input data into spikes in each timestep so most timesteps that span the input data carry information. The time constants were set to $\tau_e$ = 5ms and $\tau_m$ = 25ms, and the learning rate was Ƞ = 0.01. The classification results are reported in Table 5.2.

*Table 5.2: Validation accuracies on the FSDD.*

| **Classifier** | **Validation Accuracy** |
|---|---|
| Proposed Learning Rule | 98.0% |
| Matched SNU | 97.3% |
| Full SNU | 99.3% |

In these experiments, it is shown that the proposed learning rule can achieve high classification accuracy on the spoken digits in this dataset. The final accuracy lags behind the full SNU network trained via BPTT, however when the threshold and membrane time constant values of the SNU network were matched to the proposed network the proposed learning rule began to outperform the SNU network. This indicates that for audio and temporal classification tasks, allowing the membrane potential time constants to be trainable could be beneficial. This experiment also shows that audio classification can successfully be performed using temporally local learning rules in a multilayer SNN as a replacement for BPTT. A confusion matrix and detailed evaluation metrics resulting from the proposed learning rule are shown in Figure 5.3. In




| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 1.00 | 0.97 | 0.98 | 30 |
| 1 | 1.00 | 1.00 | 1.00 | 30 |
| 2 | 1.00 | 0.97 | 0.98 | 30 |
| 3 | 0.97 | 1.00 | 0.98 | 30 |
| 4 | 1.00 | 1.00 | 1.00 | 30 |
| 5 | 1.00 | 1.00 | 1.00 | 30 |
| 6 | 1.00 | 0.90 | 0.95 | 30 |
| 7 | 1.00 | 1.00 | 1.00 | 30 |
| 8 | 0.94 | 1.00 | 0.97 | 30 |
| 9 | 1.00 | 1.00 | 1.00 | 30 |

*Figure 5.3: Confusion matrix and detailed classification performance for the proposed algorithm on the FSDD.*

addition to a few misclassifications for digit ‘6’, there were two samples which were not assigned a class for the true classes ‘0’ and ‘2’. This type of error can occur under the proposed classification scheme when two output neurons emit the exact same number of spikes for a given input sample or no spikes are emitted at all. This behavior is most likely due to the per-timestep squared error loss used in the proposed algorithm, and it could potentially be addressed by using a more sophisticated loss function or introducing some form of winner-take-all mechanism in the output layer.

## 5.5 RWCP Dataset Experiments

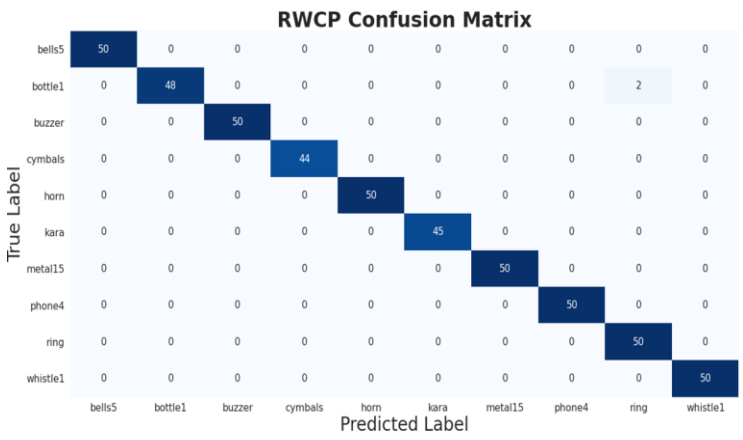
The Real World Computing Partnership (RWCP) dataset contains isolated non-speech sounds recorded in an anechoic chamber with a sampling rate of 16kHz and a duration up to 4 seconds long [26]. This dataset has been used to evaluate other SNN classifiers for audio classification [42][85][86][87]. These works use a subset of the RWCP dataset which contains the following ten classes: ‘cymbals’, ‘horn’, ‘phone4’, ‘bells5’, ‘kara’, ‘bottle1’, ‘buzzer’, ‘metal15’, ‘whistle1’, and ‘ring’. Each of these classes

contain between 80 and 100 samples. Compared to the FSDD, the classes in this dataset cover a wider range of frequencies and durations. This could simplify the classification task in the sense that these classes may occupy more distinct frequency/time regions, however it also requires the classifier to be robust to a wider range of inputs. As in the other papers which used this dataset for audio classification with SNNs, each class will be divided randomly in half to use 50% of the data for training and 50% for testing. The reported results are the average of three runs of 100 epochs each. To test the proposed learning rule, an identical SNN to the one used in the FSDD experiments is trained. Similarly, two SNU networks are trained; one is matched to allow the same parameters to be trained as the proposed SNN, and another where all the parameters are trained via BPTT. Results from [85] and [88] are included for comparison. These approaches use unsupervised STDP and self-organizing map (SOM) feature extraction steps respectively before applying a single layer tempotron SNN classifier [18]. The results on the RWCP classification task are shown in Table 5.3.

*Table 5.3: Validation accuracies on the RWCP dataset.*

| **Classifier** | **Validation Accuracy** |
|---|---|
| MLP [88] | 99.5% |
| STDP-Tempotron [85] | 99.2% |
| SOM-Tempotron [88] | 99.6% |
| Proposed Learning Rule | 99.6% |
| Matched SNU | 98.8% |
| Full SNU | 99.8% |

The proposed learning rule worked quite well on the RWCP dataset. Once again it outperformed the matched SNU network, and it performed slightly worse than the fully trainable SNU network. The proposed learning rule also matched the performance of the SOM-Tempotron, and slightly outperformed the ANN and STDP-Tempotron approaches. This result is significant since the SOM-Tempotron and STDP-Tempotron approaches make use of a separate unsupervised feature extraction step before applying a more direct gradient based learning rule for SNNs via the Tempotron learning rule (which is limited to a single layer network). The proposed approach successfully matched these approaches with a multilayer network where the feature extraction could be considered to be jointly trained along with the final classifier. A confusion matrix and detailed evaluation metrics for the proposed learning rule are shown in Figure 5.4. For the run shown in this figure, the only classification errors were two ‘bottle1’ samples being misclassified as ‘ring’. The specific classification errors could vary slightly between runs, however there were consistently between 1 and 3 misclassifications after 100 epochs.




| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| bells5 | 1.00 | 1.00 | 1.00 | 50 |
| bottle1 | 1.00 | 0.96 | 0.98 | 50 |
| buzzer | 1.00 | 1.00 | 1.00 | 50 |
| cymbals | 1.00 | 1.00 | 1.00 | 44 |
| horn | 1.00 | 1.00 | 1.00 | 50 |
| kara | 1.00 | 1.00 | 1.00 | 45 |
| metal15 | 1.00 | 1.00 | 1.00 | 50 |
| phone4 | 1.00 | 1.00 | 1.00 | 50 |
| ring | 0.96 | 1.00 | 0.98 | 50 |
| whistle1 | 1.00 | 1.00 | 1.00 | 50 |

*Figure 5.4: Confusion matrix and detailed classification performance for the proposed algorithm on the RWCP dataset.*

## 5.6 UrbanSound8k Experiments

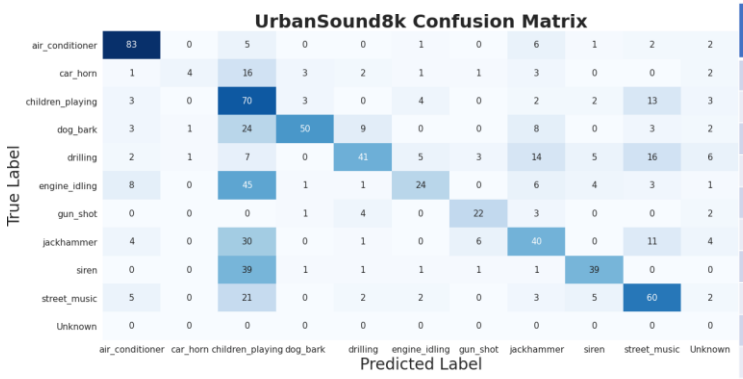
The final dataset used to evaluate the proposed learning rule is the UrbanSound8k dataset [27]. This dataset contains 8732 labeled sound clips up to 4s in length. The audio clips have varying sampled rates, so for these experiments all audio clips were re-sampled to 16kHz. This dataset contains 10 urban sound classes: ‘air_conditioner’, ‘car_horn’, ‘children_playing’, ‘dog_bark’, ‘drilling’, ‘enginge_idling’, ‘gun_shot’, ‘jackhammer’, ‘siren’, and ‘street_music’. This is a much more difficult dataset for classification compared with the previous two audio datasets. The sound events of interest can happen anywhere within the 4s of audio clip, and various background sounds and other noise can be present at any point throughout the clip. To the best of the author’s knowledge, at the time of writing, no other SNN classifiers have been applied to this dataset. Most SNN research for audio classification has used simpler datasets similar to the FSDD or RWCP datasets, whereas the UrbanSound8k dataset is challenging even for large ANNs. To test the proposed algorithm, an identical SNN that was used for the FSDD and RWCP datasets is trained. Since this task is more challenging, another SNN with 1000 hidden units is also trained on this dataset. A

*Table 5.4: Validation accuracies on the UrbanSound8k dataset.*

| Classifier | Validation Accuracy |
|---|---|
| Proposed Learning Rule (300 Hidden) | 49.5% |
| Proposed Learning Rule (1000 Hidden) | 51.7% |
| Matched SNU (1000 Hidden) | 52.3% |
| Full SNU (1000 Hidden) | 58.2% |

training/test split of 90%/10% was used for these experiments. As was done for the FSDD and RWCP experiments, two SNU networks are trained for comparison. The results for the UrbanSound8k classification task are shown in Table 5.4.

The results in Table 5.4 clearly show that as the classification task becomes more challenging the SNN classifiers begin to perform poorly. Even the fully trainable SNU network only managed to achieve 58.2% accuracy on this dataset. As seen in the previous experiments the proposed learning rule is outperformed by the fully trainable SNU network, and in this case it performs similarly to the matched SNU network where the membrane potential and threshold aren't trainable. The networks used for this experiment are relatively simple however, as they are basic feedforward networks. ANNs which achieve state of the art performance on this dataset are usually more architecturally advanced than the feedforward SNNs tested here, making use of very deep CNNs or LSTMs (or both combined). Expanding the proposed learning rule to such advanced architectures could be an avenue for future study. A confusion matrix and detailed evaluation metrics for the proposed learning rule with 1000 hidden neurons are shown in Figure 5.5. It is clear from this figure that classification is poor for every




| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| air_conditioner | 0.76 | 0.83 | 0.79 | 100 |
| car_horn | 0.67 | 0.12 | 0.21 | 33 |
| children_playing | 0.27 | 0.70 | 0.39 | 100 |
| dog_bark | 0.85 | 0.50 | 0.63 | 100 |
| drilling | 0.67 | 0.41 | 0.51 | 100 |
| engine_idling | 0.63 | 0.26 | 0.37 | 93 |
| gun_shot | 0.67 | 0.69 | 0.68 | 32 |
| jackhammer | 0.47 | 0.42 | 0.44 | 96 |
| siren | 0.70 | 0.47 | 0.56 | 83 |
| street_music | 0.56 | 0.60 | 0.58 | 100 |

*Figure 5.5: Confusion matrix and detailed classification performance for the proposed algorithm on the UrbanSound8k dataset.*

class; the highest F1-score is only 0.79 for the ‘air conditioner’ class and can drop as low as 0.21 in the case of ‘car horn’. An additional complexity that is introduced with this dataset is large intra-class variability; on the FSDD and RWCP datasets, the samples within a given class were quite similar to each other. On this dataset, classes such as ‘children_playing’ and ‘street_music’ can contain a wide variety of sounds within their own class, and they may contain a lot of overlap with other classes in the time/frequency domain. This may be a reason for their relatively low precision score. Combined with the off-class background sounds present in this dataset, it is clear that more robust feature extraction is required for high classification performance on this dataset.

## 5.7 Ablation Study

To test the effect of various portions of the proposed learning rule, a few ablation experiments were conducted. These experiments were conducted using the FSDD, as it is more challenging than the RWCP dataset while still achieving relatively good accuracy with the proposed learning rule. Table 5.5 shows the effect of hidden layer size on classification accuracy for the FSDD; after a size of approximately 300 hidden units, classification accuracy did not seem to improve significantly. For the rest of the experiments in this section the learning rate, network architecture, and time constants are the same as in the initial FSDD experiments unless otherwise stated. Seven ablation experiments are included for comparison, including the original experiment from earlier in this chapter on the FSDD. These experiments are referred to by their experiment number in Table 5.6.

*Table 5.5: Performance with different hidden layer sizes on the FSDD.*

| Hidden Layer Size | Validation Accuracy |
|---|---|
| 10 | 63.3% |
| 100 | 95.3% |
| 300 | 98.0% |
| 1000 | 98.3% |

*Table 5.6: Ablation study results on the FSDD.*

| Experiment # | Classifier | Validation Accuracy |
|---|---|---|
| 1 | Default | 98.0% |
| 2 | No e-trace scaling (straight-through estimator) | 90.0% |
| 3 | Using e-prop pseudo-derivative | 99.0% |
| 4 | 3-layer (300h-300h) | 95.0% |
| 5 | Frozen Input-Hidden Layer Weights | 41.0% |
| 6 | No win-by-1.5x condition | 92.0% |
| 7 | No win-by-1.5x condition, no e-trace error scaling | 83.0% |

To test the main modification of the proposed algorithm, experiment 2 removes the scaling of the propagating error signal by the eligibility traces and replaces it with the standard straight-through estimator. Experiment 3 tests the effect of using the voltage-based pseudo-derivative from e-prop in place of the proposed eligibility trace. Then, to test that the learning rule works for a network with more than just a single hidden layer, experiment 4 tests a 3-layer network that is trained with two hidden layers of 300 hidden neurons. To ensure that the proposed learning rule is not just training the final output layer to solve the task, experiment 5 tests a network with a single hidden layer but the input-to-hidden layer weights are frozen. This is done to see if the proposed rule is training lower layers meaningfully or if it is just training the output layer to learn random projections of the input data. Finally, in experiments 6 and 7, the condition used for classification where weight updates are stopped if the correct neuron spikes 1.5x more than any other neuron is removed and the proposed weight updates are applied

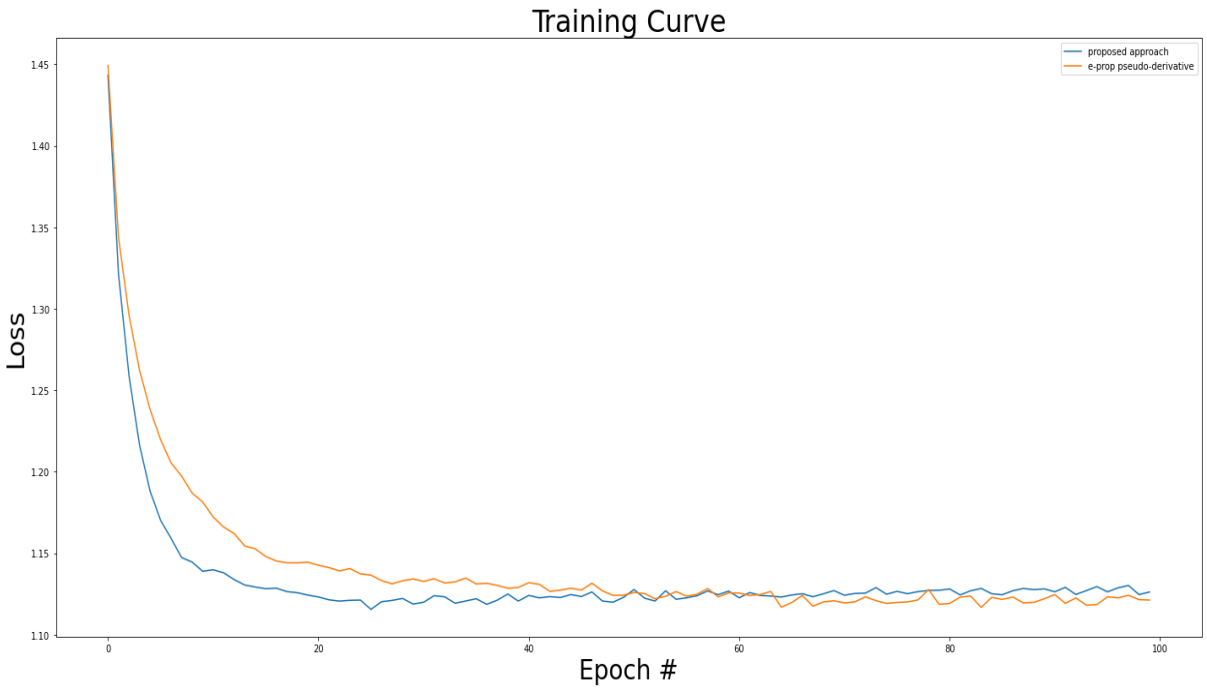



*Figure 5.6: Training curve for the proposed algorithm (blue) vs the proposed algorithm using e-prop pseudo-derivative (orange) on the FSDD.*

unconditionally. Experiment 7 additionally replaces the eligibility trace error scaling with the straight-through estimator as in experiment 2 for comparison.

As can be seen from comparing experiments 1 vs 2 and 6 vs 7 in Table 5.6, scaling the backpropagating error signal by the eligibility traces does seem to significantly improve the network's performance when compared to using the linear straight-through estimator. In both cases there is approximately an 8% decrease in final classification accuracy when the proposed change is removed. In Experiment 3, substituting the proposed eligibility trace pseudo-derivative with the pseudo-derivative from e-prop led to an increase in accuracy. Figure 5.6 shows the training curve for the proposed algorithm and the modified version that uses the e-prop pseudo-derivative. This figure shows that the proposed algorithm converges faster than the version with the e-prop pseudo-derivative, although it achieves slightly lower accuracy. However,

since the proposed eligibility trace pseudo-derivative was motivated by the biological STDP process and the e-prop pseudo derivative was heuristically designed as an approximation to the spiking function derivative, this supports the idea that the gradient of the spiking function can be approximated by the temporal eligibility trace. Experiment 4 shows that the accuracy decreased slightly when moving to a 3 layered network, however it also shows that the proposed algorithm can still work for deeper SNNs. In experiment 5, the classification accuracy decreased significantly to 41%. This shows that the proposed algorithm does indeed train lower layers in the network and it's not simply learning random projections of the input data at the final layer. Finally, experiment 6 shows that the network can still learn without the win by 1.5x condition. However, it performs worse since the classification problem becomes much more challenging. Without the win by 1.5x condition, the network is trained so that off-class neurons do not spike at all and the on-class neuron is required to spike exactly according to a target signal which was set to spike every 10 timesteps. As mentioned earlier, since it is hard to know exactly when class-relevant features occur in the input data assigning a target spike in every timestep (or even every 10 timesteps as was done here) may be a poor choice without relaxing the classification problem with something like the win by 1.5x condition. That being said, the outputs of the network trained without the win by 1.5x condition are generally cleaner than the outputs of the default network when there is a correct classification. However, the overall number of correct classifications is lower without the win by 1.5x condition. Figure 5.7 shows a comparison between a correct classification under both training schemes, and an incorrect classification under both schemes. Another interesting result from this

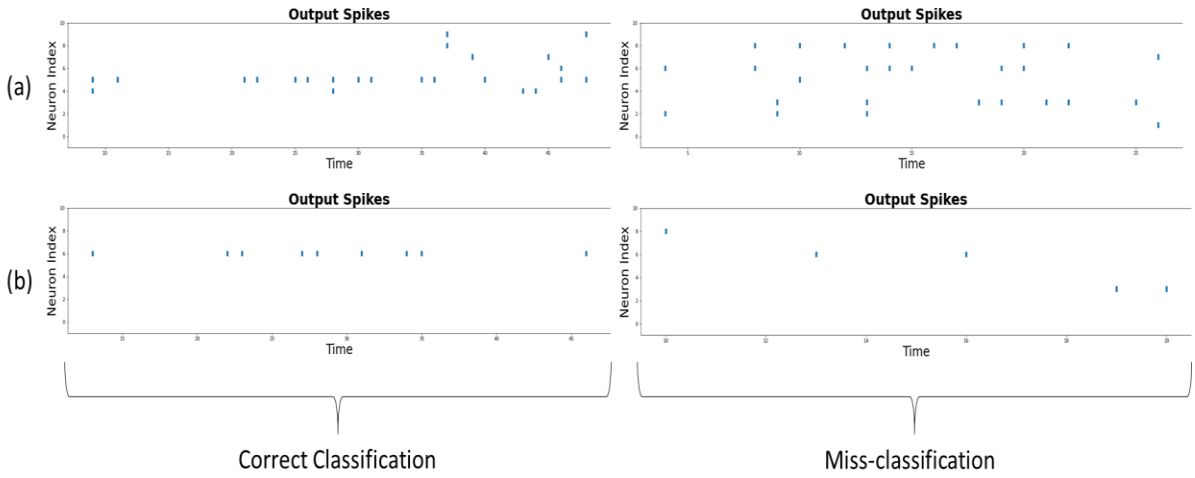



*Figure 5.7: Comparison of illustrative SNN outputs trained via the proposed learning rule. (a) shows the outputs when the network is trained with the win by 1.5x condition, and (b) shows the outputs when this condition is removed.*

experiment is that even though the accuracy from experiment 6 is ~6% worse than experiment 1, it is still higher than the experiments where the eligibility trace scaling is removed from the backwards pass. This further shows the improvement gained from the proposed algorithm.

## 5.8 Discussion

Across all of the datasets tested in this chapter, the proposed learning algorithm showed competitive performance when compared to similar learning rules and networks. For the RWCP dataset, the proposed algorithm matched the performance of the STDP-Tempotron and SOM-Tempotron approaches to sound classification in [85] and [88]. However, unlike these two approaches, the proposed rule did not require an unsupervised feature extraction step before the classification step. Instead, a multilayer SNN was used in place of the single layer Tempotron classifier to jointly learn both the classification and perform additional feature extraction. The proposed SNN performed poorly on the UrbanSound8k dataset in terms of absolute accuracy, however it was

comparable to an equivalent SNU network and ~8% worse than a fully trainable SNU network. This indicates that the limiting factor on this dataset may be the chosen architecture and not necessarily the learning rule. A common pattern across all four datasets was that the proposed algorithm outperformed or performed comparably to a matched SNU network. This is a significant observation because the SNU network is trained by taking full advantage of BPTT, whereas the proposed algorithm was designed specifically to avoid BPTT and approximate it in a temporally local manner. The SNU network that allows the membrane potential time constants and thresholds to be trained via BPTT consistently outperformed the proposed learning rule, however when the SNU network was limited to only train the values that the proposed algorithm updates, they were roughly equivalent. This indicates that the proposed algorithm could be further improved to also adjust thresholds and membrane potential time constants. Developing an update rule for these values may not be straightforward under the proposed learning scheme, however. This is because the proposed learning rule takes advantage of STDP-like interactions which is an update policy for synaptic efficacies (connection weights); it is unclear if a similar mechanism could be used to update thresholds and membrane potential time constants.

The experiments in the ablation study section of this chapter help support the ideas behind the proposed algorithm. The biggest change suggested by the proposed algorithm was to scale the backpropagating error signal by the spike time eligibility traces; when this was replaced by the linear straight through estimator, accuracy decreased significantly by ~8%. This section also showed that the proposed algorithm can work for deeper SNNs, and that it meaningfully trains the lower layers to help with

classification. When the pseudo-derivative from e-prop was used in the backwards pass, there was an increase in performance in exchange for slower training convergence. However, since the e-prop learning rule was designed to directly estimate the gradient for backpropagation and the proposed learning rule was inspired by the STDP process, this may be an indication that the gradient can reasonably be estimated by STDP or similar processes.

# 6 Conclusion

## 6.1 Summary

This thesis set out to apply SNNs to audio classification tasks. Many of the applications involving audio classification tend to benefit from deploying audio classifiers to power-limited embedded devices, such as in an IoT sensor network. This was one of the primary motivators for applying SNNs to the audio classification problem, since they consume significantly less power than equivalent ANNs due to their event-based computation. However, SNNs are currently less developed than ANNs and they lag behind in performance. Furthermore, training multi-layer SNNs is still an open research question and there is no single unifying algorithm to approach this problem. Spike encoding is another challenge unique to SNNs, where real-valued data must be converted into temporal spike trains to be compatible with an SNN. To explore some of these issues, this thesis first compared various spike encoding schemes by comparing classification accuracy on the IRIS and FSDD using an SVM and a multilayer SNU network trained via BPTT. It was found that for temporal audio data, a threshold spike encoding scheme seemed to result in the best classification accuracy while also emitting the fewest spikes and by extension potentially consumes less power. For non-temporal data, a population latency encoding scheme seemed to perform the best out of the considered encoding schemes.

After comparing various encoding schemes, a learning algorithm for training multilayer SNNs was proposed. This algorithm suggests scaling a backpropagating error signal by the spike-timing eligibility traces of the neurons in a multilayer SNN. This

was done to implement a form of STDP where the weight updates at the synapses within the network are now dependant on both incoming and outgoing spike timings. This is in contrast to similar learning algorithms such as e-prop which use membrane potential based pseudo-derivatives to make weight updates which may not explicitly account for outgoing spike timings. Rules such as Multi-ReSuMe ignore hidden layer activations altogether. The proposed algorithm can also be thought of extending the BP-STDP algorithm to LIF neurons, where originally it was only applied to IF neurons. In addition to being motivated biologically inspired STDP updates, another reason for applying this scaling to the backpropagating error signal was to apply temporal error credit by tightly coupling the output errors with recently spiking neurons. The practical motivation for this approach was to avoid the BPTT algorithm, which requires off-line propagation backwards through time at the end of a sample presentation to update weight values as well as extra memory to store state values from every timestep. Avoiding BPTT was done in hopes of simplifying training algorithms for deployment to neuromorphic hardware for on-chip and online learning where implementing the full BPTT algorithm may not be possible. In summary, the proposed algorithm attempts to assign temporal error credit based solely on spike timings using variables that propagate forward in time as the network processes the input, eliminating the need to store intermediate variables from every single timestep as would be required in BPTT.

Finally, the proposed algorithm was tested on four datasets: the IRIS dataset, the FSDD, the RWCP dataset, and the UrbanSound8k dataset. For the IRIS and RWCP dataset where there were results available from similar SNN learning rules, the proposed algorithm was able to achieve comparable performance. This is significant

since the other SNN approaches made use of single-layer SNN classifiers with additional feature extraction steps, or they used IF neurons as opposed to LIF neurons. High accuracies were achieved on the FSDD and RWCP audio datasets, which shows that the SNN approach to audio classification can be successful. However, on the UrbanSound8k audio dataset, the performance was quite poor. This dataset presents a much more challenging task to the network since the audio clips are longer and contain many additional sounds which are not part of the desired classes. Due to this, even the SNU networks trained via BPTT performed poorly, indicating that perhaps network architecture could be the limiting factor. It was found that over all four datasets, the proposed learning rule performed comparably to a matched SNU network trained via BPTT, and consistently underperformed when compared to a fully trainable SNU network. This indicates that allowing membrane potential time constants and spike threshold values to be trainable may help improve SNN performance. However, it is unclear whether applying an STDP-like update to time constants or thresholds is justified. Perhaps another biologically motivated mechanism could be applied in this situation. In conclusion, this work presented a framework to train multilayer SNNs for simple audio classification tasks. Various spike encoding schemes were considered, and a temporally local learning rule for multi-layer SNNs was developed. The proposed approach was shown to be successful for simple audio classification tasks, however there is still much room for improvement as it performed poorly on the more realistic UrbanSound8k audio dataset.

## 6.2 Future Directions

In terms of future directions for this work, finding an appropriate learning rule for updating threshold and membrane potential time constants is a top priority. Since the proposed learning rule was based on STDP-like updates and STDP updates are primarily for synaptic efficacies (weight values), they are not necessarily applicable to threshold and time constant updates. Another important area to expand on would be to implement more advanced network architectures such as a CNN or the addition of recurrent connections where output spikes feed back into the inputs of other neurons within the same layer. On the UrbanSound8k dataset even the fully trainable SNU network performed quite poorly, indicating that the biggest performance gains could come from network architecture design. Another interesting extension would be to deploy this approach or similar approaches to neuromorphic hardware, where direct energy expenditure could be measured and compared with ANNs. Online and on-chip learning could also be evaluated. The approach in this work performed an audio pre-processing step to obtain log-Mel spectrograms followed by a specific spike encoding scheme; however, there are devices being developed for machine hearing which output spike trains by default such as the silicon cochlea in [89]. This could be another avenue for future research where such physical sensing devices could be combined with neuromorphic SNN hardware to create a full end-to-end neuromorphic system for machine hearing and audio classification.

# Bibliography


[1] J. Zhao, B. Liang, and Q. Chen, “The key technology toward the self-driving car,” *International Journal of Intelligent Unmanned Systems*, vol. 6, no. 1, pp. 2–20, Jan. 2018, doi: 10.1108/IJIUS-08-2017-0008.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. Accessed: Dec. 20, 2019. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[3] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, “The Microsoft 2017 Conversational Speech Recognition System,” *arXiv:1708.06073 [cs]*, Aug. 2017, Accessed: Apr. 01, 2021. [Online]. Available: http://arxiv.org/abs/1708.06073

[4] M. Crocco, M. Cristani, A. Trucco, and V. Murino, “Audio Surveillance: A Systematic Review,” *ACM Comput. Surv.*, vol. 48, no. 4, p. 52:1-52:46, Feb. 2016, doi: 10.1145/2871183.

[5] S. Santini, B. Ostermaier, and A. Vitaletti, “First experiences using wireless sensor networks for noise pollution monitoring,” in *Proceedings of the workshop on Real-world wireless sensor networks*, New York, NY, USA, Apr. 2008, pp. 61–65. doi: 10.1145/1435473.1435490.

[6] C. Lopez-Tello and V. Muthukumar, “Classifying Acoustic Signals for Wildlife Monitoring and Poacher Detection on UAVs,” in *2018 21st Euromicro Conference on Digital System Design (DSD)*, Aug. 2018, pp. 685–690. doi: 10.1109/DSD.2018.00006.

[7] L. Deng, G. Hinton, and B. Kingsbury, “New types of deep neural network learning for speech recognition and related applications: an overview,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 8599–8603. doi: 10.1109/ICASSP.2013.6639344.

[8] “DCASE2019 Challenge,” *DCASE2019 Challenge*, 2019. http://dcase.community/challenge2019/ (accessed Sep. 06, 2019).

[9] Y. Chen, B. Zheng, Z. Zhang, Q. Wang, C. Shen, and Q. Zhang, “Deep Learning on Mobile and Embedded Devices: State-of-the-art, Challenges, and Future Directions,” *ACM Comput. Surv.*, vol. 53, no. 4, pp. 1–37, Sep. 2020, doi: 10.1145/3398209.

[10] D. Zhao, H. Ma, and L. Liu, “Event classification for living environment surveillance using audio sensor networks,” in *2010 IEEE International Conference on Multimedia and Expo*, Jul. 2010, pp. 528–533. doi: 10.1109/ICME.2010.5583889.

[11] B. Malhotra, I. Nikolaidis, and J. Harms, “Distributed classification of acoustic targets in wireless audio-sensor networks,” *Computer Networks*, vol. 52, no. 13, pp. 2582–2593, Sep. 2008, doi: 10.1016/j.comnet.2008.05.008.

[12] F. Rosenblatt, “The Perceptron—a perceiving and recognizing automaton,” *Cornell Aeronautical Laboratory*, no. Report 85-460-1, 1957.

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

[14] E. Hunsberger and C. Eliasmith, “Training Spiking Deep Networks for Neuromorphic Hardware,” *arXiv:1611.05141 [cs]*, 2016, doi: 10.13140/RG.2.2.10967.06566.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, doi: 10.1038/323533a0.

[16] C. S. Thakur *et al.*, “Large-Scale Neuromorphic Spiking Array Processors: A Quest to Mimic the Brain,” *Front. Neurosci.*, vol. 12, 2018, doi: 10.3389/fnins.2018.00891.

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

[18] R. Gütig and H. Sompolinsky, “The tempotron: a neuron that learns spike timing–based decisions,” *Nature Neuroscience*, vol. 9, no. 3, pp. 420–428, Mar. 2006, doi: 10.1038/nn1643.

[19] Q. Yu, H. Tang, K. C. Tan, and H. Li, “Precise-Spike-Driven Synaptic Plasticity: Learning Hetero-Association of Spatiotemporal Spike Patterns,” *PLOS ONE*, vol. 8, no. 11, p. e78318, Nov. 2013, doi: 10.1371/journal.pone.0078318.

[20] I. Sporea and A. Grüning, “Supervised Learning in Multilayer Spiking Neural Networks,” *Neural Computation*, vol. 25, no. 2, pp. 473–509, Feb. 2013, doi: 10.1162/NECO_a_00396.

[21] S. Woźniak, A. Pantazi, T. Bohnstingl, and E. Eleftheriou, “Deep learning incorporating biologically inspired neural dynamics and in-memory computing,” *Nature Machine Intelligence*, vol. 2, no. 6, Art. no. 6, Jun. 2020, doi: 10.1038/s42256-020-0187-0.

[22] R. J. Williams and D. Zipser, “A Learning Algorithm for Continually Running Fully Recurrent Neural Networks,” *Neural Computation*, vol. 1, no. 2, pp. 270–280, Jun. 1989, doi: 10.1162/neco.1989.1.2.270.

[23] T. P. Lillicrap and A. Santoro, “Backpropagation through time and the brain,” *Current Opinion in Neurobiology*, vol. 55, pp. 82–89, Apr. 2019, doi: 10.1016/j.conb.2019.01.011.

[24] R. A. Fisher, “The Use of Multiple Measurements in Taxonomic Problems,” *Annals of Human Genetics*, vol. 7, no. 2, pp. 179–188, 1936, doi: https://doi.org/10.1111/j.1469-1809.1936.tb02137.x.

[25] Zohar Jackson, César Souza, Jason Flaks, Yuxin Pan, Hereman Nicolas, and Adhish Thite, *Jakobovski/free-spoken-digit-dataset: v1.0.8*. Zenodo, 2018. doi: 10.5281/zenodo.1342401.

[26] S. Nakamura, K. Hiyane, F. Asano, T. Nishiura, and T. Yamada, “Acoustical Sound Database in Real Environments for Sound Scene Understanding and Hands-Free Speech Recognition,” presented at the LREC 2000, Athens, Greece, May 2000. Accessed: Mar. 02, 2021. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2000/pdf/356.pdf

[27] J. Salamon, C. Jacoby, and J. P. Bello, “A Dataset and Taxonomy for Urban Sound Research,” in *Proceedings of the ACM International Conference on Multimedia - MM ’14*, New York, New York, USA, 2014, pp. 1041–1044. doi: 10.1145/2647868.2655045.

[28] X. Valero and F. Alias, “Gammatone Cepstral Coefficients: Biologically Inspired Features for Non-Speech Audio Classification,” *IEEE Transactions on Multimedia*, vol. 14, no. 6, pp. 1684–1689, Dec. 2012, doi: 10.1109/TMM.2012.2199972.

[29] B. Logan, “Mel Frequency Cepstral Coefficients for Music Modeling,” *International Symposium on Music Information Retrieval (ISMIR)*, 2000.

[30] H. Purwins, B. Li, T. Virtanen, J. Schluter, S.-Y. Chang, and T. N. Sainath, “Deep Learning for Audio Signal Processing,” *IEEE Journal of Selected Topics in Signal Processing*, pp. 1–1, 2019, doi: 10.1109/JSTSP.2019.2908700.

[31] S. S. Stevens, J. Volkmann, and E. B. Newman, “A Scale for the Measurement of the Psychological Magnitude Pitch,” *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, Jan. 1937, doi: 10.1121/1.1915893.

[32] Z. Pan, H. Li, J. Wu, and Y. Chua, “An Event-Based Cochlear Filter Temporal Encoding Scheme for Speech Signals,” in *2018 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2018, pp. 1–8. doi: 10.1109/IJCNN.2018.8489434.

[33] S. Dreiseitl and L. Ohno-Machado, “Logistic regression and artificial neural network classification models: a methodology review,” *Journal of Biomedical Informatics*, vol. 35, no. 5, pp. 352–359, Oct. 2002, doi: 10.1016/S1532-0464(03)00034-0.

[34] Ian Goodfellow, Yoshua Bengio and Aaron CourvilleIan J. Goodfellow, *Deep Learning*. MIT Press, 2016. Accessed: Dec. 10, 2020. [Online]. Available: https://www.deeplearningbook.org/

[35] A. Bagheri, O. Simeone, and B. Rajendran, “Training Probabilistic Spiking Neural Networks with First-to-spike Decoding,” *arXiv:1710.10704 [cs, math, stat]*, Feb. 2018, Accessed: Mar. 03, 2021. [Online]. Available: http://arxiv.org/abs/1710.10704

[36] A. Tavanaei and A. Maida, “BP-STDP: Approximating backpropagation using spike timing dependent plasticity,” *Neurocomputing*, vol. 330, pp. 39–47, Feb. 2019, doi: 10.1016/j.neucom.2018.11.014.

[37] G. Bellec *et al.*, “A solution to the learning dilemma for recurrent networks of spiking neurons,” *bioRxiv*, p. 738385, Aug. 2019, doi: 10.1101/738385.

[38] 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.

[39] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *J Physiol*, vol. 117, no. 4, pp. 500–544, Aug. 1952.

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

[41] S. Bohte, J. Kok, and J. Poutré, “SpikeProp: backpropagation for networks of spiking neurons.,” *ESANN*, vol. 48, pp. 419–424, Jan. 2000.

[42] Z. Pan, J. Wu, Y. Chua, M. Zhang, and H. Li, “Neural Population Coding for Effective Temporal Classification,” *arXiv:1909.08018 [cs, q-bio]*, Sep. 2019, Accessed: Jan. 29, 2020. [Online]. Available: http://arxiv.org/abs/1909.08018

[43] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, “Going Deeper in Spiking Neural Networks: VGG and Residual Architectures,” *Front. Neurosci.*, vol. 13, 2019, doi: 10.3389/fnins.2019.00095.

[44] Y. Cao, Y. Chen, and D. Khosla, “Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition,” *Int J Comput Vis*, vol. 113, no. 1, pp. 54–66, May 2015, doi: 10.1007/s11263-014-0788-3.

[45] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. C. Liu, and M. Pfeiffer, “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing,” presented at the The International Joint Conference on Neural Networks (IJCNN) 2015, Piscataway, USA, Jul. 2015. doi: 10.1109/IJCNN.2015.7280696.

[46] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, “Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification,” *Front. Neurosci.*, vol. 11, 2017, doi: 10.3389/fnins.2017.00682.

[47] S. B. Shrestha and G. Orchard, “SLAYER: Spike Layer Error Reassignment in Time,” *arXiv:1810.08646 [cs, stat]*, Sep. 2018, Accessed: Sep. 24, 2020. [Online]. Available: http://arxiv.org/abs/1810.08646

[48] Y. Bengio, N. Léonard, and A. Courville, “Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation,” *arXiv:1308.3432 [cs]*, Aug. 2013, Accessed: Oct. 13, 2020. [Online]. Available: http://arxiv.org/abs/1308.3432

[49] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” *arXiv:1603.04467 [cs]*, Mar. 2016, Accessed: Feb. 18, 2021. [Online]. Available: http://arxiv.org/abs/1603.04467

[50] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[51] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, “Building a large annotated corpus of English: the penn treebank,” *Comput. Linguist.*, vol. 19, no. 2, pp. 313–330, Jun. 1993.

[52] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, “Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription,” *arXiv:1206.6392 [cs, stat]*, Jun. 2012, Accessed: Feb. 10, 2021. [Online]. Available: http://arxiv.org/abs/1206.6392

[53] O. Booij and H. tat Nguyen, “A gradient descent rule for spiking neurons emitting multiple spikes,” *Information Processing Letters*, vol. 95, no. 6, pp. 552–558, Sep. 2005, doi: 10.1016/j.ipl.2005.05.023.

[54] S. Ghosh-Dastidar and H. Adeli, “A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection,” *Neural Networks*, vol. 22, no. 10, pp. 1419–1431, Dec. 2009, doi: 10.1016/j.neunet.2009.04.003.

[55] Y. Xu, X. Zeng, L. Han, and J. Yang, “A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks,” *Neural Networks*, vol. 43, pp. 99–113, Jul. 2013, doi: 10.1016/j.neunet.2013.02.003.

[56] J. H. Lee, T. Delbruck, and M. Pfeiffer, “Training Deep Spiking Neural Networks Using Backpropagation,” *Front. Neurosci.*, vol. 10, 2016, doi: 10.3389/fnins.2016.00508.

[57] H. Markram, W. Gerstner, and P. J. Sjöström, “Spike-Timing-Dependent Plasticity: A Comprehensive Overview,” *Front. Synaptic Neurosci.*, vol. 4, 2012, doi: 10.3389/fnsyn.2012.00002.

[58] J. Sjöström and W. Gerstner, “Spike-timing dependent plasticity,” *Scholarpedia*, vol. 5, no. 2, p. 1362, Feb. 2010, doi: 10.4249/scholarpedia.1362.

[59] D. O. Hebb, *The organization of behavior; a neuropsychological theory*. Oxford, England: Wiley, 1949, pp. xix, 335.

[60] C. J. Schatz, “The Developing Brain,” *Scientific American*, vol. 267, no. 3, pp. 60–67, 1992.

[61] F. Ponulak and A. Kasiński, “Supervised Learning in Spiking Neural Networks with ReSuMe: Sequence Learning, Classification, and Spike Shifting,” *Neural Computation*, vol. 22, no. 2, pp. 467–510, Oct. 2009, doi: 10.1162/neco.2009.11-08-901.

[62] P. U. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Front. Comput. Neurosci.*, vol. 9, 2015, doi: 10.3389/fncom.2015.00099.

[63] N. Frémaux and W. Gerstner, “Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules,” *Front. Neural Circuits*, vol. 9, 2016, doi: 10.3389/fncir.2015.00085.

[64] W. Gerstner, M. Lehmann, V. Liakoni, D. Corneil, and J. Brea, “Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of neoHebbian Three-Factor Learning Rules,” *Front. Neural Circuits*, vol. 12, p. 53, Jul. 2018, doi: 10.3389/fncir.2018.00053.

[65] R. V. Florian, “Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity,” *Neural Comput.*, vol. 19, no. 6, pp. 1468–1502, Jun. 2007, doi: 10.1162/neco.2007.19.6.1468.

[66] E. M. Izhikevich, “Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling,” *Cereb Cortex*, vol. 17, no. 10, pp. 2443–2452, Oct. 2007, doi: 10.1093/cercor/bhl152.

[67] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe, and T. Masquelier, “Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks,” *Pattern Recognition*, vol. 94, pp. 87–95, Oct. 2019, doi: 10.1016/j.patcog.2019.05.015.

[68] A. Amirshahi and M. Hashemi, “ECG Classification Algorithm Based on STDP and R-STDP Neural Networks for Real-Time Monitoring on Ultra Low-Power Personal Wearable Devices,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1483–1493, Dec. 2019, doi: 10.1109/TBCAS.2019.2948920.

[69] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[70] S. Thorpe, “Spike arrival times: A highly efficient coding scheme for neural networks,” *Parallel Processing in Neural Systems and Computers*, Jan. 1990.

[71] D. A. Butts *et al.*, “Temporal precision in the neural code and the timescales of natural vision,” *Nature*, vol. 449, no. 7158, Art. no. 7158, Sep. 2007, doi: 10.1038/nature06105.

[72] Z. Nadasdy, “Information Encoding and Reconstruction from the Phase of Action Potentials,” *Front Syst Neurosci*, vol. 3, Jul. 2009, doi: 10.3389/neuro.06.006.2009.

[73] R. Brette, “Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain,” *Front. Syst. Neurosci.*, vol. 9, 2015, doi: 10.3389/fnsys.2015.00151.

[74] D. P. Kingma and J. L. Ba, “Adam optimizer,” *arXiv preprint arXiv:1412.6980*, 2014, doi: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503.

[75] J. C. Brown, “Calculation of a constant Q spectral transform,” *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 425–434, Jan. 1991, doi: 10.1121/1.400476.

[76] D. Heeger, “Poisson Model of Spike Generation,” Handout, University of Standford, vol.5, 2000.

[77] T. Kite, “Understanding PDM Digital Audio,” 2012. Accessed: Feb. 17, 2021. [Online]. Available: http://users.ece.utexas.edu/~bevans/courses/rtdsp/lectures/10_Data_Conversion/AP_Understanding_PDM_Digital_Audio.pdf

[78] Da-Shan Shiu and J. M. Kahn, “Differential pulse-position modulation for power-efficient optical communication,” *IEEE Transactions on Communications*, vol. 47, no. 8, pp. 1201–1210, Aug. 1999, doi: 10.1109/26.780456.

[79] R. Gütig and H. Sompolinsky, “Time-Warp–Invariant Neuronal Processing,” *PLOS Biology*, vol. 7, no. 7, p. e1000141, Jul. 2009, doi: 10.1371/journal.pbio.1000141.

[80] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1,” *arXiv:1602.02830 [cs]*, Mar. 2016, Accessed: Mar. 26, 2021. [Online]. Available: http://arxiv.org/abs/1602.02830

[81] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, no. 1, p. 3, 2013.

[82] A. Mesaros, T. Heittola, and T. Virtanen, “Metrics for Polyphonic Sound Event Detection,” *Applied Sciences*, 2016, doi: 10.3390/app6060162.

[83] A. ELDEM, H. ELDEM, and D. ÜSTÜN, “A Model of Deep Neural Network for Iris Classification With Different Activation Functions,” in *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, Sep. 2018, pp. 1–4. doi: 10.1109/IDAP.2018.8620866.

[84] C. L. Devasena, T. Sumathi, V. V. Gomathi, and M. Hemalatha, “Effectiveness Evaluation of Rule Based Classifiers for the Classification of Iris Data Set,” *Bonfring International Journal of Man Machine Interface*, vol. 1, no. Special Issue Inaugural Special Issue, pp. 05–09, Dec. 2011, doi: 10.9756/BIJMMI.1002.

[85] J. Wu, M. Zhang, H. Li, and Y. Chua, “Competitive STDP-based Feature Representation Learning for Sound Event Classification,” in *2019 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2019, pp. 1–8. doi: 10.1109/IJCNN.2019.8851688.

[86] J. Dennis, Q. Yu, H. Tang, H. D. Tran, and H. Li, “Temporal coding of local spectrogram features for robust sound recognition,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 803–807. doi: 10.1109/ICASSP.2013.6637759.

[87] R. Xiao, R. Yan, H. Tang, and K. C. Tan, “A Spiking Neural Network Model for Sound Recognition,” in *Cognitive Systems and Signal Processing*, Singapore, 2017, pp. 584–594. doi: 10.1007/978-981-10-5230-9_57.

[88] J. Wu, Y. Chua, M. Zhang, H. Li, and K. C. Tan, “A Spiking Neural Network Framework for Robust Sound Classification.,” *Frontiers in neuroscience*, vol. 12, p. 836, 2018, doi: 10.3389/fnins.2018.00836.

[89] M. Abdollahi and S.-C. Liu, “Speaker-independent isolated digit recognition using an AER silicon cochlea,” in *IEEE Biomedical Circuits and Systems Conference*, San Diego, CA, Nov. 2011, pp. 269–272. doi: 10.1109/BioCAS.2011.6107779.