Spring 2018

Spike-Based Classification of UCI Datasets with Multi-Layer Resume-Like Tempotron

Sami Abdul-Wahid
Central Washington University, sipho.1989@gmail.com

SPIKE-BASED CLASSIFICATION OF UCI DATASETS WITH MULTI-LAYER RESUME-LIKE TEMPOTRON

A Thesis

Presented to

The Graduate Faculty

Central Washington University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Computational Science

---

by

Sami Abdul-Wahid

May 2018

CENTRAL WASHINGTON UNIVERSITY

Graduate Studies

We hereby approve the thesis of

Sami Abdul-Wahid

Candidate for the degree of Master of Science

APPROVED FOR THE GRADUATE FACULTY

_____ _____
Dr. Răzvan Andonie, Committee Chair

_____ _____
Dr. Szilárd Vajda

_____ _____
Dr. Donald Davendra

_____ _____
Dean of Graduate Studies

ABSTRACT

SPIKE-BASED CLASSIFICATION OF UCI DATASETS WITH MULTI-LAYER RESUME-LIKE TEMPOTRON

by

Sami Abdul-Wahid

May 2018

Spiking neurons are a class of neuron models that represent information in timed sequences called "spikes." Though predominantly used in neuro-scientific investigations, spiking neural networks (SNN) can be applied to machine learning problems such as classification and regression. SNN are computationally more powerful per neuron than traditional neural networks. Though training time is slow on general purpose computers, spike-based hardware implementations are faster and have shown capability for ultra-low power consumption. Additionally, various SNN training algorithms have achieved comparable performance with the State of the Art on the Fisher Iris dataset. Our main contribution is a software implementation of the multilayer ReSuMe algorithm using the Tempotron principle. The XOR problem is solved in only 13.73 epochs on average. However, training time on four different UCI datasets is slow, and, although decent performance is seen, in most respects the accuracy of our SNN underperforms compared to other SNN, SVM, and ANN experiments. Additionally, our results on the UCI dataset are only preliminary, necessitating further tuning.

ACKNOWLEDGEMENTS

I would like to thank Dr. Razvan Andonie for his willingness to mentor me on a project outside his expertise and who demonstrated sincere interest and insight in the overarching direction of the project. Dr. Szilard Vajda has given wonderful feedback on this and other work, and helped give me ideas on how to find bugs in the weight update routine. Dr. Davendra has lent his share of feedback while polishing my thesis. My mother and father helped motivate me to finish while my boss, David Firth, has gone out of his way to give me the space to finish my education. I also would like to thank all my other friends, Joe and Snail Lemley, Dr. Michael Lundin, Namubirhu Ritah, Michilo Marcessen, and many others for being wonderful people.

I dedicate this thesis to Noura Hussein of the Sudan, who was robbed of her life and education after being forced into a marriage at a young age, and subsequently sentenced to death for defending herself within an abusive power structure. I also dedicate this thesis to Abdul-Baha, Who spent a life teaching the need for unconditional love and the equality of women and men.

TABLE OF CONTENTS

Chapter Page

I INTRODUCTION 1

Research Focus 2

II BACKGROUND 4

Classification 4
Spiking Neural Networks 8
Supervised Training with SNN 14
Class Encoding 33
UCI Datasets 34

III METHODS 37

Network Parameters 37
ReSuMe Based Multi-Layer Tempotron Training Algorithm 39
Brian2 Simulation Library 43
The XOR Problem 43
UCI Dataset Experiments 44

IV RESULTS 48

The XOR Problem 48
UCI Dataset Experiments 50
Comparison With Other SNNs on Iris 55

V CONCLUSION 57

Future Work 58

REFERENCES CITED 60

LIST OF FIGURES

Figure		Page
1	Feedforward architecture	6
2	The sigmoid function	6
3	The sigmoid neuron	6
4	A biological neuron diagram	9
5	Two spiking neurons	10
6	Image classification with SNN	10
7	Example dynamics in a spiking neuron	13
8	ReSuMe weight progression	23
9	DL-ReSuMe weight progression	23
10	A comparison between DL-ReSuMe and ReSuMe	25
11	Tempotron speed vs. ReSuMe speed	27
12	STIP vs. ReSuMe on Iris	29
13	Multilayer SNN with delay training on Iris	31
14	Effect of training set size on Iris classification	33
15	Connectivity structure of our SNN	38
16	All UCI datasets.	51
17	Boxplot: Epochs until recorded.	52

LIST OF TABLES

Table Page

1 SNN classification on Iris 32

2 Convergence on XOR 32

3 UCI dataset characteristics 34

4 Classification on UCI datasets 36

5 XOR problem parameters searched 43

6 XOR problem encoding 44

7 XOR convergence statistics 48

8 XOR comparison 49

9 Neuron parameters on XOR 50

10 SNN accuracy on UCI datasets 53

11 Test accuracy on UCI datasets 55

12 SNN classification on Iris 55

CHAPTER I

INTRODUCTION

Spiking neural networks (SNN) are a class of neuron models developed to simulate dynamical behaviors of biological neurons. SNN exhibit time-continuous dynamics of one or more state variables, often inspired from neuroscience, enabling them to capture phenomena not seen in traditional artificial neural networks (ANN).

Spiking neurons often communicate by sending brief, high amplitude pulses of activity, colloquially referred to as "spikes," which are key characteristics of biological neural phenomena. A spiking neuron's internal dynamics, as influenced by incoming spikes, determines if and when it sends its own spikes. Dynamical SNN models, which vary with levels of sophistication, relate these neural input and output dynamics. The more complex models are highly nonlinear, while others are linear and relatively simple.

Early paradigms of thought strictly assumed that biological neurons communicated information predominantly through mean spike firing rates. Recent evidence, however, has shown that precise spike-time encoding is widely used by the brain [1, 2], allowing for higher information transfer rate [3] and greater metabolic efficiency. SNN models are therefore commonly studied in the context of their local spike times [4, 5, 6].

SNN applications range from neuroscientific investigations [7] to classic machine learning problems. They have been trained to classify images [8, 9, 10] and recognize sound events [11], and can also perform unsupervised clustering [12]. Additionally recurrent SNN have shown good performance on time-series prediction problems [13]. While gradient-based supervised training of SNN presents mathematical challenges, important developments have been made by several researchers [14, 15, 16, 17, 18].

Logic units implemented with spike-time encoding are computationally more powerful than their digital counterparts [19]. However, as simulating SNN on digital processors often incurs high computational overhead, quick and efficient hardware implementations have been studied [20, 21, 22]. Trained deep ANN have also successfully been adapted to efficient spike-based encoding in hardware [23, 24] and hardware implementations of SNN may be trained to reduce power usage [25].

As such, SNN can be useful and viable alternatives to classic machine learning approaches. However, algorithmic studies in SNN training can be carried out with more flexibility in software, after which research may be conducted to investigate the feasibility of a useful hardware implementation.

Research Focus

Here the Tempotron-like ReSuMe rule, as seen in [26], is adapted to a two-layer network according to [16]. This forms the multi-layer ReSuMe-like Tempotron algorithm, which is similar to the multi-layer Tempotron algorithm from [17], but is based on the exponential windowing functions of the ReSuMe algorithm. We implement the multi-layer ReSuMe-like Tempotron on a general purpose desktop computer. Using this, classification is performed on the XOR problem and four UCI datasets. Results on the XOR problem show convergence in 13.73 ± 6.70 epochs, a significant improvement over the results in [17], yet does not surpass more recent work [27]. On UCI, our SNN showed decent performance on the Iris [28] and Ionosphere [29] benchmarks, yet performed poorly on both the Vertebral Column [30, 31, 32] and, especially, the Teaching Assistant Evaluation (TAE) [33] datasets. In all cases, SNN results underperformed in-house SVM experiments with respect to training time and accuracy, while performance on TAE was only slightly better than a random classifier. Our results on the UCI datasets

are preliminary, as the six weeks of computation time prevented a thorough parameter investigation. Meanwhile our classification results on the XOR problem demonstrates the training speed advantages of single-spike classification in SNN. Further study is needed on this training algorithm for a definitive conclusion.

CHAPTER II

BACKGROUND

Classification

Classification is a broad field with applications such as image recognition [34, 35, 36], audio recognition [37], and cancerous cell diagnosis [38, 39]. Humans visually perform classification by recognizing characteristic combinations of features. To classify, machines must correlate distinguishing features with the associated class. An example includes the subspecies of Iris flowers correlating with the petal and sepal dimensions. Machines can learn such correlations when given a database of iris flowers, such as the well known Fisher Iris dataset [28], that contains feature quantities paired with species label. The light representing scenes or pictures of objects will also correlate with the type of scene or object, allowing a competent machine to learn and generalize from these.

To identify correlating features, a wide variety of supervised learning algorithms are available. These must access the features and class labels from a database. Generating these databases often requires collective effort from institutions. An organization may hire workers to look through and classify thousands of human photographs by gender and age [40]. This allows scientists and engineers to study and apply algorithms for gender classification.

Artificial Neural Networks

ANN are a class of trainable functions. These can, given arbitrary size and parameters, model any mathematical function [41, 42]. ANN can represent continuous [43], discrete [44], and boolean [45] functions.

ANN consist of neural units (neurons) connected together in a directed graph. A feedforward connectivity architecture is commonly used, typically with three layers as seen in Fig. 1. Each connection is associated with one or more parameters, such as a weight value. Classical ANN are widely used in engineering applications due to their efficiency and effectiveness, and their neuron units consist of functions of weighted inputs.

The sigmoid function is a widely used neuron activation function. This is defined as

$$\text{sgm}(s) = \frac{1}{1 + \exp\left(-\beta s\right)} \tag{2.1}$$

and illustrated in Fig. 2. This function represents a neuron's output firing rate as a function of the scalar argument (s). Here $s = \sum_i w_i x_i$ where x_i represents firing rate from input i and w_i is the corresponding weight (see Fig. 3).

To derive the weight changes used to train the network, a gradient descent approach [46] can be taken in which the network output is differentiated by the weights. In a feedforward three-layer structure, the outputs of output neuron k and hidden neuron j are

$$z_k = \text{sgm}(\vec{y} \cdot \vec{W}_k) \tag{2.2}$$

$$y_j = \text{sgm}(\vec{x} \cdot \vec{V}_j). \tag{2.3}$$

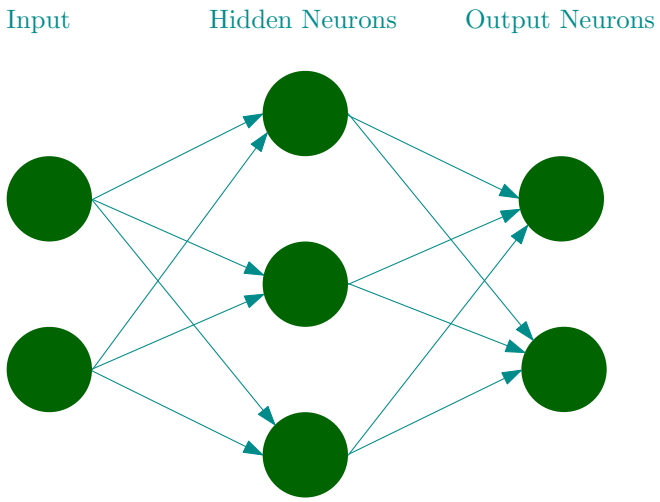


FIGURE 1.: Feedforward architecture of neurons.

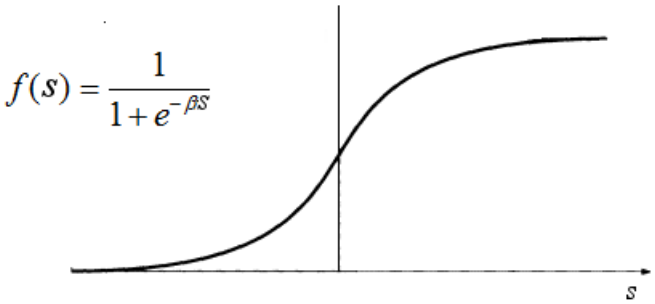


FIGURE 2.: The sigmoid function often used in traditional ANN.

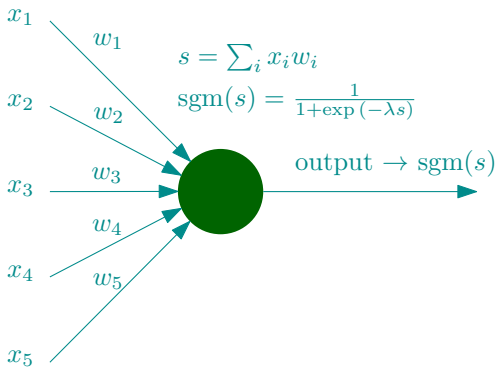


FIGURE 3.: The sigmoid neuron
The sigmoid neuron.

where $\vec{x}$ and $\vec{y}$ are inputs to the network and the outputs of hidden neurons, respectively; $\mathbf{W}$ and $\mathbf{V}$ are weight matrices to hidden neurons and output neurons respectively, and $\vec{V}_j$ and $\vec{W}_k$ are weight vectors to hidden neuron j and output neuron k, respectively.

Supervised Training in ANN

Since ANN use continuous functions, differentiation with respect to parameters is straightforward as it only involves application of the chain rule in calculus [46]. Supervised training involves modifying weights along a gradient of the error function. When given an input vector and a desired output vector is known, the relative weight modifications in a training step can be determined to bring the observed output closer to its desired value. To determine weight modifications for each connection, the partial derivatives δ_{jk} and δ_{ij} are expressed as

$$\delta_{jk} = \frac{\partial z_k}{\partial w_{jk}} \tag{2.4}$$

$$\delta_{ij} = \frac{\partial z_k}{\partial v_{ij}} \tag{2.5}$$

where w_{jk} and v_{ij} are the weight of the connection between hidden neuron j and output neuron k, and the weight between input neuron i and hidden neuron j, respectively. The gradient magnitude is expressed in terms of

$$\sigma^2 = \sum_{i,\,j} \delta_{ij}^2 + \sum_{j,\,k} \delta_{jk}^2 \tag{2.6}$$

after which the weight changes at each step can be expressed as

$$\Delta w_{jk} = r\delta_{jk}(z_k - d_k)/\sigma \tag{2.7}$$

$$\Delta v_{ij} = r\delta_{ij}(z_k - d_k)/\sigma \tag{2.8}$$

where r is the learning rate, and d_k is the desired output value of output neuron k.

Supervised training of a neural network requires a dataset of input values in which each data point has a corresponding desired output value, or label. The training process involves computing the ANN-predicted output for each data point, and modifying the weights according to the above gradient descent rule to bring the network closer to the desired output at each step. This continues until a specified stopping condition, such as meeting a desired training accuracy, is met.

Spiking Neural Networks

SNN are a different class of neural networks, as compared to ANN. Whereas ANN are modeled after a rate-based communication of biological neurons, SNN exhibit qualities of finer grained neural behavior. This requires a degree of temporal dynamism.

Spiking neuron models are often described by differential equations. Some variants exhibit complex non-linear phenomena, while other more simplistic models can be a linear sum of time varying inputs. A key commonality is the time dimension which allows dynamic evolution of one or more state variables, and is necessary for a precise spike time encoding.

SNN are often described in terms of biological neural behavior. A biological neuron, shown in Fig. 4, consists of dendrites and axons, along with a cell body. Neurons exhibit an electric potential difference, called the membrane potential, between the

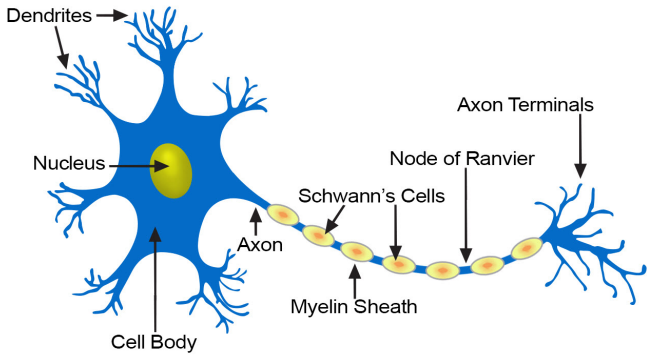


FIGURE 4.: A biological neuron diagram [47].

inside and outside of the cell membrane. In a real neuron, this membrane potential is highly dynamic and can manifest a wide-ranging array of rich behaviors. Under certain circumstances the neuron will produce an action potential, a process in which a brief spike in electric activity is sent down the axon. The axon is usually connected to dendrites of other neurons, which then receive the signal. After firing, there is usually a brief time period, called the refractory period, in which the neuron is relatively quiescent. Various SNN neuron models seek to mimic this dynamic neural behavior with different degrees of sophistication.

Fig. 5 shows two neuron models connected by a single synapse with weight w. The pre-synaptic neuron fires a series of precisely timed spikes along the synapse. Each spike is convolved with the synaptic kernel to produce the synaptic current, which flows into the post-synaptic neuron, influencing its membrane potential and inducing the post-synaptic neuron to then fire its own series of spikes.

One or more spiking neurons can be arranged for decision making. For instance, a spike-based classifier can be constructed by encoding attributes from an image dataset into spike times and feeding these into an SNN that will be trained. The classifier's output neurons can then be trained to fire in certain manners depending on the input image class. See Fig. 6 for an example.

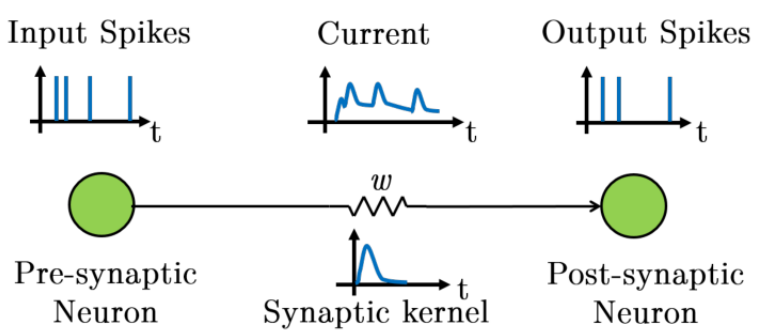


FIGURE 5.: Two spiking neurons firing a precise times [18]. Here the pre-synaptic neuron emits spikes that travel down the synapse. Each spike is in the form of the synaptic kernel, the amplitude of which is multiplied by the synaptic weight w upon arrival at the pos-synaptic neuron. These spikes then activate the membrane potential of the pos-synaptic neuron which then emits its own spikes.

Spiking Neuron Models

A classic and biologically realistic spiking neuron model is the Hodgkin Huxley (HH) neuron [48]. It is named after its discoverers, Alan L. Hodgkin and Andrew F. Huxley, who won the Nobel Prize in Physiology or Medicine for their seminal work, which included the development of the groundbreaking HH model. This model describes the electro-chemical dynamics observed in a squid neuron through a set of coupled differential equations. Simulations of the HH neuron can exhibit a set of key qualitative

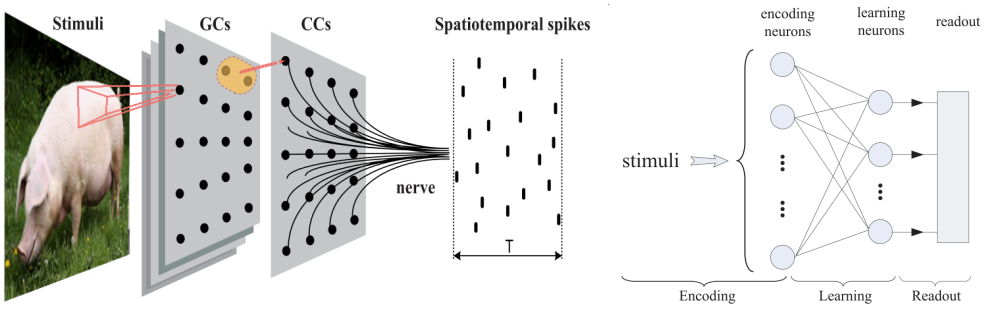


FIGURE 6.: Image classification with SNN. Encoders and architecture are taken from [10]. The image is convolved with spatial filters (GCs and CCs) into a series of spatiotemporal spikes which are fed into the encoding neurons of a feed-forward SNN classifier.

dynamics observed in biological neurons [49]. It is thus often used by neuroscientists as a template equation to model specific neural phenomena in detail.

The high computational cost in simulating HH neurons [49] has eventually led to the discovery of the Izhikevich neuron model [50]. Named after its discoverer Eugen M. Izhikevich, the Izhikevich model (or Izhikevich neuron) is more condensed and computationally tractable than the HH neuron, and manifests many of the rich behaviors seen in the HH model. However, it does not directly describe the electro-chemical dynamics of neural ionic channels.

The Leaky Integrate and Fire (LIF) neuron [6], another spiking neuron model, is much simpler and lacks many of the rich dynamics seen in the HH and Izhikevich neurons. However, it requires very little computational overhead during simulation and still allows for precise spike time encoding.

One variant of LIF neurons is governed by the following differential equation [18]:

$$C_m V' = -g_L(V - E_L) + I, \tag{2.9}$$

where V is the primary variable representing the voltage across the neuron membrane, I is the input current from other neurons, g_L and C_m are constants that influence the membrane potential's rate of evolution, and E_L is the resting potential. V' denotes the time derivative of V. Assuming an analytical form for I, the LIF neuron has an analytical solution that can manifest jump discontinuities as time evolves [18].

A simpler case of the LIF model takes no differential equation form [15], but is simply the weighted sum of synaptic input currents. This is expressed as

$$V = I \tag{2.10}$$

In either case, I represents the pre-synaptic current. When dealing with networks manifesting precise spike time coding, I often is a weighted sum of double exponential, α, where

$$I = \sum_i w_i \sum_f \alpha(t - t_{if}) \tag{2.11}$$

and

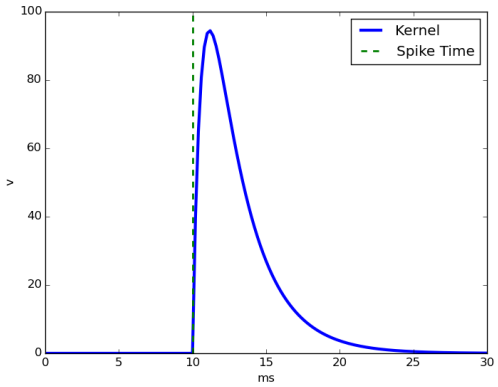
$$\alpha(t) = e^{-t/\tau_1} - e^{-t/\tau_2} \tag{2.12}$$

Here, t_{if} is the firing time of the f^{th} spike emitted by the i^{th} pre-synaptic neuron. Both τ_1 and τ_2 are time constants, and $\tau_1 > \tau_2$. The function $\alpha(t)$ is known as the *synaptic kernel*. If the voltage V reaches the value $V = V_T$, then the neuron outputs a spike at that time. This sends the output synaptic kernel $\alpha(t - t_f)$ to any neurons in the next layer. Fig. 7 (a) shows the shape of the synaptic kernel in a voltage vs. time graph. Additionally, multiple spikes can be facilitated by setting $V = E_L$ at the times that the threshold is reached, and perhaps including a brief refractory period in which all input currents are blocked.

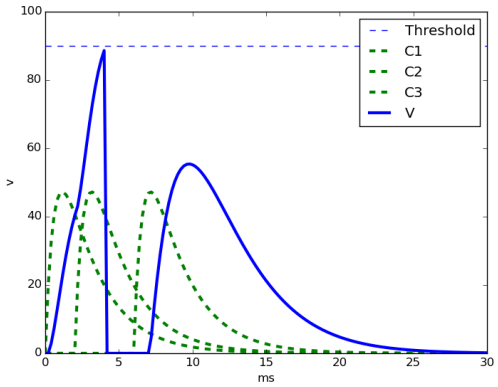
When V reaches the threshold and resets, the neuron will emit a spike. The exact value at which this happens depends on the dynamics of the state variable, which depends on the times and amplitudes of the signals preceding it. The goal of supervised learning in a feedforward spiking neural network, then, is to appropriately modify any of the preceding weights, delays, and other neuron model parameters in order to produce a desired relationship between the input and output spike times. The ultimate goal is to train the network to accurately generalize on data it has not previously seen.

Software Tools

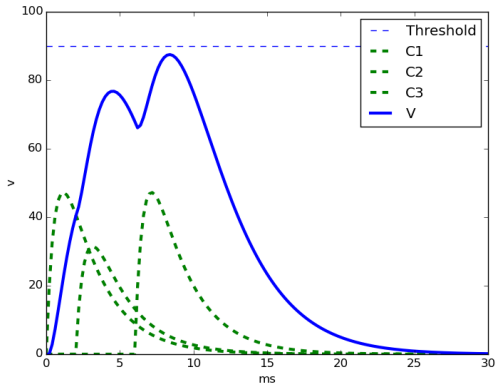
Software libraries for simulating SNN do exist, such as the Brian2 library [51] for Python. Brian2 consists of a high-level interface to a CPU-based numerically efficient



(a) Kernel



(b) Spiking



(c) Non-Spiking

FIGURE 7.: Example dynamics in a spiking neuron modeled by Eq. 2.9. The synaptic kernel Eq. 2.12 is used. (a) shows the kernel itself in blue, with the spike time indicated by a green dashed line. (b) and (c) each show the neuron voltage activity in blue, with three input synaptic kernels multiplied by their respective weights, and the threshold of spiking in dotted blue: (b) shows a case in which the input kernels induce the neuron potential to reach the threshold and spike at around 4.5 ms, followed by a refractory period of 3 ms; (c) shows a case in which the input spikes do not cause spiking.

backend, allowing the user to incorporate one or more user-specified spiking neuron models, connectivity structures, and synaptic equation models. Both numerical time integration, and analytic spike time computations, are supported in the backend, which uses tools such as C++ and NumPy. However, it does not provide weight or delay training algorithms for SNN.

Hardware Implementations

Von Neumann-based architectures are inherently energy-inefficient, and more complex spiking neuron models require expensive time integration techniques. As such, a number of studies have sought to implement SNN efficiently in hardware. For instance, IBM developed TrueNorth, a clockless, spike-based digital microarchitecture chip capable of simulating one million neurons with only 70 mW of power, far less than typically seen in general-purpose computing processors [52, 22]. Likewise, as spike-based classification often varies in the needed number of spikes for decision making, training SNN for power efficiency has been theoretically shown to be a viable way of reducing power consumption [25]. In addition, the inherent asynchrony of analog-based hardware also makes for an elegant and simple 'brain-like' medium for neural computation. This has inspired recent research in overcoming the challenges [53] analog hardware poses to learning, demonstrating the ability for high precision on imprecise architectures [21, 54].

Supervised Training with SNN

Several approaches have been taken to train SNN. Early work with evolutionary algorithms [55] and statistics-based algorithms involving stochastic neuron models [56, 57] have been conducted. However, the evolutionary approach to training converges

slowly, with 450 generations needed to learn the XOR problem. In addition, the statistical approach in [56] [58].

Gradient Descent with SNN

As the efficiency of gradient descent-based training algorithms have repeatedly been seen in ANN, attempts have been made to mimic this in SNN. ANN can be trained using a direct calculation of a gradient because of their continuity. Computing gradient descent with SNN, however, presents challenges. Although some forms of Eq. 2.9 are analytically tractable, yet mathematical discontinuities occur at neural firings. This is significant in the case of a multi-layer architecture, where a small change in weight can cause the sudden appearance or disappearance of an incoming spike. Other neuron models such as HH and Izhikevich neurons have no known general analytic solution.

The importance of training multi-layer SNN lies in their capacity for learning patterns, and in their ability to solve linearly non-separable problems [14, 16]. The memory capacity of a single layer SNN is limited by the number of input synapses, thus putting constraints on the spike encoding, whereas in a multi-layer network, capacity can be effected by the number of hidden units without effecting encoding. This has motivated research on how to effectively train single-layer and multi-layer SNN.

Spikeprop

Spikeprop is the first supervised training algorithm devised for SNN. It minimizes the energy function through differentiation of the output spike time with respect to the synaptic weights. Spikeprop solves the problem of discontinuities by requiring each neuron to fire exactly once [14, 59, 60]. However, because it is a gradient based approach, it is prone to getting stuck in local minima [58]. In addition, in order to approximately

differentiate the state variable, each pre-synaptic spike must have a linear effect on $V_i(t)$, i.e. the sum of their effects should equal their lone effects summed. As such, Spikeprop can best be used with linear neuron models such as LIF. When training on the Iris dataset, after 1000 epochs 97.4% accuracy on the training set, and 96.1% accuracy on the test set, were achieved.

The original Spikeprop algorithm was derived only for training synaptic weights. Later it was improved [60] by incorporating the training of other network parameters, such as synaptic delays. This reduces the number of synaptic connections needed to solve the XOR problem by an order of magnitude, and results in much faster convergence.

Widrow-Hoff Rule in Gradient Descent

Other approaches to training SNN attempt to use an adapted version of the gradient descent approach. This is done by basing weight changes on a firing rate approximation of the neuron, i.e. the Widrow-Hoff (WH) rule. The WH rule is

$$\Delta w = rx(y_d - y), \tag{2.13}$$

where w is the weight in question, x is the input firing rate to the neuron, y is the neuron output firing rate, and y_d is the desired output firing rate of the neuron. However, this is problematic, since training involves the minimization of an energy function often determined by differences between spike trains:

$$E(t) = (S_o^a(t) - S_o^d(t))^2 \tag{2.14}$$

which is inherently discontinuous. Here

$$S_o^a(t) = \sum_f \delta(t - t_f^a) \tag{2.15}$$

t_f^a is the f^{th} observed firing time of output neuron o and

$$\delta(t) = \begin{cases} \infty & \text{if } t = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

is the Kronecker delta function that satisfies

$$\int_{-\infty}^{\infty} \delta(t)dt = 1. \tag{2.17}$$

The set of desired firing times meanwhile are denoted as

$$S_o^d(t) = \sum_f \delta(t - t_f^d) \tag{2.18}$$

where t_f^d is the f^{th} desired firing time of output neuron o.

The task of the training algorithm is to connect the continuous WH approximation to the disconinuous spike-train behavior of the neurons. Several training algorithms such as ReSuMe [16], PSD [61], and SPAN [62], achieve this.

ReSuMe

The ReSuMe training algorithm [16] is a very flexible and biologically plausible algorithm that can, in principle, be used to train weights and delays in networks of arbitrary number of hidden layers. It incorporates the principle of spike-timing dependent

plasticity (STDP), which has been seen in actual neural behavior (in different variations). STDP is a principle by which weights of synapses passing correlating spikes will strengthen over time, while those passing non-correlating spikes will correspondingly weaken.[1] The derivation of ReSuMe begins by initially expressing the energy function in terms of a difference in firing rates of the output neurons as

$$E(t) = \frac{1}{2} \sum_{o \in O} (R_o^a(t) - R_o^d(t))^2, \tag{2.19}$$

where $R_o^a(t)$ is the actual firing rate of output neuron o, and $R_o^d(t)$ is the desired firing rate of output neuron o. Then, as in the WH rule, a linear relationship between the firing rates of connected neurons in adjacent layers is assumed:

$$R_o(t) = \frac{1}{n} \sum_{h \in H} w_{ho} R_h(t) \tag{2.20}$$

$$R_h(t) = \frac{1}{m} \sum_{i \in I} w_{ih} R_i(t). \tag{2.21}$$

Here $R_o(t)$ is the time instantaneous firing rate of output neuron o, and n is the number of hidden neurons; $R_h(t)$ is the time instantaneous firing rate of hidden neuron h while w_{oh} is the weight of the connection between hidden neuron h and output neuron o, and m is the number of input neurons.

[1] STDP alone, when properly configured for, can be used to train an SNN in an unsupervised manner.

This energy function, expressed in continuous terms, can be differentiated with respect to any of the weight parameters in the network:

$$\frac{\partial E(t)}{\partial w_{ho}} = \frac{1}{n}(R_o^a(t) - R_o^d(t))R_h(t) \tag{2.22}$$

$$\frac{\partial E(t)}{\partial w_{ih}} = \sum_{o \in O} \frac{w_{ho}}{n}(R_o^a(t) - R_o^d(t))R_i(t) \tag{2.23}$$

After this, the rate terms are switched into Kronecker delta terms which describe spike trains

$$R(t) \rightarrow S(t) \tag{2.24}$$

where

$$S(t) = \sum_f \delta(t - t_f) \tag{2.25}$$

to yield the expressions

$$\frac{\partial E(t)}{\partial w_{ho}} = \frac{1}{n}(S_o^a(t) - S_o^d(t))S_h(t) \tag{2.26}$$

$$\frac{\partial E(t)}{\partial w_{ih}} = \sum_{o \in O} \frac{w_{ho}}{n}(S_o^a(t) - S_o^d(t))S_i(t) \tag{2.27}$$

Although products of Kronecker delta terms are mathematically problematic, a statistical model [63] has been developed that describes long-term plasticity between a pre-synaptic and post-synaptic neuron, i.e. weight changes, associated with pre-synaptic

and post-synaptic firing correlation. According to this model,

$$S_{post}(t)S_{pre}(t) \rightarrow S_{pre}(t)\left[a + \int_0^\infty a^{pre}(s)S_{post}(t-s)ds\right] + \\ + S_{post}(t)\left[a + \int_0^\infty a^{post}(s)S_{pre}(t-s)ds\right]. \tag{2.28}$$

It has been shown that this plasticity model will weaken the weights that transfer highly variable spike trains, such as those subjected to high amounts of random noise fluctuations, while synapses transferring consistent spike trains will strengthen over time. By substituting Eq. 2.28 into Eqs. 2.26 and 2.27, the resulting weight update expressions are:

$$\frac{\partial w_{ho}}{\partial t} = \frac{1}{n}S_h(t)\left[\int_0^\infty a^{pre}(s)[S_o^d(t-s) - S_o^a(t-s)]ds\right] \\ + \frac{1}{n}[S_o^d(t) - S_o^a(t)]\left[a + \int_0^\infty a^{post}(s)S_h(t-s)ds\right] \tag{2.29}$$

$$\frac{\partial w_{ih}}{\partial t} = \frac{1}{mn}S_i(t)\sum_{o\in O}\left[\int_o^\infty a^{pre}(s)[S_o^d(t-s) - S_o^a(t-s)]ds\right] w_{ho} \\ + \frac{1}{mn}\sum_{o\in O}[S_o^d(t) - S_o^a(t)]\left[a + \int_0^\infty a^{post}(s)S_i(t-s)ds\right] w_{ho} \tag{2.30}$$

The terms a^{pre} and a^{post} are called the window functions, and are often exponential in nature:

$$a^{pre}(-s) = -A_- \exp\left(\frac{s}{\tau_-}\right), \qquad \text{if } s \leq 0 \tag{2.31}$$

$$a^{post}(s) = +A_+ \exp\left(\frac{-s}{\tau_+}\right), \qquad \text{if } s > 0 \tag{2.32}$$

Here a is a non-Hebbian term, and is an important component of spike timing dependent plasticity. The non-Hebbian term causes desired spikes to unconditionally increase the associated synaptic weights, and observed spikes to decrease the weights. This allows the network to undergo training even when all synaptic weights are zero and all hidden and output neurons do not fire. It also speeds up the training process [16].

Often times inhibitory synaptic connections are used. These are connections with negative weights. It has been heuristically found [16] that ignoring the weight sign, when modifying inhibitory and excitatory synaptic weights, will increase the training speed, despite the fact that this is not consistent with the gradient descent approach. As a result, the weight updates for the input-to-hidden neurons are

$$\begin{aligned}\frac{\partial w_{ih}}{\partial t} =& \frac{1}{mn} S_i(t) \sum_{o \in O} \left[\int_o^\infty a^{pre}(s)[S_o^d(t-s) - S_o^a(t-s)]ds \right] ||w_{ho}|| \\ &+ \frac{1}{mn} \sum_{o \in O} [S_o^d(t) - S_o^a(t)] \left[a + \int_0^\infty a^{post}(s) S_i(t-s) ds \right] ||w_{ho}|| \end{aligned} \tag{2.33}$$

where $||\cdot||$ denotes the absolute value.

Another useful tool in speeding up the training process is synaptic scaling. This can be used to keep the firing rates, or number of spikes, within a pre-define range. This is done by using the update step of

$$w_{ij} = \begin{cases} (1+f)w_{ij} & \text{if } w_{ij} \geq 0 \\ \frac{1}{1+f} w_{ij} & \text{if } w_{ij} < 0 \end{cases} \tag{2.34}$$

where $f > 0$ for $r_j < r_{min}$ and $f < 0$ for $r_j > r_{max}$.

Overall, ReSuMe training involves a process in which a set of hypothetical[2] teacher neurons use STDP processes to modify the weights between neurons in the network, while anti-STDP processes decrease weights after observed spiking activity. Thus it can be intuitively seen that, using ReSuMe, the firing times of the output neurons in the neural network can reliably converge to the firing times of the teacher neurons.

ReSuMe has no constraints on the number of spikes per neuron, and it's only constraint for the neuron model is a roughly linear relationship between input and output firing rate. It can also be extended to any number of hidden layers and even to recurrent structures.

Simulations show that, when applied on a feedforward neural network with a single hidden layer, the ReSuMe algorithm converges faster than Spikeprop does when conducting comparisons on benchmarks such as the Iris dataset or the XOR problem. Small changes in the number of delayed subconnections can have significant and effects on the final performance without a clear pattern or direction. In addition, training with ReSuMe shows an erratic evolution of the performance measure over training epochs, yet despite this, the weight vector is seen to progress in a consistent direction (see Fig. 8).

Overall, the ReSuMe training algorithm is very flexible as it allows for any neuron model to be used, and can ostensibly be used to train an SNN to produce arbitrary relationships between input and output spike trains. When trained on the Iris dataset, 97% accuracy was achieved on the training set and 94% on the test set after 137 epochs [16].

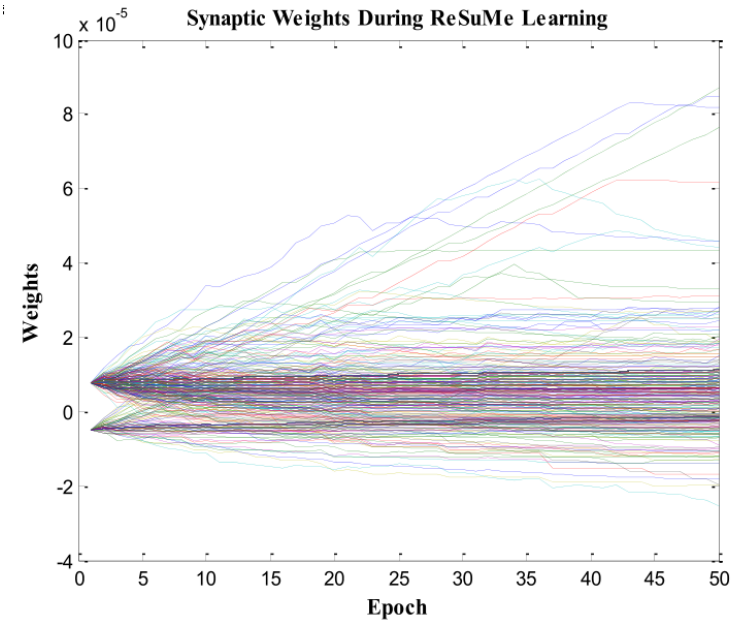


FIGURE 8.: ReSuMe weight progression: A plot of each weight value with respect to training epoch while training a neuron.

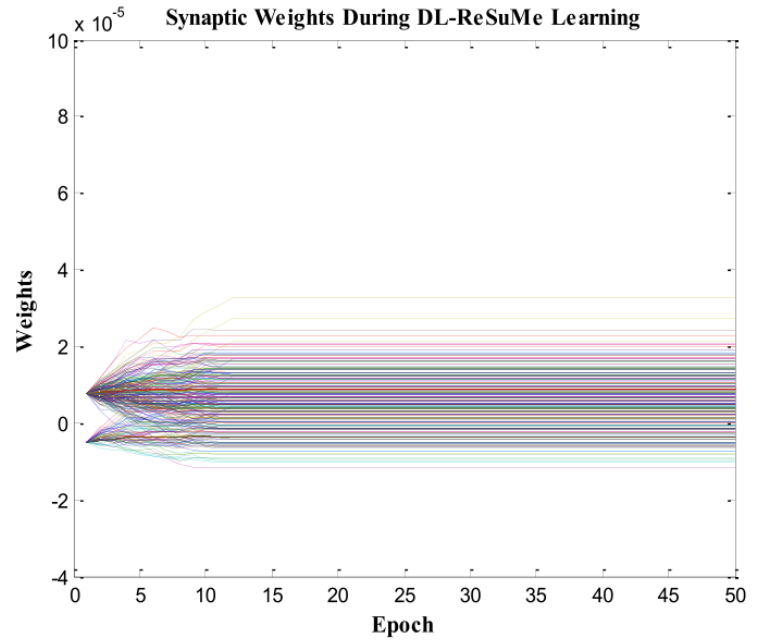


FIGURE 9.: DL-ReSuMe weight progression: A plot of each weight value with respect to training epoch while training a neuron.

DL-ReSuMe and EDL

The ReSuMe algorithm, for the case of a single layer of neurons, was later extended to include heuristic training steps that modify the pre-synaptic delays of a single neuron [58] and a single layer SNN [64]. The result is a drastic increase in training performance, training convergence rate, and consistency of the performance evolution. According to these heuristics, delays are modified in order for the neuron to fire at the desired spike times. This is done in conjunction with weight updates using Eq. 2.29. At desired spike times at which no spike is observed, the delay associated with the most recently fired excitatory synapse is modified so that the post-synaptic potential of that connection coincides with the desired spike time, and the weights are adjusted to elevate the neuron state variable to the spiking threshold. At observed spike times that do not coincide with a desired spike time, the same is done except with the most recently fired inhibitory synapse, and the weight is adjusted to bring the neuron state variable below the firing threshold. Each delay is only modified once during the entire training run.

In a simulation [58] performed to test the efficacy of this algorithm, a single neuron was used with 400 synaptic inputs. For each input synapse, a separate Poisson process was used to generate a random spike train, and likewise for the desired spike train. Training was performed using both DL-ReSuMe and ReSuMe, and the performance comparison can be seen in Fig. 10. It is evident that with the delay training heuristics the network's performance measure progresses in a much cleaner and more consistent fashion under DL-ReSuMe rather than under plain ReSuMe. When trained using DL-ReSuMe, the network ultimately reaches a higher final performance accuracy in significantly less training steps. In addition, the range of values traversed by the different weight vector components is drastically lower in DL-ReSuMe than in ReSuMe.

[2] i.e. these neurons do not have any connections in the network other than to direct weight modifications

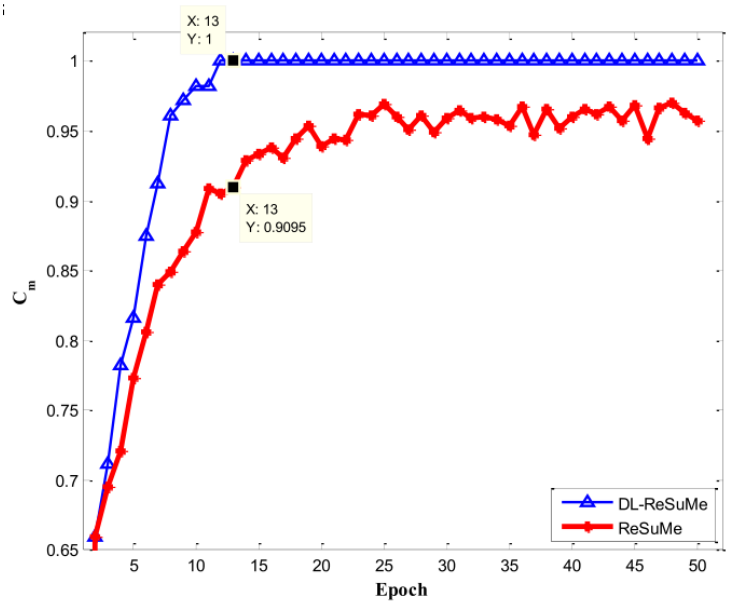


FIGURE 10.: A comparison between DL-ReSuMe and ReSuMe as applied to a single neuron receiving 400 spike trains. Input and desired output spike trains were randomly generated by Poisson processes [58]

A number of improvements [65] were later made to DL-ReSuMe, with the resulting algorithm known as Extended Delay Learning (EDL). In contrast with DL-ReSuMe, EDL allows each delay to undergo multiple changes during the training process, supports initializing the SNN with non-zero delays, and modifying the delays associated with both inhibitory and excitatory connections. These improve the performance and accuracy over DL-ReSuMe.

Classifying with Tempotron and ReSuMe

The general case of the ReSuMe weight update routines trains an SNN to fire spikes at precisely set times. However, a classification problem presents the option to record the SNN-predicted class based on which output neurons fire or don't fire. A simple training heuristic, known as the Tempotron rule [15], trains the neural network in this manner.

This heuristic, applied to the simple LIF neuron described in Eq. 2.10, is

$$\Delta w_i = \lambda \sum_{t_i < t_{\text{max}}} \alpha(t_{\text{max}} - t_i) \tag{2.35}$$

and is explicitly a gradient descent approach. Here, t_{max} is the time at which the neuron reaches its maximum voltage $v(t)$. This rule applies only to a single layer SNN, and also assumes an infinite refractory period[3]. This learning rule, which is a specific case of the ReSuMe algorithm, has also been extended to multiple layers using a simple heuristic to back-propagate error [61]. Using this multi-layer Tempotron algorithm to train an SNN for Iris classification, 98.1% accuracy was achieved on the training set and 94.7% on the test set.

The ReSuMe update rules of Eq. 2.29 can also be adapted for classification using the same principle [26], where the desired spike train is replaced by t_{max}. When this is done, and accounting for single firing, the weight update becomes

$$\Delta w_i = \frac{\lambda}{m} \left[a + \int_0^\infty a^{post}(s) S_i(t_{\text{max}} - s) ds \right], \tag{2.36}$$

where t_{max} is the time at which the membrane potential reaches its maximum value, $S_i(t)$ is a sum of Kronecker delta functions denoting the input spike train, and λ is defined as

$$\lambda = \begin{cases} -1 & \text{if the neuron fires when it should be quiescent} \\ 1 & \text{if the neuron does not fires when it should fire} \\ 0 & \text{otherwise} \end{cases} \tag{2.37}$$

This rule is known as Tempotron-like learning with ReSuMe.

[3]The neuron fires at most once.

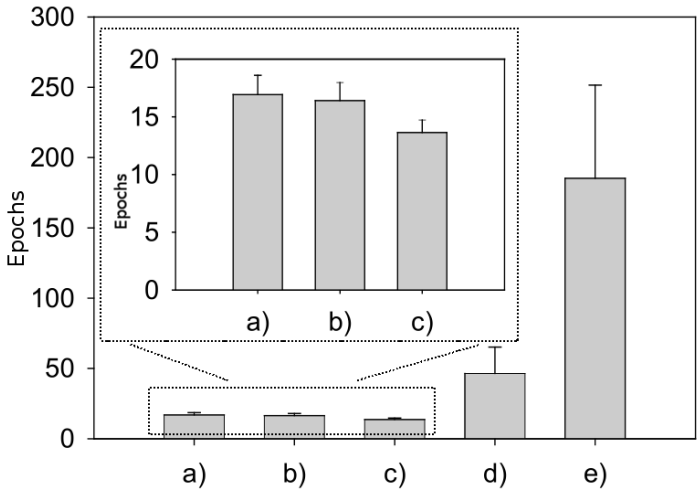


FIGURE 11.: Tempotron speed vs. ReSuMe speed: A comparison [26] between Tempotron, ReSuMe, and Tempotron-like ReSuMe as applied to a single neuron receiving spike trains of two classes. Input spike patterns were uniformly randomly generated [15, 26]. a) Original Tempotron algorithm. b) Original Tempotron algorithm, modified such that, when training away undesired spikes, $t_{\max}$ in Eq. 2.35 is replaced by the firing time. c) Tempotron-like learning with ReSuMe. d) ReSuMe training to fire once at a specific time on positive input, and to not fire on negative input. e) ReSuMe training to fire once at specific time on positive input, and to fire once at a different specific time on negative input. The inset shows the zoom-in view of a), b), and c).

There are significant advantages to using the Tempotron principle for classification as compared to training for precise spike times. Instructing neurons to simply fire or not fire, as opposed to producing spikes at specific times, involves less constraint and thus is ultimately an easier problem. This can be seen in Fig. 11, which shows the number of epochs to convergence on a simple learning problem using a) Tempotron, b) a slightly modified Tempotron, c) Tempotron-like learning with ReSuMe, and d)-e) ReSuMe. Tempotron training converges the neuron drastically faster ReSuMe, while the Tempotron-like ReSuMe learning rule was slightly faster than Tempotron alone.

SPAN and PSD

SPAN and PSD are training algorithms that take a similar approach as ReSuMe by basing the training algorithm on the WH rule. SPAN solves the problem of discontinuity by expressing neural dynamics as convolutions of incoming spikes with a continuous kernel function. This forms the basis of the energy function to be minimized, though in the actual computation of the SNN output the spikes are unchanged [62]. The PSD rule is similar to SPAN, except that only input spikes are convolved. Like Tempotron, PSD was also extended to multi-layer SNN using the same heuristic for error back-propagation [17].

Chronotron

The Chronotron refers to two different learning rules, I-learning and E-learning, both of which are single layer training algorithms. I-learning is heuristically derived, but performs slightly worse than the single layer version of ReSuMe. A neuron trained by E-learning, however, will exhibit a memory capacity that is roughly an order of magnitude greater than a neuron trained with either ReSuMe or I-learning. However, I-learning will converge faster than E-learning [66].

NSEBP

Recently, the Normalized Spiking Error Back Propagation (NSEBP) algorithm [27] was developed. It is designed for SNN of arbitrary numbers of hidden layers for a simple case of a neuron model called the Spike Response Model (SRM).

When the input spike kernel is Eq. 2.12 (see pg. 12), the SRM is analytically tractable if $\tau_1 = 2\tau_2$. Thus, when training a single neuron, an analytic formula can be used to determine the ideal shifts in pre-synaptic spike times—known as spike "jitter"—

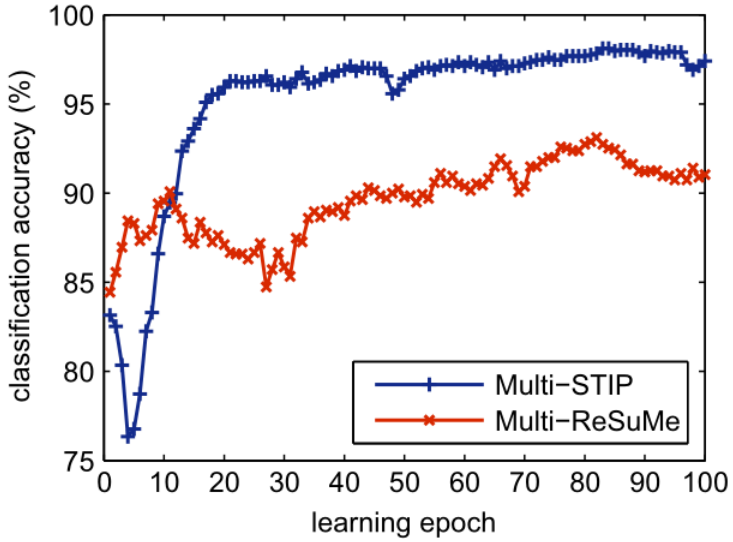


FIGURE 12.: STIP vs. ReSuMe on Iris: Training accuracy vs. epoch on a multi-layer architecture [67].

to produce the desired output spike time. This approach is carried recursively by directly training the previous neurons to produced the desired spike jitters. As a result, training with NSEBP required an order of magnitude less training epochs compared with the multi-layer ReSuMe algorithm. When training on Iris using NSEBP, 98% training accuracy and 96% test accuracy were achieved after only 18 epochs, whereas ReSuMe required 137 epochs.

Inner Products of Spike Trains

Another recent algorithm [67], referred to here as STIP, expresses the energy function as an inner product between the observed output spike train and the desired spike train. The spike train, a sum of Kronecker delta functions, is convolved with a symmetric and continuous kernel function to produce an expression that can be differentiated with respect to the weights in both the output and hidden layers.

As such, this algorithm can be applied to multi-layer SNN to solve problems such as Iris and XOR. Compared to multi-layer ReSuMe [16], training with STIP results in faster and more consistent progression of accuracy over epochs. This is demonstrated in Fig. 12. STIP also achieves a higher final performance than multi-layer ReSuMe on the Iris dataset, resulting in 96.7% accuracy on the test set as compared to 94.7%.

BP-STDP

Another approach [68] to training SNN, known as BP-STDP, is based on a mathematical proof that the non-leaky variant of the LIF neuron's membrane potential behaves similarly to a rectified linear unit (ReLU). By replacing the values of the ReLU with spike counts over time intervals, the weight update routines for training the SNN can be modeled after those for training an ANN with ReLU units. Doing this involves using STDP and anti-STDP rules to handle products of spike trains. The derived training algorithm can be applied to multi-layer SNN in which each neuron has no constraint on the numbers of spikes. Using BP-STDP, the XOR problem was solved in around 150 iterations, and a 96% test accuracy on the Iris dataset was achieved.

Multi-layer SNN with Delay Training

As relatively few studies have investigated delay training in SNN, a recent study [69] sought to apply the EDL [65] approach to a multi-layer SNN. EDL was used to train the weights and delays of the hidden-to-output neurons. Meanwhile the input-to-hidden neuron weights (not the delays) were trained with STDP and anti-STDP rules. Using this approach, 99.8% training accuracy and 95.7% test accuracy was achieved on the Iris dataset, and a smooth progression in accuracy over epochs was observed (see Fig. 13).

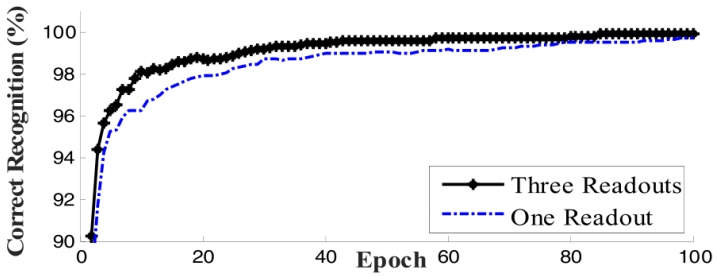


FIGURE 13.: Multilayer SNN with delay training on Iris: Training accuracy vs. epoch [67]. A multilayer SNN was trained with EDL on the hidden-to-output weights and delays, and STDP and anti-STDP rules on input-to-hidden weights. A comparison between one and three output neurons is shown.

Tab. 1 summarizes, ordered from highest to lowest test accuracy, the accuracy and training epochs of the previously discussed SNN training algorithms on the Iris dataset. The best result was achieved by STIP, while ReSuMe was the worst. Despite being one of the earliest algorithms, Spikeprop achieved one of the highest levels test accuracy, though this may be due to the enormous number of training epochs used. However, from the available data, NSEBP is by far the most efficient in terms of training time, and may easily surpass the others if trained over more epochs.

Comparison of Algorithms

Comparisons on the XOR benchmark were made between multi-layer PSD, multi-layer Tempotron, Spikeprop, multi-layer ReSuMe, and a Spikeprop implementation with a learning rate adaptation technique incorporated [70]. Both multi-layer PSD and multi-layer Tempotron converged on 100% of simulations. Spikeprop was slowest while NSEBP was much faster. This can be seen in Tab. 2. The slow convergence of Spikeprop may be due to the necessity for small learning rates, while the analytic approach of the NSEBP algorithm allows for any desired spike time shifts for a given input to be effected in one training step.

TABLE 1.: SNN classification on Iris.

	Accuracy		
Algorithm	Train	Test	Epochs
STIP [67]	0.981	0.967	-
Spikeprop [59]	0.974	0.961	1000
NSEBP [27]	0.98	0.96	18
BP-STDP [68]	-	0.960	-
Multi-DL-ReSuMe [69]	0.998	0.957	-
Multi-Tempotron [61]	0.981	0.947	-
ReSuMe [16]	0.960	0.94	174

TABLE 2.: Convergence on XOR with several SNN training algorithms.

Algorithm	Epochs
Spikeprop [59]	250
Fast Spikeprop [70]	127
Multi-layer ReSuMe [16]	137
Multi-layer PSD [17]	86
Multi-layer Tempotron [17]	37
NSEBP [27]	4-11

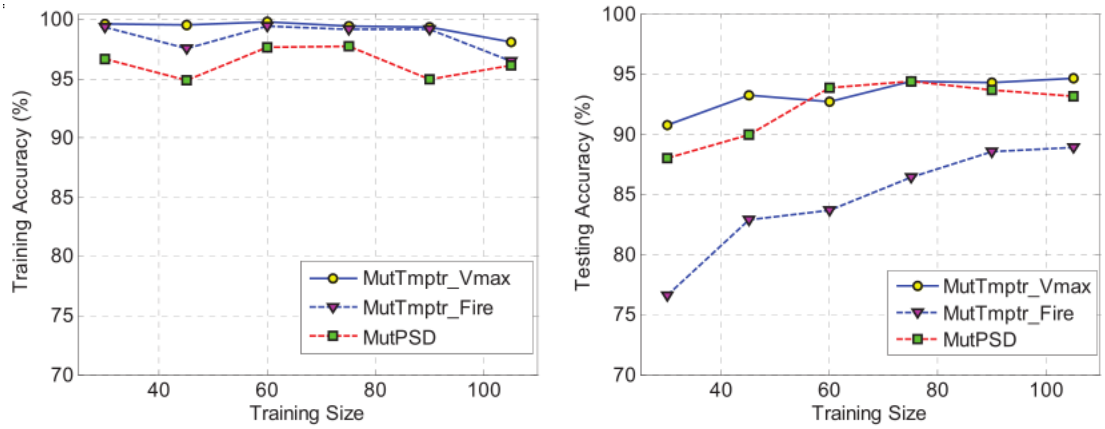


FIGURE 14.: Effect of training set size on Iris classification taken from [17]. (a) and (b) respectively show the training and testing accuracy with respect to the size of the training set.

Class Encoding

A few approaches towards output encoding can be taken when training an SNN to classify. In binary encoding, a neuron that fires represents a "1"-valued digit in the binary class label while a quiescent neuron represents a "0." When index encoding is used, output neuron i firing signifies class i. Index encoding uses the first neuron to fire for class labeling.

Index encoding allows predictions to be made based on which output neuron reaches the highest voltage (relative confidence), as opposed to which output neuron fires (absolute confidence). Fig. 14 depicts a comparison of three a multi-layer SNN classification approaches on Iris dataset [17]. Performance of the multi-layer Tempotron rule using both absolute confidence (MuTmptr_Fire) and relative confidence (MuTmptr_Vmax), and the multi-layer PSD rule (MuPSD), are compared. It is seen that relative confidence results in higher training and generalization accuracy.

TABLE 3.: UCI dataset characteristics: The TAE is the only among here with categorical attributes. The three columns under "Categorical" indicate the number of categorical attributes with number of classes as indicated by the corresponding subcolumn

		No. Original Attributes					
			Categorical				
Dataset	Datapoints	Numerical	2	25	26	Total	Classes
tae	151	1	2	1	1	5	3
iris	150	4				3	3
vertebrae	310	6				3	3
ion	351	34				2	2

UCI Datasets

The University of California at Irvine currently hosts a well known online repository of small to medium data sets to be used for machine learning benchmarking. These include databases for classifying cancer malignancies, predicting income from census information, and predicting car safety, to name a few.

Four examples of UCI datasets include the Teaching Assistant Evaluation dataset (TAE), the Iris dataset, the Vertebral Column dataset, and the Ionosphere dataset. For purposes of minimizing computation time, these datasets were selected because they were small and had low numbers of attributes.

TAE

The Teaching Assistant Evaluation dataset consists of evaluations of teaching assistants at the Statistics Department of the University of Wisconsin-Madison over five semesters, including two summer semesters. This set contains 151 datapoints with qualities about the teacher and the class, along with the three-class label of teaching

assistant's performance. It was originally developed to research bias elimination within decision tree splits [33].

Numerous classification experiments have been performed on TAE. The Buntine decision tree algorithm [71] achieved 67% accuracy on TAE [72]. Raw SVM experiments with grid search have yielded 64% accuracy, while further incorporating particle swarm optimization and feature selection techniques improved accuracy to 83% [73].

Iris

The Iris dataset [28] is a very widely used benchmark for machine learning algorithms. It consists of 150 datapoints, each containing the petal width, petal length, sepal width, and sepal length of an individual of three different species of Iris plants. Two of the classes are linearly non-separable.

SVM studies on this dataset have yielded 96% accuracy with grid search optimization, and 98% accuracy with particle swarm optimization, and further improved to 99.20% when feature selection techniques are added [73]. Recently, 100% accuracy was achieved with a back-propagation neural network (BPNN) after 10,000 training epochs [74].

Vertebrae

The Vertebral Column [30, 31, 32] dataset depicts normal and abnormal spinal anatomical features. It consists of 310 datapoints of vertebral attributes, along with classifications of normal, disk hernia, or spondilolysthesis, and was collected by Guilherme de Alencar Barreto and Ajalmar Rêgo da Rocha Neto at the Department of Teleinformatics Engineering, Federal University of Ceará in Brazil, and Henrique Antonio Fonseca da Mota Filho at the Hospital Monte Klinikum in Ceará.

TABLE 4.: Classification on UCI datasets.

	Accuracy		
Dataset	ANN	SVM	Best
tae	0.55 [78]	0.64 [73]	0.83 [73]
iris	1.00 [74]	0.96 [73]	1.00 [74]
vertebrae	0.85 [75]	0.82 [75]	0.97 [75]
ion	0.90 [77]	0.93 [73]	0.99 [73]

One experiment used 10-fold cross validation on the raw dataset and solved the 3-way classification to 85.48% accuracy using ANN and 81.61% with SVM. They further improved this to 96.77% and 96.45% by using a novel pairwise fuzzy C-means based feature set [75]. Another experiment achieved further improvements on the raw dataset: 93.55% with ANN and 86.33% with SVM, using 10-fold cross validation, although they cleaned the dataset by "handling" noisy values [76].

Ion

The Ionosphere dataset is taken from an array of high frequency antennas located in Goose Bay, Labrador, which are used as a radar system to collect data on electrical activity in the ionosphere. This dataset consists of 351 datapoints, with 33 features and 1 class value, i.e. whether suitable, or not suitable, for further investigation [29].

Basic BPNN applied to the dataset with 80/20 split between training yield an average accuracy of 90.24%, while using a CMTNN improved this to 93.43% [77]. Likewise, SVM studies on this dataset have shown 97.50% accuracy on test set when parameters are optimized using PSO, and this increases to 99.01% when features selection is added [73].

CHAPTER III

METHODS

We program a multi-layer feed-forward spiking neural network with one hidden layer and multiple delayed subconnections (see Fig. 15). The multi-layer ReSuMe training algorithm is derived for the case of multiple delayed subconnections, and reduced to the case of a classification problem by incorporating Tempotron-like learning [26]. We benchmark on the XOR problem along with four UCI datasets: Iris, Teaching Assistant Evaluation (TAE), Vertebral Column, and Ionosphere. For the XOR problem, the number of epochs until perfect accuracy is recorded for one set of parameters in order to demonstrate the correctness or our implementation. Experiments on the UCI datasets are done using cross-validation and a grid search over the numbers of hidden neurons.

Network Parameters

We first derive the mathematical expressions that govern the behavior of a spiking neural network with one hidden layer. Our feedforward neural network consists of integrate and fire neurons. We simulate the voltage $v(t)$ of a neuron in either the hidden or output layer via a weighted sum of spike kernels arriving at various times:

$$v_j(t) = \sum_i \sum_k w_{ij}^k \sum_f \alpha(t_{i,f} - d_{i,j}^k) \tag{3.1}$$

where $w_{i,j}^k$ is the weight associated with the k^{th} connection between neuron i in the previous layer, and neuron j in the next layer, and $d_{i,j}^k$ is the delay along that connection.

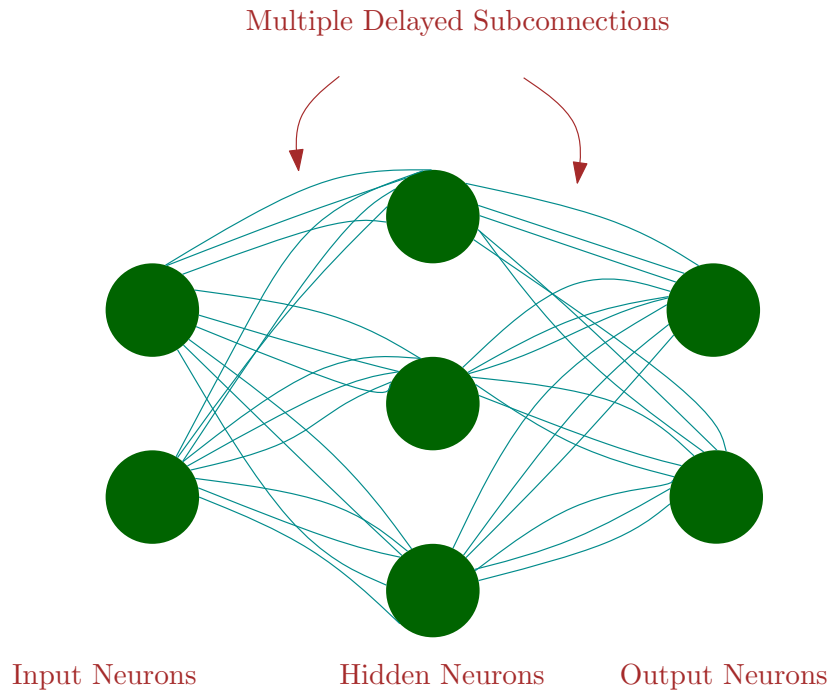


FIGURE 15.: Connectivity structure of our SNN. A feed-forward arrangement is used with multiple connections between each neuron pair. Each connection may have a different weight and delay.

We use $\alpha(t)$ to denote the spike kernel:

$$\alpha(t) = e^{-\frac{t}{\tau_1}} - e^{-\frac{t}{\tau 2}} \tag{3.2}$$

and $t_{i,f}$ to denote the f^{th} firing time of neuron i in the preceding layer. Here $\tau_1 = 5\ ms$ and $\tau_2 = 1.25\ ms$ are decay parameters selected to have a ratio of $\tau_1/\tau_2 = 4$ as was done in [16, 61]. The input layer produces a set of discrete spike times which are determined from the data.

If $v_i(t)$ reaches the threshold v_T, where $v_T = 100$, a spike time is recorded[1]. Spikes in the hidden layer cause a reset in the value of $v(t)$ to zero, and a refractory period begins.

ReSuMe Based Multi-Layer Tempotron Training Algorithm

Weight updates for hidden-to-output weights

The multi-layer ReSuMe training algorithm involves weight updates and can be derived to account for multiple delayed subconnections. The starting point for calculating the weight updates for this case are:

$$\Delta w_{ho}^{k}(t) = \delta_o(t) R_h(t) \tag{3.3}$$

$$\delta_o(t) := \frac{1}{m n_h} \left[R_o^d(t) - R_o^a(t) \right]. \tag{3.4}$$

where $\Delta_{ho}^{k} w(t)$ is the change in weight of the k^{th} subconnection between hidden neuron h and output neuron o; $R_h(t)$, $R_o^d(t)$, and $R_o^a(t)$ are the instantaneous firing rate of hidden neuron h, the desired firing rate of output neuron o, and the actual firing rate of output neuron o, respectively; m is the number of delayed subconnections between each neuron pair; and n_h is the number of hidden neurons. Eqs. 3.3 and 3.4 are Eqs. 3.28 and 3.29 from [16].

The weight update equations for multiple delayed subconnections are derived by using the same approach derived for SNN with single-synapse connections between neuron starting from these preceding two expressions as described in [16]. First the firing

[1]It must be noted that the simulated membrane potential is linearly related to input current. As such, $V_T = 100$ when the learning rate is 50 is equivalent to the case of $V_T = 1$ and the learning rate is 0.5, assuming the weights are scaled correspondingly

rates for spike-trains are substituted as

$$R(t) \rightarrow S(t), \tag{3.5}$$

which results in the weight update equation of

$$\Delta w_{ho}^{k}(t) = \frac{1}{mn_h}\left[S_o^d(t) - S_o^a(t)\right] S_h(t - d_{ho}^{k}). \tag{3.6}$$

Next, products of spike trains with the statistical model of STDP and anti-STDP relationship [63], expressed as

$$\begin{aligned} S_{post}(t)S_{pre}(t) =& S_{pre}(t)\left[a + \int_0^\infty a^{pre}(s)S_{post}(t-s)ds\right] + \\ &+ S_{post}(t)\left[a + \int_0^\infty a^{post}(s)S_{pre}(t-s)ds\right], \end{aligned} \tag{3.7}$$

are substituted into Eq. 3.6, resulting in the following expression for the weight updates for the synapses between the hidden and output layers:

$$\begin{aligned} \Delta w_{ho}^{k}(t) = &\frac{1}{mn_h} S_h(t - d_{ho}^{k}) \int_0^\infty a^{pre}(s)\left(S_o^d(t-s) - S_o^a(t-s)\right) ds + \\ &+ \frac{1}{mn_h}\left(S_o^d(t) - S_o^a(t)\right)\left[a + \int_0^\infty a^{post}(s)S_h(t - d_{ho}^{k} - s)ds\right]. \end{aligned} \tag{3.8}$$

Weight updates for input-to-hidden weights

To derive the weight update routines for the synapses between the input neurons and the hidden neurons, the starting equations are

$$\Delta w_{ih}^{k} = \delta_h(t) R_i(t - d_{ih}^{k}) \tag{3.9}$$

$$\delta_h(t) = \frac{1}{mn_i} \sum_{l,o\in O} \delta_o(t) w_{ho}^l \tag{3.10}$$

where Δw_{ih}^k is the weight change in the k^{th} delayed subconnection between input neuron i and hidden neuron h; n_i is the number of inputs; and d_{ih}^k is the delay along that subconnection. Again, after substituting rate terms into spike train terms the expression

$$\Delta w_{ih}^k = \frac{1}{m^2 n_h n_i} \sum_{l,o\in O} w_{ho}^l \left[S_o^d(t) - S_o^a(t)\right] S_i(t - d_{ih}^k) \tag{3.11}$$

is acquired.

Next, after incorporating the absolute values of the weights, as in Eq. 2.33, the final weight update expression is

$$\begin{aligned}\Delta w_{ih}^k = \frac{1}{m^2 n_i n_h} S_i(t - d_{ih}^k) \sum_{l,o\in O} ||w_{ho}^l|| \int_0^\infty a^{pre}(s) \left[S_o^d(t-s) - S_o^a(t-s)\right] ds + \\ + \frac{1}{m^2 n_i n_h} \sum_{l,o\in O} ||w_{ho}^l|| \left(S_o^d(t) - S_o^a(t)\right) \left[a + \int_0^\infty a^{post}(s) S_i(t - d_{ih}^k - s) ds\right].\end{aligned} \tag{3.12}$$

where $||\cdot||$ denotes the absolute value.

Adaptation to classification

Tempotron-like learning is applied to the derived ReSuMe weight update algorithm. This is done by updating the corresponding weights when there is a discrepancy between actual firing and desired firing of an output neuron. If the output neuron fires but is not desired to fire, then the weight is updated according to this algorithm without change. However, in cases in which the output neuron was supposed to fire, but doesn't, the time at which the neuron reaches its maximum voltage is used as the desired spike time. Using

this idea results in the definition

$$\lambda_o = \begin{cases} -1 & \text{if neuron } o \text{ fires when it should be quiescent} \\ 1 & \text{if neuron } o \text{ does not fires when it should fire} \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

Additionally, $t_{\text{max},o}$ will be defined as the time at which neuron o either reaches a maximum value of $V_o(t)$, or the time at which it spikes (and is unable to spike thereafter)

Using these definitions, the weight updates for the synapses between the hidden neurons and output neurons becomes

$$\begin{aligned} \Delta w_{ho}^k(t) = \frac{\lambda_o}{mn_h} S_h(t - d_{ho}^k) \int_0^\infty a^{pre}(s)\delta(t_{\text{max},o} - s)ds + \\ + \frac{\lambda_o}{mn_h}\delta(t_{\text{max},o} - t)\left[a + \int_0^\infty a^{post}(s)S_h(t - d_{ho}^k - s)ds\right], \end{aligned} \tag{3.14}$$

while the weight updates for synapses between input neurons and hidden neurons becomes

$$\begin{aligned} \Delta w_{ih}^k = \frac{1}{m^2 n_i n_h} S_i(t - d_{ih}^k) \sum_{l,o\in O} \lambda_o ||w_{ho}^l|| \int_0^\infty a^{pre}(s)\delta(t_{\text{max},o} - s)ds + \\ + \frac{1}{m^2 n_i n_h} \sum_{l,o\in O} \lambda_o ||w_{ho}^l|| \delta(t_{\text{max},o} - t)\left[a + \int_0^\infty a^{post}(s)S_i(t - d_{ih}^k - s)ds\right]. \end{aligned} \tag{3.15}$$

Since each neuron can only spike once, these equations reduce to

$$\Delta w_{ho}^k(t) = \frac{\lambda_o}{mn_h}\delta(t_{\text{max},o} - t)\left[a + \int_0^\infty a^{post}(s)S_h(t - d_{ho}^k - s)ds\right] \tag{3.16}$$

$$\Delta w_{ih}^k = \frac{1}{m^2 n_i n_h} \sum_{l,o\in O} \lambda_o ||w_{ho}^l|| \delta(t_{\text{max},o} - t)\left[a + \int_0^\infty a^{post}(s)S_i(t - d_{ih}^k - s)ds\right]. \tag{3.17}$$

TABLE 5.: XOR problem parameters searched: For every unique every combination of parameters, 44-45 experiments were performed using different random initialization seeds.

Hidden neurons	Delayed subconnections	Learning rate	No. random initializations
4-7	10, 12, 14	1, 10, 25, 50 100	44-45

Brian2 Simulation Library

Version 2.0.1 of the Brian2 SNN simulation library is used. Brian2 enables the use of synaptic weights and delays, and also interfaces to Numpy, Weave, or auto-generated C++ code for the underlying simulation. Brian2 also incorporates MatPlotLib functions for plotting, and allows the user to record certain neural state variables during simulation, such as spike times.

The XOR Problem

The XOR problem is a small, linearly non-separable classification problem. It consists of four data points, each with two single-digit binary features. If both binary digits are the same, the class is labeled a "1." If the binary digits differ from each other, the class is labeled "0". To encode this in a spiking neural network, input digits of type "1" are set to spike at 0 ms, while digits of type "0" are set to spike at 8 ms. The third input is a reference spike at 0 ms. The SNN uses one output neuron.

To investigate the XOR problem, an extensive parameter search space is conducted. As shown in Tab. 5, numbers of hidden neurons, numbers of delayed subconnections, and the learning rate were all varied. The subconnection delays were randomly generated to be between zero and ten milliseconds, while the weights are uniform-randomly generated between -320 and 1,280.

TABLE 6.: XOR problem encoding

inputs (ms)			output
0	8	0	silent
8	0	0	silent
8	8	0	fire
0	0	0	fire

For each combination of parameters, 44-45 experiments with different random initialization seeds were conducted. Training was performed until either validation accuracy reached 100%, or 250 epochs were traversed. The number of epochs until convergence was recorded. The optimal parameters for fast and reliable convergence were searched for.

Training and testing are performed each epoch until the network exhibits the behavior seen in Tab. 6 on the test runs.

UCI Dataset Experiments

The network architecture was also used to classify on the four UCI datasets listed in chapter 2. Each dataset was split into separate sets for training, validation, and testing, and the results averaged using 11-fold cross validation for the Vertebral Column dataset, and 10-fold cross validation for the other three datasets. To prepare each set, the data is split into separate sets for each class, after which they were each scrambled and split into their corresponding number of folds: 1 fold each for validation and testing, and the rest of the folds for training. The corresponding training, validation, and test sets for each class were recombined and scrambled again for the training process. The TAE dataset included attributes with categorical data, while all other datasets had only numerical attributes.

Encoding Input Features to Spike Times

Categorical and numerical attributes were treated differently in the input encoding process. Two separate experiments were performed on TAE dataset, one using binary encoding, and the other using index encoding, for input categorical features.

Binary encoding is performed by setting the number of input neurons for a categorical attribute to the lg of the number of categories. The input firing pattern is then set such that the "on" and "off" neurons form the binary representation of the category index. Index encoding is performed by using the number of categories as the number of neurons for that attribute. The index of the category then corresponds to the index of the "on" neuron, whereas the rest were "off". In all cases, "on" neurons fire at time $t = 0$ ms while "off" neurons fire at time $t = 8$ ms.

Encoding numerical attributes is simpler. Each numerical input attribute is linearly normalized, independently of the other features, across all datapoints to a range of time $t = 0$ ms to $t = 8$ ms. Each attribute is then linearly reversed so that a higher number in a particular numerical attribute implies a sooner input spike time encoding. Additionally, in order to give absolute reference for each dataset, input spikes at 0 ms were added.

After converting input datapoints to spike times, the experiment was conducted by training with the Tempotron-like ReSuMe algorithm for a maximum of 500 epochs. After each epoch, the training, validation, and testing accuracy are recorded. The training algorithm finishes if 500 epochs were computed, with no early stopping criteria. After termination, the epoch with maximum performance is determined. The values for training, validation, and test accuracy at this epoch are recorded as the final values.

However, determining the optimal epoch proved somewhat ambiguous. This was due to the erratic behavior seen in the accuracy-vs-epoch graph. For instance, over one epoch, the training accuracy may drop significantly while validation accuracy jumps up,

and visa versa. This is consistent with the known behavior of an SNN being trained with the ReSuMe algorithm [16]. In addition, the relative smallness of the validation sets made them prone to such rapid changes. Therefore, for each training run, the final recorded train, validation, and test accuracy was selected based on the minimization

$$\text{Epoch}_{\text{opt}} = \text{argmin}[(1 - y_{\text{train}})^2 + (1 - y_{\text{val}})^2 + 4(y_{\text{val}} - y_{\text{train}})^2]. \tag{3.18}$$

Here y_{train} and y_{val} are arrays of training and validation accuracy at each epoch, respectively. $\text{Epoch}_{\text{opt}}$ is the epoch at which final values for train, validation, and test accuracies are reported. Elimination of the second and third terms will result in the classic fitness function for minimizing training error. However, Eq. 3.18 was selected to minimize training error, validation error, and the difference between the two, in an attempt to acquire good results during an erratic evolution of accuracy over epochs. In addition, Eq. 3.18 contains no information on test accuracy. Once the epoch is selected, the training, validation, and test accuracy at that epoch are recorded for the particular cross validation fold and set of parameters.

In the UCI dataset experiments, a grid search with 5, 6, and 7 hidden neurons was performed. The only exception is the Iris dataset, which ranged from 4 to 8 hidden neurons. In all experiments, 10 subconnections with delays uniformly randomly distributed between 0 and 11 ms were used, and weights were uniformly and randomly initialized within a range of -320 to 1,280. This was done in order for 20% of weights to be inhibitory, while the magnitude of the range facilitated initial spiking with a large voltage threshold in the neuron units. All folds in a cross-validation experiment used the same initialization seed and hence the same initial synaptic weights. For each number of hidden neurons, 5 trials of cross-validation were performed using different initialization seeds.

Just as is done with the XOR simulation setup, additional input neurons firing at 0 ms were added so that linear offsets can be distinguishable.

Comparison Experiments

Predictions on all four UCI datasets were also made using an SVM with RBF kernel. To optimize SVM parameters, both a grid search and a particle swarm optimization (PSO) algorithm was used. For both, in order to search a wide range without exorbitant amount of experimental runs, parameters C and γ varied along an exponential scale:

$$C = 10^i \tag{3.19}$$

$$\gamma = 10^j/N \tag{3.20}$$

where N is the number of input features, and i and j are parameters set by the optimization algorithm. For grid search, the variables i and j were searched in the range (-3,7) inclusively over 26 steps in order to balance both a large range of values and small spacing. With PSO experiments, i and j were constrained within the same range using a population size of 150 over 300 iterations. Optimal parameters were determined based on the fitness function Eq. 3.18.

Inputs to SVM were the additive inverse of inputs to SNN scaled to interval of (0-1). This was done because, for SNN inputs, smaller numbers correspond to later spikes, while stronger stronger inputs correspond to earlier spikes. With SVM, the opposite is true.

CHAPTER IV

RESULTS

The Tempotron-like ReSuMe training algorithm [26] has been adapted for the multi-layer ReSuMe training algorithm [16] and applied to a three-layer classifier with multiple delayed subconnections. Using this, the XOR problem was solved and varying levels of accuracy were seen on the four UCI datasets. Experiments on UCI datasets have totaled 690 data runs, and computations took roughly 6 weeks on two 3.40GHz Intel(R) Core(TM) i7-3770 quad-core processors (totaling 16 logical processors). The extensive grid search on the XOR problem involved 2685 experiments total.

The XOR Problem

The XOR problem was solved over an extensive parameter space. Numbers of hidden neurons ranged from 4 to 7; delayed subconnections had values of 10, 12, and 14; and the learning rate took values of 1, 10 25, 50, and 100[1]. Tab. 7 shows convergence statistics for the optimal combination of parameters over 44 trial runs. On average just under 14 epochs were needed to find the solution.

[1]These may seem like high learning rates, but with spike threshold of 100, a learning rate of 50 corresponds to a learning rate of 0.5 with a spike threshold of 1.

TABLE 7.: XOR convergence statistics using optimal parameters of 7 hidden neurons, 10 delayed sub-connections, and a learning rate of 50.

Epochs to solve				
Mean	Stdev	Min	Max	Convergence Rate (%)
13.73	6.70	3	35	100

TABLE 8.: XOR comparison: Convergence of several algorithms on the XOR problem. Bold indicates our experiment. In all cases, the neuron model consisted of a linear sum of the spike kernel. Models using spike kernel of $\alpha = \frac{t}{\tau}\exp(1-\frac{t}{\tau})$ are denoted by SRM, while models using $\alpha = \exp(t/\tau_1) - \exp(t/\tau_2)$ are denoted by LIF. In all cases, the neuron model was the simplest case as denoted by Eq. 2.10.

Algorithm	Spike Kernel	Epochs
Spikeprop [59]	SRM	250
Multi-Layer ReSuMe [16]	SRM	137
Fast Spikeprop [70]	SRM	127
Multi-Layer PSD [17]	LIF	86
Multi-Layer Tempotron [17]	LIF	37
ReSuMe-like Tempotron	LIF	14
NSEBP [27]	LIF	4

Tab. 8 compares various algorithms on the XOR problem. Ours converges faster than all listed except NSEBP. Differences in efficiency across experiments may potentially be due to differences in extensiveness of the parameter search space, or differences in the neural parameters used. All experiments listed in Tab. 8 use a linear sum of pre-synaptic kernels, as opposed to more complex, non-linear differential equation. The only difference is the presynaptic kernel themselves. Those listed with the SRM kernel use

$$\alpha = \frac{t}{\tau}\exp(1-\frac{t}{\tau}) \tag{4.1}$$

while those listed as LIF use

$$\alpha = \exp(\frac{t}{\tau_1}) - \exp(\frac{t}{\tau_2}) \tag{4.2}$$

TABLE 9.: Neuron parameters on XOR: Comparison of algorithms on XOR problem. Bold indicates our experiment.

Algorithm	Spike Kernel	parameters
Spikeprop [59]	SRM	$\tau = 7\text{ms}$
Multi-Layer ReSuMe [16]	SRM	$\tau = 7\text{ms}$
Fast Spikeprop [70]	SRM	$\tau = 7\text{ms}$
Multi-Layer PSD [17]	LIF	$\tau_1 = 7\text{ms}$, $\tau_2 = 1.75\text{ms}$
Multi-Layer Tempotron [17]	LIF	$\tau_1 = 7\text{ms}$, $\tau_2 = 1.75\text{ms}$
ReSuMe-like Tempotron	LIF	$\tau_1 = 5\text{ms}$, $\tau_2 = 1.25\text{ms}$
NSEBP [27]	LIF	$\tau_1 = 5\text{ms}$, $\tau_2 = 2.50\text{ms}$

In each, the time constants can be modified to make both kernels very similar in shape. The studies listed as SRM used $\tau = 7\text{ms}$, while those listed as LIF used different parameters.

UCI Dataset Experiments

Results on all four UCI datasets for all parameters, averaged across five trials of k-fold cross-validation, are shown in Fig. 16. Additionally, Fig. 17 shows numbers of epochs until values were recorded according to Eq. 3.18, averaged over all experiments for each dataset. In both cases, middle ticks correspond to mean accuracy while error ranges correspond to one standard deviation. A numerical summary of train, validation, and test accuracy, for optimal numbers of hidden neurons, is shown in Tab. 10. Significant overfitting can be seen in all experiments, while training and validation accuracies are close to each other. There is some correlation between number of hidden neurons and final performance in all datasets, especially Iris. Training runs on the Iris dataset converged faster with higher numbers of hidden neurons, while convergence epochs with

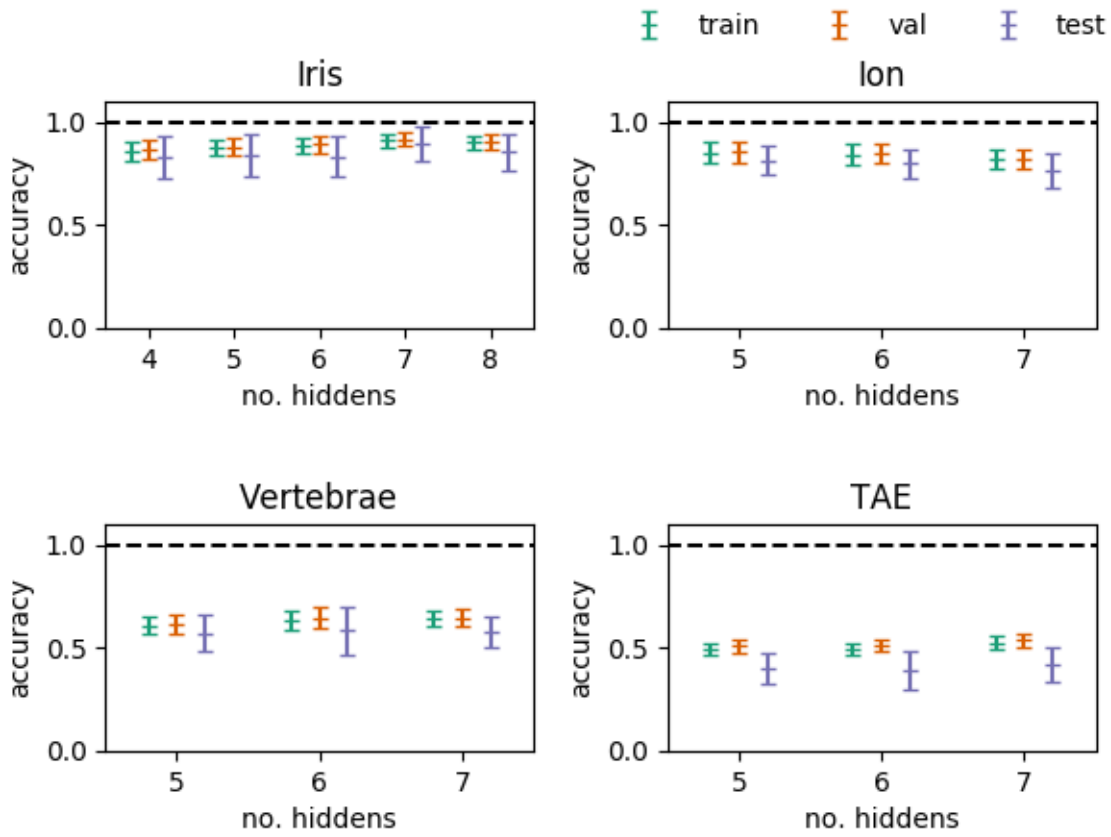


FIGURE 16.: All UCI datasets. All UCI datasets. Middle ticks correspond to mean accuracy while error ranges correspond to one standard deviation.

other datasets did not show this phenomenon. Additionally, extremely high variance in the number of training steps is seen.

Iris

The Iris dataset contains 150 datapoints in 3 classes. 10-fold cross validation was performed, and datapoints were split evenly between 3 classes: 40 data points per class were used for training, 5 for validation, and 5 for testing.

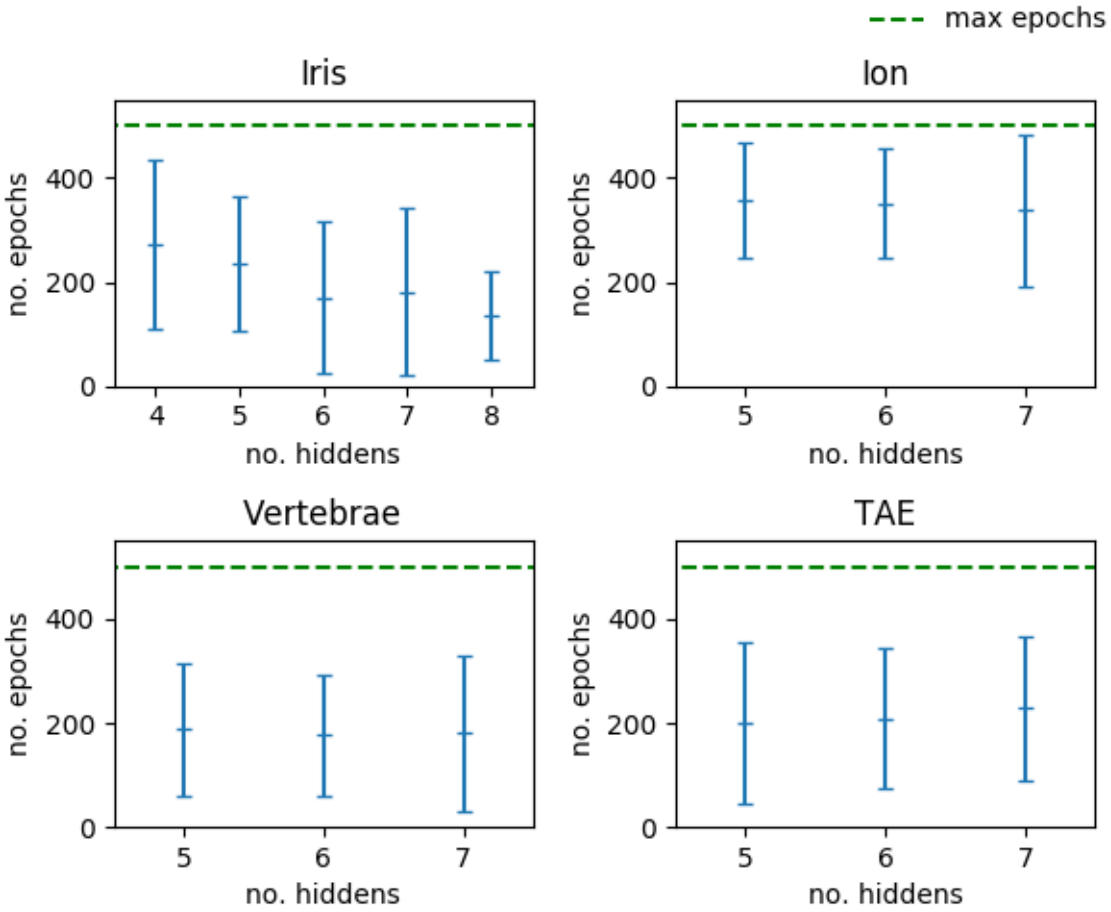


FIGURE 17.: Boxplot: Epochs until recorded. Middle ticks correspond to mean epochs while error ranges correspond to one standard deviation.

Looking at Fig. 16 we see that test accuracy has a higher variance and lower mean than both validation and training accuracy, while interestingly, validation accuracy is slightly higher than training accuracy. This latter fact may be due to the relative smallness of the validation set compared to the training set, which could allow for greater variation in validation accuracy over its evolution vs. epoch than training accuracy. This would give more leeway for the second term in Eq. 3.18 to be optimized. Overall, accuracy peaks at seven hidden neurons.

TABLE 10.: SNN accuracy on UCI datasets, averaged over five iterations of k-fold cross-validation. The second column indicates the number of hidden neurons that yielded optimal value of the fitness function (Eq. 3.18).

	Hidden	Accuracy			
Dataset	Neurons	Train	Validation	Test	Epochs
iris	7	0.911±0.032	0.917±0.032	0.893±0.084	182±160
ion	5	0.848±0.052	0.853±0.047	0.815±0.074	359±110
vertebra	7	0.642±0.039	0.646±0.044	0.577±0.074	181±151
tae	7	0.526±0.039	0.540±0.041	0.420±0.102	229±138

Ion

The Ionosphere dataset contains 351 data points. Using 10-fold cross validation, 273 datapoints were allocated for training, and 39 for each, validation and testing.

Accuracy is consistently above 75%, and sometimes above 80%. There is some overfitting with respect to training and testing sets, while again, accuracy on the validation set is often greater than on the training set. No obviously consistent correlation between numbers of hidden neurons and overall accuracy can be seen.

Vertebrae

Vertebral Column is a 3-class dataset containing 310 datapoints. Using 11-fold cross validation, 248 datapoints were allocated for training, and 31 datapoints for each, validation and testing.

Test accuracy on the Vertebral Column hovers around 60%. In all but one case, mean test accuracy is consistently lower than training accuracy, while again, validation accuracy is consistently higher than training accuracy except in one case.

TAE

TAE is a 3-class dataset containing 151 datapoints. 10-fold cross validation was used. As folds had slightly different numbers of data points for each class, training sets ranged from 115 to 121 datapoints, while validation and test sets ranged from 15 to 18 datapoints. Additionally, binary encoding resulted in slightly higher accuracy than index encoding with SNN computations, and much higher accuracy than index encoding for SVM computations. Therefore all TAE results listed here use binary encoding.

TAE shows high variance in test accuracy, while training and validation accuracy show narrower ranges. The consistent trend of underfitting with respect to training and validation accuracy is especially pronounced for TAE. Test accuracy tends to be significantly lower than train and validation accuracy, which shows the poor performance of the classifier on this dataset.

Comparison with SVM and Literature

Tab. 11 shows the comparison between the overall test sets from the optimal parameters in both SNN computations and SVM computations, as determined by training and validation accuracy. Datasets are ordered from easiest (top) to hardest (bottom). Evidently the Iris dataset was easiest for both our algorithms and published research, while TAE dataset was the most difficult.

Our SNN performed best on Iris and Ionosphere. While it outperformed in-house SVM experiments on Iris with respect to accuracy, it by no means reaching the state-of-the-art in vanilla algorithms[2]. In addition, accuracy performance on TAE was extremely poor, averaging at around 42%, which is a little better than a random classifier.

[2]No preprocessing

TABLE 11.: Test accuracy on UCI datasets for both SNN and SVM. For each dataset, the experiment resulting in the lowest value of the argument to Eq. 3.18 was used for final reporting. Bold column labels indicate in-house experiment.

Dataset	Hidden Neurons	**SNN**	**SVM (PSO)**	**SVM (Grid)**	SVM	ANN
iris	7	0.893±0.084	0.868±0.293	0.869±0.294	0.96 [73]	1.00 [74]
ion	6	0.815±0.074	0.847±0.301	0.843±0.299	0.93 [73]	0.90 [77]
vertebra	6	0.577±0.074	0.768±0.262	0.774±0.264	0.82 [75]	0.94 [76]
tae	6	0.420±0.102	0.513±0.208	0.496±0.206	0.64 [73]	0.55 [78]

TABLE 12.: SNN classification on Iris. Bold indicates our experiment.

	Accuracy		
SNN Algorithm	Train	Test	Epochs
STIP [67]	0.981	0.967	-
Spikeprop [59]	0.974	0.961	1000
NSEBP [27]	0.98	0.96	18
BP-STDP [68]	-	0.960	-
Multi-DL-ReSuMe [69]	0.998	0.957	-
Multi-Tempotron [61]	0.981	0.947	-
ReSuMe [16]	0.960	0.94	174
ReSuMe-like Tempotron	0.911	0.893	182

Comparison With Other SNNs on Iris

A comparison of performance on the Iris dataset between in-house SNN experiments and those of SNN training algorithms taken from the literature is shown in Tab. 12. Our experiment performs worst of all listed. However, it must be noted that the original experiments of the multi-layer ReSuMe training algorithms found significant, random changes in test accuracy when varying the numbers of delayed subconnections. With 9 delayed subconnections, test accuracy was 94%, while with 10 delayed subconnections, test accuracy was 89%. Because computation time was prohibitive (6 weeks), and several reruns needed to be made to correct mistakes, it was

impractical to perform a comprehensive grid search over all the important parameters. These results must therefore be counted as preliminary. Additionally, a different input encoding scheme, such as the gaussian based population encoding seen in [59], may improve results.

CHAPTER V

CONCLUSION

We have taken an approach to optimizing an SNN for classification and adapted it to the multi-layer case using readily available algorithmic methods. Our SNN has solved several classic numerical benchmark problems. After a thorough parameter search, we have improved upon previous results on the XOR benchmark where the number of epochs until full recognition is the concern. On the UCI datasets, however, our SNN performed best on Iris and Ionosphere, outperforming in-house SVM experiments on Iris with respect to accuracy. However, on the other datasets cases it did worse than our SVM experiments, and in all cases our SNN was less accurate than ANN and SVM experiments, or SNN experiments, from the literature. Our SNN accuracy was also especially poor on the Vertebral Column and TAE datasets. More tuning of the SNN may improve results significantly.

SNN training with this setup shows that training effectiveness is effected by the profile of initial synaptic weights and/or delays. The presence of multiple delayed subconnections spreads out the arrival times of pre-synaptic spikes. This allows for greater ranges of potential output spike times given the appropriate synaptic weights. However, cases in which a spike is desired to be produced at a time in which the incoming pre-synaptic kernels are relatively sparse would necessitate a correspondingly large change in synaptic weights. This phenomenon can be seen in Fig. 8 where, although weight progression during training shows consistent direction, some weights seem to grow indefinitely. This problem with a lack of homogeneous 'supply' of pres-synaptic kernels, however, is solved to some extent in our case since the desired spike time is set to the point at which the membrane potential is at its maximum. This can be seen in the

fact that Tempotron training, and Tempotron-like ReSuMe training, are both much more efficient than ReSuMe alone when training for the single-spike classification problem. However, it is possible that sparsity of pre-synaptic kernels may be improved upon. This may be done by incorporating appropriate delay training. The principle cause by which delay training influences spiking behavior seems to be the coordination of the synchrony of incoming spikes, and the reduction of extent to which weight values need to change in order to produce desired changes in spike times.

As seen in our improvements on the XOR problem, the multilayer ReSuMe-like Tempotron algorithm may have potential. If further investigations demonstrate its capability to train SNN with high accuracy on meaningful datasets, then research may be conducted on the feasibility of implementing the learning algorithm efficiently in hardware. Applications that depend crucially on energy efficiency therefore may be best suited with an SNN solution in hardware.

Future Work

A comprehensive grid search over the parameters used for classification on the UCI datasets still needs to be performed to produce a final and reliable expression for the performance of this algorithm. Varying parameters should include the numbers of delayed subconnections, the range of delays allowed, and the learning rate parameters. In addition, different feature encoding schemes may have better results. It may also be useful to investigate any bias in the fitness function.

The ReSuMe training algorithm was derived based on the assumption of a linear relationship between firing rates of connected neurons. A similar approach may be taken to derived a general formula that includes delay training. A gradient-based multi-layer delay training algorithm would have to find an energy function that describes the

efficiency by which synaptic kernels coincide with desired spike times in the hidden or output neurons. Finding this will be left for future work.

REFERENCES CITED

[1] D. Perrett, E. Rolls, and W. Caan, "Visual neurones responsive to faces in the monkey temporal cortex." *Experimental Brain Research*, vol. 47, no. 3, pp. 329–342, 1982.

[2] S. J. Thorpe and M. Imbert, "Biological constraints on connectionist modelling." *Connectionism in Perspective*, pp. 63–92, 1989.

[3] R. V. Rullen and S. J. Thorpe, "Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex." *Neural Computation*, vol. 13, no. 6, pp. 1255–1283, 2001.

[4] W. Gerstner, "Time structure of the activity in neural network models." *Physical Review E*, vol. 51, no. 1, p. 738, 1995.

[5] A. Borst and F. E. Theunissen, "Information theory and neural coding." *Nature Neuroscience*, vol. 2, no. 11, p. 947, 1999.

[6] A. N. Burkitt, "A review of the Integrate-and-Fire neuron model: I. Homogeneous synaptic input." *Biological Cybernetics*, vol. 95, no. 1, pp. 1–19, 2006.

[7] N. K. Kasabov, "NeuCube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data." *Neural Networks*, vol. 52, pp. 62–76, 2014.

[8] B. Meftah, O. Lézoray, S. Chaturvedi, A. A. Khurshid, and A. Benyettou, "Image processing with spiking neuron networks," in *Artificial Intelligence, Evolutionary Computing and Metaheuristics*, pp. 525–544, Springer, 2013.

[9] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network." *Neuromorphic Engineering Systems and Applications*, p. 61, 2015.

[10] Q. Yu, H. Tang, K. C. Tan, and H. Li, "Rapid feedforward computation by temporal encoding and learning with spiking neurons." *Neural Networks and Learning Systems. IEEE Transactions on*, vol. 24, no. 10, pp. 1539–1552, 2013.

[11] J. Dennis, T. H. Dat, and H. Li, "Combining robust spike coding with spiking neural networks for sound event classification." in *Acoustics, Speech and Signal Processing (ICASSP). 2015 IEEE International Conference on*, pp. 176–180, IEEE, 2015.

[12] S. M. Bohte, H. L. Poutré, and J. N. Kok, "Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks." *Neural Networks. IEEE Transactions on*, vol. 13, no. 2, pp. 426–435, 2002.

[13] A. Zhang, W. Zhu, and M. Liu, "Self-organizing reservoir computing based on spiking-timing dependent plasticity and intrinsic plasticity mechanisms." in *Chinese Automation Congress (CAC), 2017*, pp. 6189–6193, IEEE, 2017.

[14] J. Xin and M. J. Embrechts, "Supervised learning with spiking neural networks." in *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, vol. 3, pp. 1772–1777, IEEE, 2001.

[15] R. Gütig and H. Sompolinsky, "The Tempotron: a neuron that learns spike timing–based decisions." *Nature Neuroscience*, vol. 9, no. 3, pp. 420–428, 2006.

[16] I. Sporea and A. Grüning, "Supervised learning in multilayer spiking neural networks.." *Neural Computation*, vol. 25, pp. 473–509, 2013.

[17] Q. Yu, H. Tang, J. Hu, and K. C. Tan, "Temporal learning in multilayer spiking neural networks through construction of causal connections," in *Neuromorphic Cognitive Systems*, pp. 115–129, Springer, 2014.

[18] N. Anwani and B. Rajendran, "NormAD-Normalized Approximate Descent based supervised learning rule for spiking neurons." in *Neural Networks (IJCNN). 2015 International Joint Conference on*, pp. 1–8, IEEE, 2015.

[19] W. Maass, "Lower bounds for the computational power of networks of spiking neurons." *Neural Computation*, vol. 8, no. 1, pp. 1–40, 1996.

[20] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm." in *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pp. 1–4, IEEE, 2011.

[21] J. Binas, D. Neil, G. Indiveri, S.-C. Liu, and M. Pfeiffer, "Precise deep neural network computation on imprecise low-power analog hardware." *arXiv preprint arXiv:1606.07786*, 2016.

[22] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface." *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[23] 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." in *Neural Networks (IJCNN). 2015 International Joint Conference on*, pp. 1–8, IEEE, 2015.

[24] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition." *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[25] D. Neil, M. Pfeiffer, and S.-C. Liu, "Learning to be efficient: Algorithms for training low-latency, low-compute deep spiking neural networks." 2016.

[26] R. V. Florian, "Tempotron-like learning with resume," in *Artificial Neural Networks-ICANN 2008*, pp. 368–375, Springer, 2008.

[27] X. Xie, H. Qu, G. Liu, M. Zhang, and J. Kurths, "An efficient supervised training algorithm for multilayer spiking neural networks." *PLoS ONE*, vol. 4, 2016.

[28] R. Fisher, "The use of multiple measurements in taxonomic problems." *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[29] V. G. Sigillito, S. P. Wing, L. V. Hutton, and K. B. Baker, "Classification of radar returns from the ionosphere using neural networks." *Johns Hopkins APL Technical Digest*, vol. 10, no. 3, pp. 262–266, 1989.

[30] J. R. P. L. H. Berthonnaud, Eric; Dimnet, "Analysis of the sagittal balance of the spine and pelvis using shape and orientation parameters." *Journal of Spinal Disorders & Techniques*, vol. 18, pp. 44–47, February 2005.

[31] A. R. da Rocha Neto and G. de Alencar Barreto, "On the application of ensembles of classifiers to the diagnosis of pathologies of the vertebral column: A comparative analysis." *IEEE Latin America Transactions*, vol. 7, no. 4, pp. 487–496, 2009.

[32] A. R. da Rocha Neto, R. Sousa, G. d. A. Barreto, and J. S. Cardoso, "Diagnostic of pathology on the vertebral column with embedded reject option." in *Iberian Conference on Pattern Recognition and Image Analysis*, pp. 588–595, Springer, 2011.

[33] W.-Y. Loh and Y.-S. Shih, "Split selection methods for classification trees." *Stistica Sinica*, vol. 7, pp. 815–840, 1997.

[34] Y. LeCun *et al.*, "Generalization and network design strategies." *Connectionism in Perspective*, pp. 143–155, 1989.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks." in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification." in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[37] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition." *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30–42, 2012.

[38] J. Khan, J. S. Wei, M. Ringner, L. H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. R. Antonescu, C. Peterson, *et al.*, "Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks." *Nature Medicine*, vol. 7, no. 6, p. 673, 2001.

[39] Z.-H. Zhou, Y. Jiang, Y.-B. Yang, and S.-F. Chen, "Lung cancer cell identification based on artificial neural network ensembles." *Artificial Intelligence in Medicine*, vol. 24, no. 1, pp. 25–36, 2002.

[40] E. Eidinger, R. Enbar, and T. Hassner, "Age and gender estimation of unfiltered faces." *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 12, pp. 2170–2179, 2014.

[41] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators." *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[42] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function." *Neural Networks*, vol. 6, no. 6, pp. 861–867, 1993.

[43] A. Sarangi and A. Bhattacharya, "Comparison of artificial neural network and regression models for sediment loss prediction from Banha watershed in India." *Agricultural Water Management*, vol. 78, no. 3, pp. 195–208, 2005.

[44] L. Zhang and B. Zhang, "A geometrical representation of McCulloch-Pitts neural model and its applications." *Neural Networks, IEEE Transactions on*, vol. 10, no. 4, pp. 925–929, 1999.

[45] R. Kohut and B. Steinbach, "Boolean neural networks." *Transactions on Systems*, no. 2, pp. 420–425, 2004.

[46] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," Tech. Rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[47] N. C. Institute, "Nerve tissue." http://training.seer.cancer.gov/anatomy/nervous/tissue.html. Accessed May 9, 2018.

[48] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve." *The Journal of Physiology*, vol. 117, no. 4, p. 500, 1952.

[49] E. M. Izhikevich, "Which model to use for cortical spiking neurons?." *Neural Networks, IEEE Transactions on*, vol. 15, no. 5, pp. 1063–1070, 2004.

[50] E. M. Izhikevich *et al.*, "Simple model of spiking neurons." *Neural Networks, IEEE Transactions on*, vol. 14, no. 6, pp. 1569–1572, 2003.

[51] M. Stimberg, D. F. M. Goodman, V. Benichoux, and R. Brette, "Equation-oriented specification of neural models for simulations." *Frontiers in Neuroinformatics*, vol. 8, no. 6, 2014.

[52] J. Hsu, "IBM's new brain [News]." *IEEE Spectrum*, vol. 51, no. 10, pp. 17–19, 2014.

[53] M. A. C. Maher, S. P. Deweerth, M. A. Mahowald, and C. A. Mead, "Implementing neural architectures using analog VLSI circuits." *Circuits and Systems, IEEE Transactions on*, vol. 36, no. 5, pp. 643–652, 1989.

[54] S. Lim, J.-H. Bae, J.-H. Eum, S. Lee, C.-H. Kim, B.-G. Park, and J.-H. Lee, "Adaptive learning rule for hardware-based deep neural networks using electronic synapse devices." *arXiv Preprint arXiv:1707.06381*, 2017.

[55] A. Belatreche, L. Maguire, M. McGinnity, and Q. Wu, "A method for supervised training of spiking neural networks." in *Proc. IEEE Conf. Cybernetics Intelligence–Challenges and Advances, CICA*, pp. 39–44, Citeseer, 2003.

[56] J.-P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner, "Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning." *Neural Computation*, vol. 18, no. 6, pp. 1318–1348, 2006.

[57] B. Gardner, I. Sporea, and A. Grüning, "Encoding spike patterns in multilayer spiking neural networks." *arXiv preprint arXiv:1503.09129*, 2015.

[58] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "DL-ReSuMe: A delay learning-based remote supervised method for spiking neurons." *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 26, no. 12, pp. 3137–3149, 2015.

[59] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons." *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

[60] B. Schrauwen and J. Van Campenhout, "Improving spikeprop: enhancements to an error-backpropagation rule for spiking neural networks." in *Proceedings of the 15th ProRISC workshop*, vol. 11, 2004.

[61] 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, 2013.

[62] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, "Training spiking neural networks to associate spatio-temporal input–output spike patterns." *Neurocomputing*, vol. 107, pp. 3–10, 2013.

[63] W. M. Kistler and J. L. Van Hemmen, "Modeling synaptic plasticity in conjunction with the timing of pre-and postsynaptic action potentials." *Neural Computation*, vol. 12, no. 2, pp. 385–405, 2000.

[64] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "Multi-DL-ReSuMe: Multiple neurons delay learning remote supervised method." in *Neural Networks (IJCNN), 2015 International Joint Conference on*, pp. 1–7, IEEE, 2015.

[65] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "EDL: an extended delay learning based remote supervised method for spiking neurons." in *Neural Information Processing*, pp. 190–197, Springer, 2015.

[66] R. V. Florian, "The Chronotron: A neuron that learns to fire temporally precise spike patterns." *PLOS ONE*, vol. 7, no. 8, p. e40233, 2012.

[67] X. Lin, X. Wang, and Z. Hao, "Supervised learning in multilayer spiking neural networks with inner products of spike trains." *Neurocomputing*, vol. 237, pp. 59–70, 2017.

[68] A. Tavanaei and A. S. Maida, "BP-STDP: Approximating backpropagation using spike timing dependent plasticity." *arXiv preprint arXiv:1711.04214*, 2017.

[69] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "A supervised learning algorithm for learning precise timing of multiple spikes in multilayer spiking neural networks." *Neural Networks and Learning Systems. IEEE Transactions on*, 2018.

[70] S. McKennoch, D. Liu, and L. G. Bushnell, "Fast modifications of the spikeprop algorithm." in *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pp. 3970–3977, IEEE, 2006.

[71] W. Buntine, "Learning classification trees." *Statistics and Computing*, vol. 2, pp. 63–73, 1992.

[72] T.-S. Lim, W.-Y. Loh, and Y. Shih, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms." *Machine Learning*, vol. 40, pp. 203–228, 2000.

[73] S.-W. Lin, K.-C. Ying, S.-C. Chen, and Z.-J. Lee, "Particle swarm optimization for parameter determination and feature selection of support vector machines." *Expert Systems with Applications*, vol. 35, no. 4, pp. 1817–1824, 2008.

[74] R. Abdulkadir, K. A. Imam, and M. B. Jibril, "Simulation of back propagation neural network for iris flower classification." *American Journal of Engineering Research*, vol. 6, pp. 200–205, 2017.

[75] Y. Unal, K. Polat, and E. H. Kocerc, "Pairwise FCM based feature weighting for improved classification of vertebral column disorders." *Computers in Biology and Medicine*, vol. 46, pp. 61–70, 2014.

[76] S. Ansari, F. Sajjad, Z. ul Qayyum, N. Naveed, and I. Shafi, "Diagnosis of vertebral column disorders using machine learning classifiers," Tech. Rep., Jul 2013.

[77] P. Jeatrakul and K. Wong, "Comparing the performance of different neural networks for binary classification problems." pp. 111–115, Institute of Electrical and Electronics Engineers, 2009.

[78] S. R. Konda, "A comparative evaluation of symbolic learning methods and neural learning methods." Department of Computer Science, University of Maryland.