Configuring Spiking Neural Network Training Algorithms

by

©Mst Mausumi Sabnam Mustari

A Dissertation submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of

Master of Science

Department of Scientific Computing

Memorial University of Newfoundland

November 2017

St. John's Newfoundland

Abstract

Spiking neural networks, based on biologically-plausible neurons with temporal information coding, are provably more powerful than widely used artificial neural networks based on sigmoid neurons (ANNs). However, training them is more challenging than training ANNs. Several methods have been proposed in the literature, each with its limitations: SpikeProp, NSEBP, ReSuMe, etc. And setting numerous parameters of spiking networks to obtain good accuracy has been largely ad hoc.

In this work, we used automated algorithm configuration tools to determine optimal combinations of parameters for ANNs, artificial neural networks with components simulating glia cells (astrocytes), and for spiking neural networks with SpikeProp learning algorithm. This allowed us to achieve better accuracy on standard datasets (Iris and Wisconsin Breast Cancer), and showed that even after optimization augmenting an artificial neural network with glia results in improved performance.

Guided by the experimental results, we have developed methods for determining values of several parameters of spiking neural networks, in particular weight and output ranges. These methods have been incorporated into a SpikeProp implementation.

Acknowledgements

I would first like to express my sincere gratitude and thanks to my thesis supervisor, Prof Dr. Antonina Kolokolova. To fulfill the research objective, it could not have been successfully conducted without her supervisions, and passionate participation. The door was always open whenever I ran into a trouble spot or had a question about my research or writing. I appreciate her expertise, constant guidance, valuable discussion, fruitful suggestions and kind encouragement throughout this study. Special thanks to my friends in the Department of Computer Science, particularly Ms. Zahra Sajedinia for effective discussions and for sharing her research work together. I am also grateful to my parents for their love, support they provided me through my entire life and venting of frustration during our graduate program. Also, I would like to express my special thanks to my brother Dr. Nur Alam for providing moral help and financial support throughout my study life.

Table of Contents

List of Tables

List of Figures

Acronyms

AI artificial intelligence. 1

ANGN artificial neuron glia network. 3, 13, 37, 38

ANN artificial neural network. 3, 37, 38, 40

AP action potential. 12

ASA accurate synaptic efficiency adjustment. 25, 27

CNS central nervous system. 13

EPSP excitory post synaptic potential. 14, 15

HHM hodgkin huxley model. 16

IF integrate-and-fire model. 16

ILS iterated local search. 32, 33

LIF leaky integrate-and-fire model. 16, 19, 20, 27

LTD long term depression. 25

LTP long term potentiation. 25

NSEBP normalized spiking error back propagation. ii, 24

PSP post synaptic potential. 14, 15

SNN spiking neural network. 4, 14, 26, 31, 36, 38, 40

SRM spike response model. 16, 20, 27

STDP spike timing dependent plasticity. 2, 7, 8, 25

WBC wisconsin breast cancer. 3, 4, 36, 39, 40, 43–46, 50, 51

Chapter 1

Introduction

Recent years have brought us numerous advances in artificial intelligence (AI) and machine learning. With AlphaGo beating the world's Go champion Lee Sedol in 2016, self-driving cars and trucks being tested on the roads, and a proliferation of voice-based virtual assistants, AI is becoming a household word. Much of this renaissance of AI seems to be due to algorithmic advances in training artificial neural networks, in particular deep neural networks. That started in earnest from the celebrated paper by Hinton, Osindero and Teh [HOT06], which presented a method to pre-train deep neural networks one layer at a time, and paved the way for widespread industrial use of deep neural networks.

However powerful deep neural networks seem to be, they are still second-generation neural networks, with each neuron computing a logistic/sigmoidal function. Though better than neurons computing a linear threshold function of its inputs in first-generation neural networks, sigmoidal neurons still lack the power of third-generation, most biologically plausible type of neurons: the spiking neurons. There, the output of a function computed by each neuron depends not only the value of its inputs, but also on the time each input arrived. Such temporal coding allows a spiking neuron to

compute provably more than its sigmoidal counterpart can [Maa97]. Moreover, spiking neurons are much more closely related to actual biological neurons, with similar models describing their behaviour.

However, the power of spiking neural networks comes at a price. Whereas it is well understood how to train a second-generation neural network, with the backpropagation algorithm being the gold standard, the question of training a spiking neural network is still wide open. In unsupervised learning scenario the commonly used approach is STDP: spike timing dependent plasticity spike-timing dependent plasticity, where the connection between two neurons strengthen whenever first (pre-synaptic) neuron emits a spike right before the second one, and weakens when the first neuron fires right after the second (and so its spike cannot contribute to the second neuron's output). This is a time-dependent extension of the Hebb's "neurons that fire together wire together" rule. Though appropriate for unsupervised learning setting (for example, for clustering data), this approach is not as suitable for supervised learning (a classification task where the network is first trained on labelled data). There is a number of proposed algorithms (see section 2.3.2 for more details), however most of them have significant disadvantages such as only being able to output one spike per neuron, or being not suitable for training multi-layer networks.

Even if we focus on a specific training algorithm such as SpikeProp [BKLP02], with its advantages and disadvantages, still another problem arises. Second-generation artificial neural networks have a relatively small and manageable number of parameters (number of layers and neurons at each layer, learning rate, number of training epochs, regularization whenever it is used). In addition to that, a spiking neural network has a number of extra parameters coming from the model describing the behaviour of the neurons: time constants, coding interval, threshold, time step, number and length of delays (when there are multiple delayed synapses between neurons), etc. There seems

to be no standard method for selecting even the most crucial set of parameters: target output times, the goal for the training...

1.1 Our results

In this thesis, we study the problem of configuring spiking network training algorithms, in particular SpikeProp. As a starting point, we used SpikeProp code for the XOR problem available on GitHub [mba], however that code required numerous changes. Once we had a working version of SpikeProp, we used automated parameter tuner, ParamILS [HHLBS09], to find a better set of parameters for two benchmark classification data sets, wisconsin breast cancer (WBC) and Iris. This allowed us to achieve better accuracy on these data sets than what is reported in the literature. We also achieve better accuracy for second-generation artificial neural networks, and artificial neural networks augmented with glia cells (astrocytes), improving upon results of [Saj14].

Then, we looked at the interplay between different parameters used in the SpikeProp algorithm (and in general in spiking neural networks with delays), and their effect on the classification accuracy. We obtain the following results.

Improved accuracy in ANNs vs. ANNs with glia Using automated algorithm configuration (ParamILS), we were able to improve the performance of artificial neural network (ANN) and artificial neural networks with glia (ANGN) presented in [Saj14] from 87% to 96% (ANN) and from 91% to 97% for WBC dataset, and from 80% to 90% (ANN), 86% to 96% (ANGN) for the Ionosphere data set (see table 4.1). In addition to better accuracy results, this adds more weight to Sajedinia's findings that adding glia (astrocytes) to an artificial neural networks improves performance. Indeed, even after optimization artificial neural networks with glia performed better

than plain ANNs. Thus, the improvement in accuracy with ANGNs over ANNs seems a genuine result of the change of the network structure.

Improved accuracy for SpikeProp-trained spiking neural network. We were able to improve upon accuracy of SpikeProp-trained SNN reported in the original paper introducing SpikeProp, [BKLP02]. In particular, for Iris dataset we were able to achieve the accuracy of 98.1% (versus 96.1% of [BKLP02]), and for WBC dataset our parameter optimization improved the accuracy from 97.0% to 98.5%.

Weight initialization bounds. We build upon the results of [MLB06] to give new formulas for initial weight range for each layer in the network. In contrast to [MLB06], our formulas do not involve target times.

Interplay between membrane time constant and delays. From our experimental results there seem to be a linear dependency between the membrane time constant τ and the number of synapses each with a different delay d corresponding to the region of best accuracy. In particular, for WBC dataset the best performance was achieved when the number of delays d was more than $1.6\tau + 1.9$, and for Iris data set the good performance started from $d \geq 0.96\tau - 0.21$. For both data sets, we considered values of τ starting from the length of the coding interval (that is, interval which contains the range of input data): it was shown in [Maa96] that this restriction is necessary for good performance. There is also a drop in performance when d becomes too large with respect to τ. For WBC dataset and a permutation of the original Iris data set, there was a well-defined line containing pairs (τ, d) with optimal accuracy (see figures 4.3, 4.9,4.7).

Target times and their ranges. For data sets such as WBC and Iris, it is the structure of the data that seems to influence selection of the target times the most. Even though the data is not linearly separable, even a simple heuristic of looking at the sum of the input values gives a pretty good estimation of the class for these data sets: see figures 4.10 and 4.11. Thus, it might be difficult to train a network to produce a spike for malignant class for the breast cancer data before a spike for benign class. For such data sets, the heuristic sometimes used in the literature [HLQ+16] of setting the target time to an average of a few runs for each class on a randomly initialized untrained network seems to give selections similar to what we obtain with ParamILS.

That said, we do need bounds on the output spike times, not only for the range of parameters to give to ParamILS, but also to determine when to stop running the network. For that, we develop formulas for the earliest and latest output spikes, based on the weight range bounds.

1.2 Thesis organization

In Section 2, we cover the basics of learning with artificial neural networks. Section 3 gives an overview of spiking neuron models, and a survey of algorithms for training spiking neural networks. Section 4 covers methods for parameter tuning and automated algorithm configuration with ParamILS. The core of the thesis is Section 5, where we present our results. Finally, Section 6 summarises the work we have done and discusses some ongoing and future work.

Chapter 2

Background

In this section, we cover definitions of various types of neural networks and learning settings, with the focus on artificial neural networks most commonly used in practice: second-generation neural networks.

2.1 Learning algorithms

There are three different settings for machine learning that appear in practice, in particular in the context of learning with artificial, including spiking, neural networks. In this work we focus on the first one, supervised learning (classification). Though quite natural in the context of artificial neural networks, supervised learning is harder for spiking neural networks; biologically inspired mechanisms tend to unsupervised or reinforcement models.

2.1.1 Supervised learning

Supervised learning is a technique for predicting a label of a previously unseen instance from the prior knowledge about input and the target output [Dav13]. It can be viewed

as a machine learning task of inferring a function from training data which can be used for correctly mapping a class for unseen instances. Each training sample (consisting of an input vector X, and its corresponding target output vector Y) may feed into network several times so that the actual output can approach the target output. An error value is calculated from each given sample as a function of the difference between the target outputs vector, Y and the actual output vector, Z (for example, min square error). In neural networks, this error is used to update connection weights in the network, so that network can generate a result closer to or exactly the desired output next time if similar input pattern arrives. Two most common approaches to minimise this error in spiking neural networks are gradient descent rule and learning windows rule [XQYK17]. In order to minimise errors, gradient descent based learning algorithm finds a local minimum of linear systems. The second type changes the synaptic weights as a function of the relative timing of pre- and postsynaptic action potentials.

2.1.2 Unsupervised learning

Unsupervised learning is a technique for exploring data to find some intrinsic structures in the input under an unknown probability distribution. The convergence analysis of unsupervised learning is much more complicated than other learning as the input datasets are unlabelled. For traditional artificial neural networks, a n-dimensional input is processed by the exact same number of computing units or by minimising a cost function for clustering, feature extraction, dimension reduction, and others. Self-organizing map (SOM), adaptive resonance theory (ART), principal component analysis (PCA), Independent Component Analysis (ICA), Hebbian learning and BCM rule are commonly used unsupervised learning algorithms for non-spiking neural networks.

In case of spiking neural networks, Hebbian learning, BCM rule, and STDP rules are being successfully used as unsupervised learning methods in real-world applications. STDP learning is an asymmetric form of Hebbian learning in a sense of tightening temporal correlations between weakening and strengthening the connection. Since unsupervised learning takes into account competition and lateral inhibition, the weights of the winner neurons (ones that emit the first spike) are increased while other neurons suffer a small weight reduction [IRMGM+15]. Similarly, STDP learning rule considers the lateral inhibition between pairs of spikes: a pre-post pairing causes potentiation and a post-pre pairing causes depression. The most recent presynaptic and postsynaptic spike pair is used to detect the correlations of next attempting fire.

2.1.3 Reinforcement Learning

Reinforcement learning is a control optimization technique which is used to recognize the best action in every state visited by the system. In reinforcement learning a general error signal back ("reward") is determined in every state that describes how well the system is performing. The typical framing of a reinforcement learning is the following scenario. An agent takes actions in an environment. Based on the action it changes state, and the learning algorithm also receives a reward signal a short time later. The current state and reward both are then fed back into the agent. The algorithm modifies its strategy in order to achieve the highest reward.

2.2 Artificial Neural Networks

Much of neural networks success story in recent years is due to adapting some powerful set of learning techniques in neural networks which is called deep learning. Intelligent adaptive control, decision support, complex systems identification, image compression,

pattern recognition, optimization, signal processing, speech recognition, face recognition and natural language processing is also driven by advances in neural networks and deep learning. Though a history of the neural network had made by William James in 1980 after a long time neurophysiologist Warren McCulloch and mathematician Walter Pitts first developed a simple neural network model in 1943 which was capable of computing any arithmetic or logical function [Yad15]. The founder of Neurocomputing Frank Rosenblatt invented the basic version of an artificial neuron called perceptron.

2.2.1 Backpropagation Algorithm

Feedforward neural networks do not contain any feedback connections to the previous or current layer. They can only pass information to next layer through neuron response function which is evaluated from input value X, and the intermediate connection weights, W and finally reach the output neuron. Backpropagation rule is applied to network from output layer to consecutive previous layer to adjust weights so that the network can learn mapping arbitrary unseen inputs to exact outputs. How backpropagation algorithm works to train feedforward neural network has been shown by an example (see figure 2.1). Here, we have shown feedforward network with one hidden layer H, two bias neurons (B_1, B_2) and one output neuron that needs to be trained up to get target (0.99) result. By this example we will explain working procedure of feedforward neural networks in four main steps: feedforward computation, backpropagation at the output layer, backpropagation at the hidden layer and weight updates. For more detailed explanation see, for example, this link: [Maz].

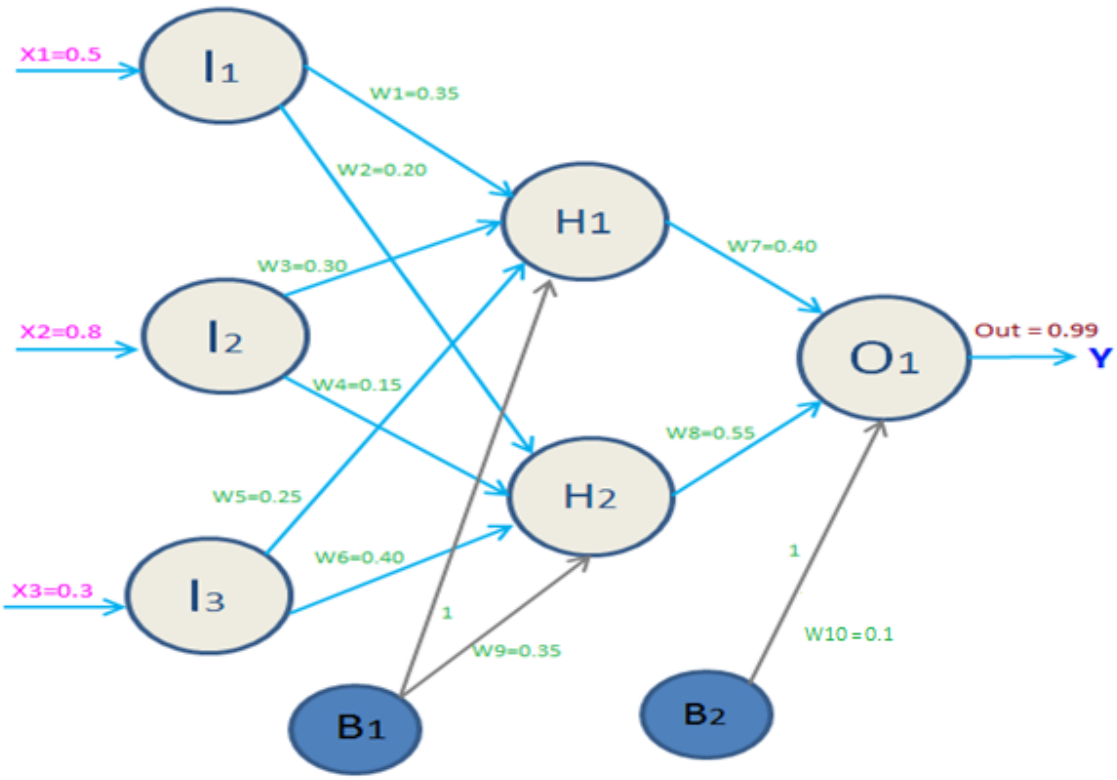

Figure 2.1: Incorporated with feedforward neural network

Feedforward Computation:

To know the output of each neuron first we need to determine the weighted sum of all incoming inputs of a neuron. Suppose the weighted sum of incoming input of a neuron is in_N and output is op_N.

Weighted sum of neuron H_1: $in_{H1} = w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + w_9 * B_1 = 0.35 * 0.5 + 0.3 * 0.8 + 0.25 * 0.3 + 0.35 * 1 = 0.84$

Output of neuron H_1: $op_{H1} = \frac{1}{1+e^{-in_{H1}}} = \frac{1}{1+e^{-0.84}} = 0.69846$

Similarly; $in_{H2} = 0.69$ and $op_{H2} = 0.66596$

$in_{O1} = 0.74566$ and $op_{O1} = 0.67823$

Network error is determined by the following function of the difference between actual and desired output:

$$E = \frac{1}{2}(desired - actual)^2 = \frac{1}{2}(d_{out} - op_{O1})^2 = \frac{1}{2}(0.99 - 0.67823)^2 = 0.04860$$

Backpropagation at the output layer:

A portion of the network error E is fed back to consecutive previous layer (see figure 2.2) to change the connection weights so that the actual output and the target output can get closer. For this reason the chain rule is applied to know how much each connection has been affected according to network error. The partial derivative of E with respect to w_7 describe the affected portion of that connection.

$$\frac{\delta E}{\delta w_7} = \frac{\delta E}{\delta op_{O1}} * \frac{\delta op_{O1}}{\delta in_{O1}} * \frac{\delta in_{O1}}{\delta w_7}$$

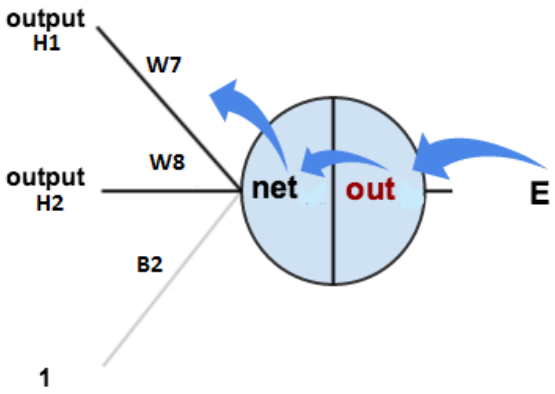

Figure 2.2: Illustration of backpropagation algorithm. Adapted from [Maz]

Step 1:

$$E = \frac{1}{2}(d_{out} - op_{O1})^2$$

$$\frac{\delta E}{\delta op_{O1}} = -(0.99 - 0.67823) = -0.31177$$

Step 2:

$$op_{O1} = \frac{1}{1+e^{-in_{O1}}} = (1 + e^{-in_{O1}})^{-1}$$

$$\frac{\delta op_{O1}}{\delta in_{O1}} = -1(1 + e^{-in_{O1}})^{-2} * \frac{\delta(1+e^{-in_{O1}})}{\delta in_{O1}} = op_{O1}(1 - op_{O1})$$

$$\frac{\delta op_{O1}}{\delta in_{O1}} = 0.67823 * (1 - 0.67823) = 0.21823$$

Step 3:

$in_{O1} = w_7 * op_{H1} + w_8 * op_{H2} + B_2 * 1$

$\frac{\delta in_{O1}}{\delta w_7} = op_{H1} = 0.69846$

Putting it all together:

$\frac{\delta E}{\delta w_7} = \frac{\delta E}{\delta op_{O1}} * \frac{\delta op_{O1}}{\delta in_{O1}} * \frac{\delta in_{O1}}{\delta w_7} = -(0.31177) * 0.21823 * 0.69846 = -0.04752$

After combining abvove three step, we can generalised the chain rule:

$\frac{\delta E}{\delta w_7} = \frac{\delta E}{\delta op_{O1}} * \frac{\delta op_{O1}}{\delta in_{O1}} * \frac{\delta in_{O1}}{\delta w_7} = -(d_{out} - op_{O1}) * op_{O1}(1 - op_{O1}) * in_{H1}$

Rewrite the equation by replacing $\delta_{O1} = (d_{out} - op_{O1}) * op_{O1}(1 - op_{O1})$

$\frac{\delta E}{\delta w_7} = -\delta_{O1} * in_{H1}$

In general, If we denote the backpropagated error at the j_{th} node by δ_j , and weight between node i and j by w_{ij}; then we can say $\Delta w_{ij} = -\eta * op_i * \delta_j$ (η = learning rate)

Backpropagation at the hidden layer:

$\frac{\delta E}{\delta w_1} = \frac{\delta E}{\delta out_{H1}} * \frac{\delta out_{H1}}{\delta net_{H1}} * \frac{\delta net_{H1}}{\delta w_1}$

The way has shown above is applied to consecutive previous layers with the partial derivative of the error E with respect to their weights. This process continues until the input layer is reached.

Overall, the above process is repeated for every sample until the network converges.

2.2.2 Augmenting ANNs with astrocytes

Recent advances have revealed that there is another type of cells called astrocyte (the most abundant type of macroglial cell) which is affecting signaling on synapses as well as impacting on neuronal information processing [PSA14]. Although astrocytes cannot propagate action potential (AP) like neurons do, they release neuroactive substances to communicate over short distances called “gliotransmitters”. According to Pereira and Furlan [PF10], astrocytes and neurons play a different vital role in the nervous system: astrocytes propagate a substance which is responsible for conveying "the feeling" and neurons carry information about "what happens" [Saj14, PF10].

In 1985, Dr. Marian Diamond published anatomical studies of slivers of Einstein's brain where she claimed that Einstein's brain had a greater ratio of glial cells to neurons compared to a sample group of 11 other brains [DSMH85]. Research has shown that the density of astrocytes can vary from region to region in the central nervous system (CNS) approximately 20 to 40 percent. Glia astrocytes can communicate in a bi-directional manner with neurons and other astrocytes by releasing transmitters and propagating calcium spikes. The term "tripartite synapse" refers to a concept of coupling (see figure 2.3) between astrocytes and neurons (pre and postsynaptic) which provides a pathway for chemical communication between the cells [WMH$^+$11].

A computational model of glia astrocyte named ANGN (artificial neuron glia network) introduces by Pazos et al([PGP09]) in 2009, where the connectionist systems was designed by adding an artificial glia astrocyte with each neuron [Saj13]. From the computational point of view, it is assumed that each astrocyte counts the number of firing and non-firing times of its associated neuron for a specific period of time. Depending on active and inactive times of the neuron, the connected weights will be changed by a pre-defined factor [Saj14, AGPPP12]. The effect of astrocytes in such network can be defined by the following parameters:

k: number of iterations (number of times each sample is fed into the network)

a : weight increment rate

b : weight decrement rate

c : threshold on the number of times j^{th} neuron has fired or not during k iterations.

If a neuron remains active c times, the weights of the connections will be increased by the factor of a, while connection weight will be decreased by the factor of b if the neuron remained inactive c times during k iterations.

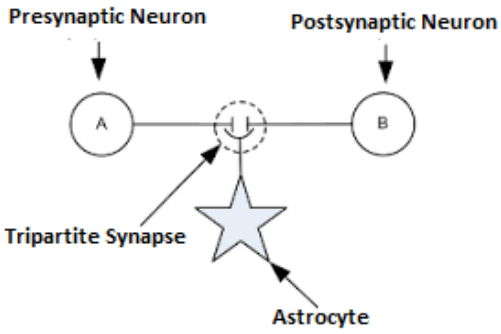

Figure 2.3: Network consists of presynaptic neuron A, postsynaptic neuron B and an interconnecting tripartite synapse. Adapted from figure 3 in [WMH+11].

2.3 Spiking Neural Networks

The network model which is comprised of spiking neurons is called spiking neural network (SNN). Designing a spiking neuron model starts with mimicking the characteristics of a biological neuron. Wolfgang Maass described a mathematical formation of a formal spiking neural network in ([Maa95, Maa96]) [Maa97].

Each spiking neuron operates by integrating the voltage sum of all presynaptic spike in a time-dependent manner. It emits an output spike (see figure 2.4 A) when the excitory post synaptic potential (EPSP), also called the membrane potential, reaches a threshold value. Thus, in a spiking neural network each neuron conveys information to the next layer by individual spike times. When a single presynaptic spike arrives from presynaptic neuron i to a postsynaptic neuron j at time t_i, it generates a post synaptic potential (PSP) which reflect membrane potential (see figure 2.4B). The membrane potential at current time t is defined by $V(t)$:

$$V(t) = \sum_{i \in \Gamma_j} w_{ij} \varepsilon_j (t - t_i) \tag{2.1}$$

Here, the kernel function ε describes the post-synaptic potential (PSP) contribution by each incoming spike (see figure 2.4C), Γ_j is a set containing the spike times t_j emitted by all the presynaptic neurons of the neuron i, and w_{ij} are the corresponding weights. Lets $(t - t_i) = s$; if $s <= 0$, then $\varepsilon(s) = 0$. Otherwise,

$$\varepsilon(s) = \frac{s}{\tau} e^{1-s/\tau} \tag{2.2}$$

where the membrane time constant τ determines the rise and decay time of the PSP.

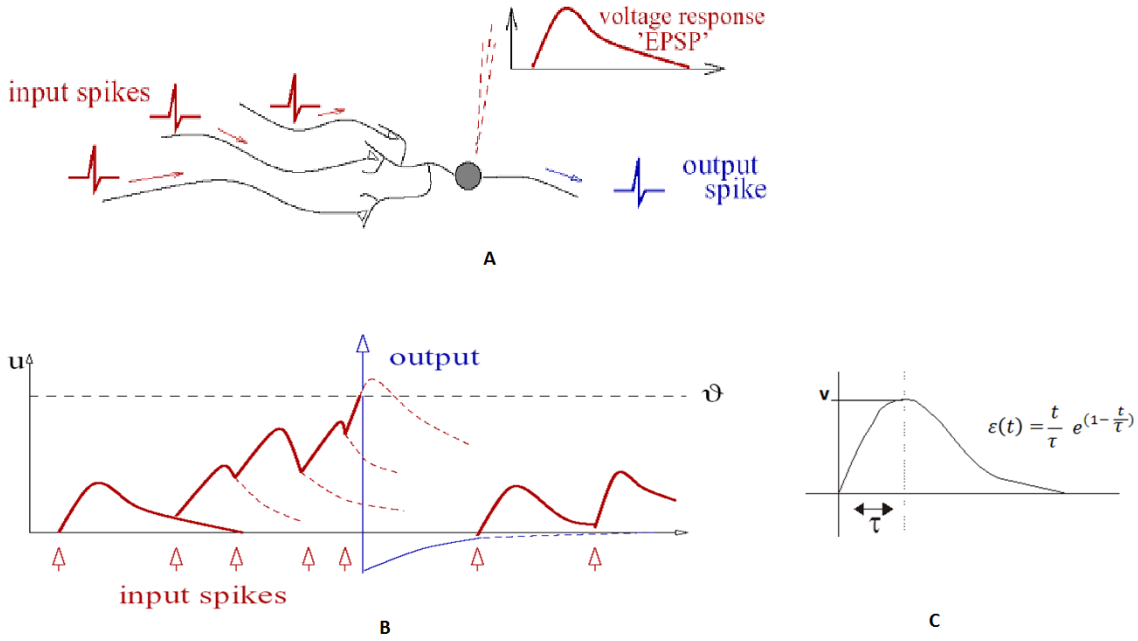

Figure 2.4: (A)Each input pulse causes an excitatory postsynaptic potential (EPSP). (B)All EPSPs are added one after one. When it reaches the threshold voltage v, it generates an output spike. (C) The value of ϵ(t) is maximum at membrane time constant τ. Taken from [Boh].

A spiking neuron (see figure 2.5B) computes membrane potential at each point of time as a function of input spikes, while a sigmoidal neuron (see figure 2.5A) computes output voltage after receiving all input from the previous layer. Spiking neurons work with precise timing of spike, biologically more plausible, considerably

faster and computationally more powerful [Maa97]. The artificial sigmoid neuron replaces a sharp threshold with a smoother function with sigmoid or logistic and produces real-valued output bounded by (0,1).

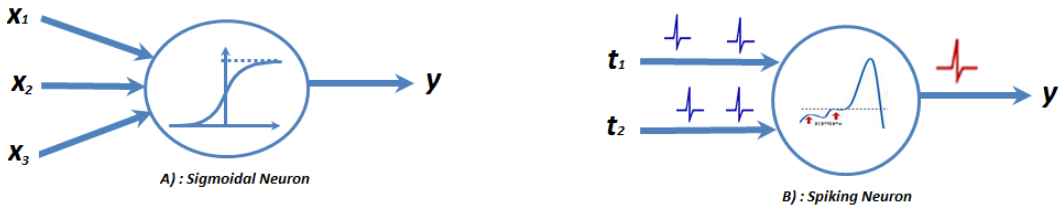

Figure 2.5: Response of neuron after receiving input. (A) The output pattern of a sigmoidal neuron. (B) The membrane potential of a spiking neuron.

Spiking Neuron Models

Over time, multiple models of a biological neuron have been proposed, of varying complexity. Here, we survey common computational models of a biological neuron, inlcuding hodgkin huxley model (HHM), integrate-and-fire model (IF), leaky integrate-and-fire model (LIF), and spike response model (SRM).

2.3.1 Biological Neuron

A neuron is an electrically excitable cell that receives, processes, and transmits information to other cells in the body through electrical and chemical signals. Biological neuron (see figure 2.6) mainly consist of three major parts: dendrites by through information comes into the neuron from other neurons, cell body (soma), the main part of the neuron, which processes information and then passes it along to the third main part, an axon. The operation of nervous system depends on how well neurons communicate with each other. For an electrical signal to travel between two neurons, it usually must first be converted to the chemical signal. Then it crosses a space about

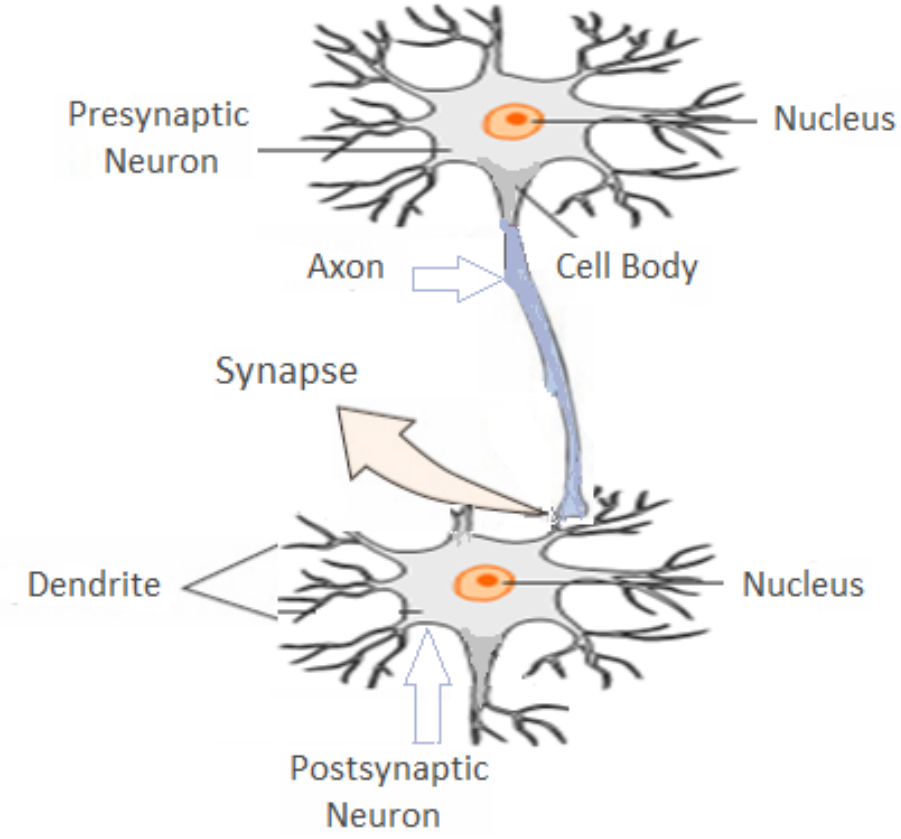

Figure 2.6: Demonstration of main part of biological neuron

one-millionth of an inch. This space is called synapse and the chemical signal is called neurotransmitter. Neurotransmitters allows billions of neurons in the nervous system to communicate with each other. Though number of encoding strategies such as binary coding, rate coding, latency coding, fully temporal codes, population coding, predictive spike-coding, probabilistic spike-coding, etc are being used to understand the responses to stimuli, the two most popular types of encoding methods are the frequency of spiking (rate coding) and the firing times of spikes (pulse or temporal

coding).

In the algorithms we consider, information is encoded in the precise spike firing time [XZHY13]. If the ranges of input values are very different, then each feature value is often mapped to a high-dimensional space by defining some function such as Gaussian RFs or square cosine encoder. In our experiments, we do not use such encodings, instead setting spike times directly to the (normalized) input values.

Integrate-and-Fire model

The neuron model where the action potentials (very rapid change in membrane potential when a cell membrane is stimulated) are described as events is called the integrate-and-fire model. This model is represented by a linear differential equation, which is the time derivative of the law of capacitance, $Q = CV$.

$$I(t) = C\frac{dv}{dt} \tag{2.3}$$

When an input current $I(t)$ is applied, the membrane capacitor is charged with time until it reaches a constant threshold voltage V_{th}, at which point a sharp electrical pulse called spike is triggered and the voltage is reset to its resting potential. The firing frequency of the model increases linearly without bound as input current increases. By introducing a refractory period t_{ref}, we can limit firing frequency of a neuron by preventing it from firing during that period. The limitation of integrate-and-fire model is that if it receives a below threshold signal at some time, it will retain that voltage boost forever until it fires again.

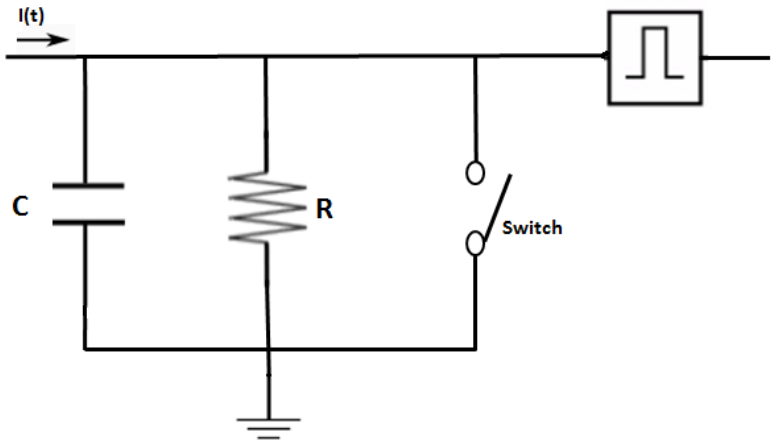

Figure 2.7: Integrate-and-Fire model

Leaky Integrate-and-Fire model (LIF)

The leaky integrate-and-fire model neuron is a commonly used spiking neuron model, owing to its relative simplicity and analytical tractability. The limitation of the integrate-and-fire model is solved in Leaky Integrate-and-Fire model by adding a term "leaky integrator" to the membrane potential. The basic circuit of an integrate-and-fire model (see Fig.2.7) consists of a capacitor C in parallel with a resistor R driven by a current $I(t)$. The driving current can be split into two components.

$$I(t) = I_R + I_C \tag{2.4}$$

$$I(t) = \frac{v(t)}{R} + C\frac{dv}{dt} \tag{2.5}$$

Equation 2.5 derives from equation 2.4: multiply equation 2.4 by R and introduce the time constant $\tau_m = RC$ of the 'leaky integrator'. This yields the standard form of Leaky Integrate-and-Fire model . We refer to $v(t)$ as the membrane potential at time t and to τ_m as the membrane time constant of the neuron.

$$\tau_m \mathrm{d}v\frac{\mathrm{d}}{t} = -v(t) + RI(t). \tag{2.6}$$

In a simulation, spiking events of LIF model are used in a slightly different way. The membrane potential can be described explicitly by the constant input current, time-varying input current and synaptic currents. In a more realistic case, spikes are characterized by a firing time where the neuron is stimulated by pre-synaptic spikes arriving at its synapses [GK02b]. Post-synaptic current to the i_{th} neuron at time t is:

$$I(t) = \sum_i w_{ij} \sum_f \varepsilon(t - t_j^f) \tag{2.7}$$

Spike Response Model (SRM)

The spike response model (SRM) is a generalised version of the leaky integrate-and-fire model. It includes a new term "refractoriness", which describes the temporary inability of generating new spikes immediately after firing.

The SRM produces very good predictions of the target spike trains over a broad range of means and standard deviations of the injected current [JTG03]. Suppose that the neuron has fired its last spike at time t. At each time $t > \hat{t}$, the state of this spiking neuron at time t is represented by $u(t)$:

$$u(t) = \eta(t - \hat{t}) + \int_{-\infty}^{+\infty} k(t - \hat{t}, s) I(t - s) ds \tag{2.8}$$

Here, kernel k is the linear response to an input pulse. The form of the action potential and the after-potential is described by a function η. Function $\eta(t - \hat{t}_i)$ defines the refactory period of a neuron and the last term accounts for the effect of an external current $I(t)$.

SRM_0 models

SRM_0 models is a simplified version of the Spike Response Model. In SRM_0 models the state of spiking neuron at time t is represented by $u(t)$:

$$u_i(t) = \eta(t - \hat{t}_i) + \sum_j W_{ij} \sum_{t_j} \varepsilon_0(t - t_j) + \int_0^\infty k_0(s) I^{ext}(t - s) ds \tag{2.9}$$

$$\eta(s) = \delta(s) - \eta_0 \exp(-\frac{s}{\tau_m}) \tag{2.10}$$

$$\varepsilon_0(s) = \frac{1}{1 - \frac{\tau_c}{\tau_m}} (\exp(-\frac{s}{\tau_m}) - \exp(-\frac{s}{\tau_c})) \tag{2.11}$$

Where,

s is the time after the arriving spike

τ_c is the current time constant.

τ_m is the membrane time constant.

Hodgkin–Huxley model

Hodgkin-Huxley model (see figure 2.8) describes the current at the neuron in terms of of three types of channels: a sodium channel with index Na, a potassium channel with index K and an leakage channel with index L. This is the most biologically realistic model, but it is computationally more difficult to simulate.

$$I = C_m \frac{dv_m}{dt} + I_{ion} \tag{2.12}$$

where C_m is the membrane capacitance per unit, V_m is the intracellular membrane potential, I_{ion} is the sum of all ion channels current, and I is the externally applied

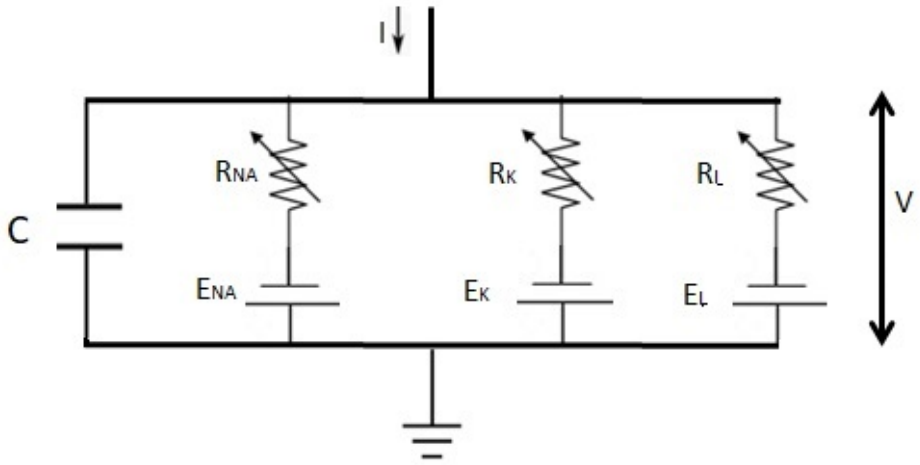

Figure 2.8: Hodgkin–Huxley model

current. The most significant advantage of the above equations is to determine the membrane capacitance in such a way that is independent of the sign or magnitude of the intracellular membrane potential and minimally affected by the time course of V_m.

Here,

$$I_{ion} = g_{NA}(V_m - V_{NA}) + g_k(V_m - V_k) + g_L(V_m - V_L) \tag{2.13}$$

where

g_{NA} = sodium conductances per unit area

g_K = potassium conductances per unit area

g_L = leak conductance conductances per unit area

V_{NA} = sodium reversal potentials

V_K = potassium reversal potentials

g_L = leak reversal potential

Hodgkin-Huxley model is the most biologically realistic model, but it is computationally difficult to simulate.

Izhikevich model

Izhikevich [Izh03] attempts to create a more computationally efficient version of Hodgkin-Huxley model by using a system of differential equations with a quadratic function of the membrane potential v. He uses an auxiliary variable u to represent the negative feedback from activation of K^+ and inactivation of Na^+ channels. The derivatives are taken with respect to time, and a, b, c, d are parameters. The variable I denotes the external current.

$$v' = 0.04v^2 + 5v + 140 - u + I$$

$$u' = a(bv - u)$$

$$\text{if } v \geq 30mV, \text{ then } v = c, \text{ and } u = u + d$$

This neuronal model is capable of reproducing a variety of spiking patterns, including RS (regular spiking), IB (intrinsically bursting), and mixed mode firing patters, FS (fast spiking), TC (thalamo-cortical), RZ (resonator) and LTS (low-threshold spiking) [Izh03].

2.3.2 Supervised learning with spiking neural networks

Several learning algorithms have been proposed to train spiking neurons. In terms on a number of spikes supervised learning algorithm can be subdivided into two categories: single spike and multiple spikes.

Survey of supervised learning algorithms for SNNs

S.M Bohte et al. [BKLP02] proposed a gradient descent-based supervised learning algorithm called SpikeProp, that transfers information in the timing of single

spike [BKLP02]. Sam McKennoch et al. (2006) made some modifications on SpikeProp by adding momentum and has shown faster convergence on his proposed modified algorithm QuickProp and RProp (Resilient Propagation) [MLB06]. But still, these error backpropagation algorithms were limited in that they did not allow multiple spikes, which made them less realistic to perform a more complex computation. Weight limit learning algorithm is another simplified version of SpikeProp algorithm has been proposed by Q.X.Wu et al. (2006) [WMM+06]. Inspired by biological neurons, a weight limit constraint is applied on SpikeProp to ensure that all neurons fire at least once in a simulation period. The training accuracy depends on the proper adjustment of synaptic weights of the network, such that the actual outputs are close to target outputs. Since the network error is calculated as a function of the time difference between actual and target firing of a neuron, special learning rules need to be added if a neuron does not fire during the simulation period.

Later, Y. Xu et al. [XZHY13] introduced another gradient descent based supervised multi-spike learning algorithm, where error function has been constructed considering the same number of output spikes dynamically as of target spikes in each learning epoch. Tempotron is a gradient-descent-based single layer supervised learning algorithm introduced by Gutig and Sompolinsky. It only works for binary classification, though it does allow multiple spikes [GS06].

An error back propagation multi-spike supervised learning named normalized spiking error back propagation (NSEBP) [HLQ+16] is a multilayer training algorithm. Unlike SpikeProp, QuickProp, and RProp, there are no delays in NSEBP network. The computational error is propagated back to previous layers by presynaptic spike jitter instead of the traditional gradient-descent rule. In the feedforward calculation, the time boundaries are determined to avoid the action potential going infinitesimal. Presynaptic spikes between this time boundaries are selected for hidden layer neuron.

Random spikes are added in a hidden layer if there is no spike generated between this range. For improving training efficiency only the output voltage is determined at target time and ignore the state of other times. This learning approach adopted with STDP has been described by same authors in accurate synaptic efficiency adjustment (ASA) supervised training method [XQYK17].

Ponulak proposed a remote supervised method (ReSuMe) for a spiking neuron [Pon05], and it has been revised to train SNNs afterward [KP05, Pon06, KPK06, PBR08] [PA10]. The fact behind the naming "remote supervised method (ReSuMe)" is that the target signal does not directly influence the membrane potential of the corresponding learning neuron [XZZ13]. Instead of gradient descent method, ReSuMe is motivated by Widrow-Hoff rule and it minimizes the error based on the interaction between two spike-timing-dependent plasticity processes instead of gradient calculation. An anti-STDP process is applied for weakening synapses while the input spike trains are co-related with the actual spike trains and STDP process for strengthening synapses while the input spike trains are co-related with the target spike trains respectively [XZHY13].

A new concept for training spiking neural network based on plasticity window, called synaptic weight association training (SWAT), was published in 2010 by John J. Wade, Liam J. McDaid, Jose A. Santos, and Heather M. Sayers [WMSS10]. There, STDP is combined with Bienenstock–Cooper–Munro (BCM) theory, where a single neuron is used as a training neuron and data associated with all class is passed to this neuron. A sliding threshold associated with BCM model controls long term depression (LTD) over long term potentiation (LTP) of STDP so that the action potential of the neuron cannot go to infinitesimal.

Spike-based processes, such as Long-term potentiation (LTP), long-term depression (LTD) and spike-timing dependent plasticity (STDP), are being widely used

as Unsupervised learning for spiking neural network. It has been investigated and explored in the literature ([BSA89, GK02a, GKvHW96, MLFS97, KvRST02, Kis02]). [KP06]

The summarized view of supervised learning algorithms with spike time coding of SNN is shown below:

Algorithm	Model	Encodings	Summary	References
SpikeProp	SRM; multi-layer; with delay	Gaussian RF; single spike	No random noise, back propagation,gradient descent	[BKLP02]
Learning under weight constraints	SRM; multi-layer; with delay	Square cosine RF;single spike	weight limit constraints are applied to SpikeProp algorithm.	[WMM+06]
NSEBP	SRM_0; multi-layer; no delay	Gaussian RF; multiple spikes	Weight from i/p to hidden are only adjusted and o/p neuron has weight 1; use presynaptic spike jitter instead of gradient descent.	[HLQ+16]
ASA	SRM_0; multi-layer; no delay	Gaussian RF; multiple spikes	Learning is completed when voltage of output neuron is equal to threshold; Weight from i/p to hidden are only adjusted using normalized STDP rule	[XQYK17]
ReSuMe	Izhikevich; single layer; with delay	Single spike	Learning proceeds by correlation of spike times instead of gradient descent;uses reference signal; learning depend on network size and learning window.	[Pon05] [PA10]
SWAT	LIF; multi layer; no delay	Single spike	Use a single training neuron in training phase; use combined STDP/BCM training rule.	[WMSS10]
Tempotron	LIF; single layer; no delay	Single spike	Gaussian noise added to all spike time; only work for binary classification.	[GS06]

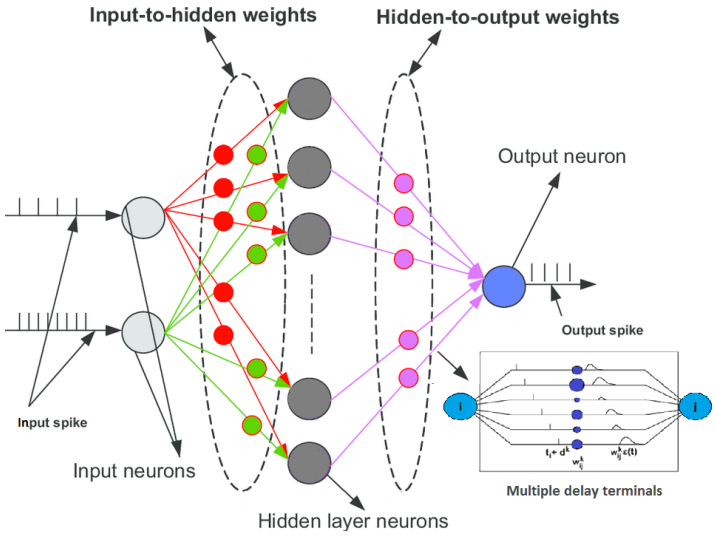

Figure 2.9: Spiking neural network with delay path

2.3.3 SpikeProp

In this section, we will demonstrate the procedure of training a multilayer spiking neural network with a error back propagation based learning algorithm which is capable of learning complex nonlinear tasks. Connectionist system of our feedforward spiking neural network with multiple delayed synaptic terminal has been shown in figure 2.9

- Delayed sub-connections:

  Assume that there are k number of delay path way associated with each synaptic connection between neuron i to j. Let t_i be the firing time of a pre-synaptic neuron i, and d_k is the delay associated with the synaptic terminal k, then the membrane potential is written as follows:

$$x_j(t) = \sum_{i \in \Gamma_j} \sum_{k} w_{ij}^k \varepsilon_j(t - t_i^k - d_k) \tag{2.14}$$

- Back propagation of the network error:

When the membrane potential reach to threshold voltage, a output spike is emitted from neuron j which goes to next layer (output neuron). The timing of emitted spike is called actual firing time. An error E is calculated by taking difference between target firing time t_j^{tg} and actual firing time t_j^{ac}

$$E = \frac{1}{2}\sum_j (t_j^{ac} - t_j^{tg})^2 \tag{2.15}$$

The error is then back propagated to network and according to error of subsequent layer the synaptic weights of associated layer are updated.

- Weight modifications for the output neurons:

  we need to calculate

$$\Delta w_{ij}^k = -\eta \frac{\partial E}{\partial w_{ij}^k} \tag{2.16}$$

$$\Delta w_{ij}^k = -\eta y_i^k(tj)\delta_j \tag{2.17}$$

- Weight modifications for the hidden neurons

$$\Delta w_{ij}^k = -\eta y_h^k(t_i)\delta_i \tag{2.18}$$

Chapter 3

Parameter Tuning

Setting the parameters is one of the most challenging tasks in designing effective algorithms. To optimize the performance of a target algorithms, we look for expert experience, rules of thumb, or sometimes brute-force search for parameter configuration. Unfortunately, this manual parameter tuning approaches is often tedious and unsatisfactory.

3.1 Manually Tuning

In the famous Coursera ML course by Andrew Ng [Ng], it is suggested to look at the "learning curves", and compare the behaviour of the training error and the cross-validation error. This can indicate whether low accuracy is due to underfitting or overfitting.

There are many possible options that could improve the performance of the learning algorithm such as getting more training examples, smaller set of features, getting an additional feature, adding polynomial features, decreasing learning rate, and increasing learning rate. If the training set becomes larger and larger, the average training error will increase simultaneously while cross-validation error and testing er-

ror will decrease as most of the data fit to train set. Causes of poor performance of machine learning algorithms could be either underfitting or overfitting problem.

High Bias (underfitting): High bias problem occurs when an algorithm made the wrong assumption between features and target outputs. A model falls into underfitting if the algorithm is having a low variance but high bias problem. When a model experiences both high cross-validation error and high training error, it is a sign of the high bias problem. The techniques such as decreasing regularization parameter λ, adding features and polynomial features are the powerful steps, but getting more training data will not help much when it is suffering from the high bias problem.

High Variance (overfitting): In reality, we would want to choose a model with low bias and low variance. When a model experience low training error but high cross-validation is referred to as high variance problem. Cross-validation error is always much bigger than the training error for high variance problem. If a learning algorithm is suffering from high variance problem, then getting more training data, smaller sets of features, and increasing the regularization parameter are likely to help.

It is harder to apply this reasoning to SNN, as there are too many parameters that might influence the outcome.

3.2 Automated parameter configuration

Automated parameter configuration deals with an algorithm whose performance is to be optimised under a given domain of parameters. Depending on the behaviour of parameters, it determines the nature of the configuration space so that the target algorithm can perform well in the new domain other than in given instance set. Some of the standard techniques for automated algorithm configuration are the exhaustive

search, hill climbing search, and genetic algorithms etc. Grid search and random are the most commonly used algorithm configuration strategy.

Exhaustive Grid Search It exhaustively generates candidates model by choosing their parameter values from a grid and evaluate the model for each combination of algorithm parameters using cross-validation.

Random Search Since exhaustive grid searches expensive, there is an alternative method called random search comparatively more powerful than grid search in high-dimensional spaces [BB12]. It selects a random combination of hyperparameter values from specifying range, without repetition, and evaluate the models sequentially.

3.3 ParamILS

Automated algorithm configuration tools ParamILS, which is based on the idea of iterated local search (ILS) in parameter configuration, was developed for optimizing SAT and MILP solvers by Hutter et al [HHS07]. It is a state-of-the-art method for parameter tuning and algorithm configuration which is actively being used to improve the variety of application including artificial intelligence. It can improve parameter configuration of a large and complex model by avoiding an unnecessary run of the algorithm. With a given ranges of parameters, ParamILS capable of optimising algorithm in term of minimum error, number of successes and minimizing computational resources such as runtime, memory, or communication bandwidth etc. To configure an algorithm with ParamILS, it will have some characteristics such as parameterized algorithm, a domain of parameters, a set of problem instances, and an objective function. ParamILS perform multiple runs with a different combination of the parameter from parameter configuration space that yields the best outcome across the bench-

mark dataset. For deterministic algorithms configuration, each run varies with input instances but the seed of algorithm is fixed. This feasible combination of parameter values depends on some conditional parameters such as:

cutoff_time define a specific time after which the candidate algorithm will be terminated.

cutoff_length define a specific run length after which algorithm will be terminated.

tunerTimeout refer validation of the final best found parameter configuration.

Automated algorithm configuration tool make a decision by addressing the following choice with the lowest cost.

1. Cutoff time for each run?
2. Number of runs for each instance?
3. Which problem instances have to consider for evaluating each parameter configurations?
4. Which parameter configurations should be evaluated?

A comparison is made between previous optimised algorithm which is reffed to as target algorithm and candidate algorithm whose performance has to be optimized. Adaptive capping approaches has been used in ParamILS for selecting the number of problem instances and to determine the cutoff time while ILS use for searching parameter configuration space.

ILS method

iterated local search (ILS) generates a sequence of local optima by employing repeated random trials to get better one. The way of searching the local neighbourhood and acceptance criterion to decide whether to keep or reject a newly obtained candidate solution is as follows [HZHS13].

1. Initial solution: generate an initial solution.

2. Local search: get another solution perturbation to escape from local optima.
3. Perturbation: random move in higher order neighbourhood.
4. Acceptance criterion: compare and choose a better solution with minimum cost.

This iteration process is repeated until met the termination condition. ParamILS follows the above procedure to find substantially improved parameter configurations. It generates initial solution with a default and random initialization. A subsidiary local search procedure is employed at random with changing only one parameter at a time. After accepting a parameter configuration, it reinitializes the parameter setting with a given probability.

Adaptive capping method

ParamILS evalutes parameter configuration pairwise with the same number of run on same instances and seeds. Without adaptive capping, this evaluation process takes a long period of time. It is capable of avoiding an unnecessary run of the algorithm. Let evaluate a pairwise (Θ_1 and Θ_2) parameter configuration process on same number of instances ($N = 100$) to see how it avoid unnecessary run. Suppose Θ_1 needs $10ms$ and Θ_2 needs $50ms$ to process 100 instances. Processing time per instance for Θ_1 is $10/100 = 0.1ms$ and for Θ_2 is $0/100 = 0.5ms$. ParamILS compute the lower bound of cost function after each run. As the runtime of Θ_1 is less than Θ_2, it will reject Θ_2 after $10ms$. ParamILS compare parameter configuration with a bounded evaluation period and evaluate all parameter configurations and select the best one with probability arbitrarily close to one. When the lower bound over the bounded time, it skips the rest run.

Chapter 4

Configuring Neural Networks

The main focus of this work is on configuring neural network training algorithms, in particular SpikeProp. First, we use automated parameter configuration (ParamILS) described in the previous chapter to see how much an accuracy can be improved by optimizing the parameters of learning algorithms. Then, we look at the interplay between different parameter settings for spiking neural networks, in particular initial weight range, as well as membrane time potential τ and number of delays d, and analyse the datasets to see why the common heuristic for setting target times seems to give good results.

4.1 General Methodology

We used ParamILS to configure two supervised learning algorithms, backpropagation (with and without glia) and SpikeProp. In both cases, we used neural networks with one hidden layer. The optimization was done with respect to the percentage of correctly classified instances in test data. We supplied a range for each parameter, in particular for the number of hidden layer nodes, learning rate, number of epochs and a variety of spiking neural network parameters.

4.1.1 Datasets

We used the following publicly available datasets, for which previous accuracy data for SNN classification was available.

Wisconsin breast cancer dataset: Wisconsin breast cancer dataset (WBC) contains 699 samples, divided into benign and malignant class with 16 cases of missing value. Remaining 683 samples contain 444 benign and 239 malignant type data.4 Each sample contains 11 attributes including an identification number, input attributes, and class. First attribute considered as an identification number, middle 9 attributes as an input variable and last attribute as a class.

Iris dataset: Iris dataset contains 150 samples having 3 classes and each sample has 4 attributes.

Ionosphere dataset: The 351 instances of ionosphere dataset were divided into 175 training and 176 testing instances. The input layer consisted of 34 neurons, each neuron corresponded to one attribute and the output layer had two neurons, representing "good" and "bad"

4.1.2 Cross Validation

A model performance is determined by how well the target function from training data of that model generalizes to new data. Though a learning algorithm fits on training set well, it might fail to generalize new data. Cross-validation actually can help to pick parameter that is going to be best. It is the most commonly used parameter tuning method which also helps to avoid a lot of testing. Generally, a randomly sorted dataset is divided into the training set, validation set and testing set.

In our experiments, we generated 10 permutations of each dataset. We used 5 of them as ParamILS training datasets (that is, for model parameter optimization), and 5 for model parameter testing (i.e., as ParamILS's testing data). For each individual experiment, the dataset was split in half, with the first half used for training, and the second half for testing. In this way, the actual training was done on different set of samples for each of the 10 permutations of datasets, and testing done on the remaining samples.

4.2 Optimizing ANNs and ANGN

The network model ANGN inspired by the work done by Zahra Sajedinia [Saj14] is an extension of ANN where a new processing element called glia astrocytes was added to the network. It was tested with two benchmark dataset (Ionosphere and Wisconsin breast cancer). Backpropagation learning algorithm was used for training both ANN and ANGN network. We have worked on both ANNs and ANGNs network and configured a number of parameters that are responsible for varying accuracy. Comparison table (see Table: 6.1) demonstrate that configured network outperform typical network.

ANNs

Configured parameters value for ANNs such as Learning rate α, number of neuron at hidden layer m, number of epoch e and others non-configurable parameters such as number of input neuron n and number of output neuron o has been shown in the table: 6.2.

Dataset	ANN ([Saj14])	Configured ANN	ANGN ([Saj14])	Configured ANGN
WBC	87%	96%	91%	97%
Ionosphere	80%	90%	86%	96%

Table 4.1: Comparison result between ANNs and ANGNs

Dataset	n	m	o	α	e	accuracy
WBC	9	17	2	0.03	100	96%
Ionosphere	34	18	2	0.35	700	90%

Table 4.2: ANNs simulation parameters

ANGNs

List of configured parameters of ANGN such as learning rate α, number of neurons at hidden layer m, number of cycles to activate glia k, Weight increments rate a, weight decrements rate b, and glia threshold θ has been shown in table 6.3.

4.3 Optimizing SNN with SpikeProp

In this section, we present results of our experiments with ParamILS-optimized SpikeProp.

Dataset	n	m	o	α	e	a	b	k	θ	accuracy
WBC	9	10	2	0.45	100	1.25	0.55	80	0.95	97%
Ionosphere	34	10	2	0.1	1000	0.7	0.75	0.35	0.65	96%

Table 4.3: ANGNs simulation parameters

4.3.1 SpikeProp implementation

In our experiment, we use discrete time slots like Bohte et al [BKLP02]. As datasets we used to have comparable ranges of inputs, we did not encode inputs with the Gaussian receptive field, but fed direct input to the network; that made for a smaller network. We used 1 output neuron for all experiments, differentiating between classes by target times. Coding interval is defined by the range of data. All experiments were done on computer with configuration:Intel (R) Xeon (R) CPU X5550 @ 2.67GHz, x86_64, 32-bit, 64-bit CPU op_mode (s), 4 core (s) per socket, 16 CPU (s), 6 CPU family and 26 model.

To select target times, we compute potential earliest and latest output spike times, then a different time slot is assigned for each class in between earliest and latest spike time using ParamILS. Spiking Neural Network model builds upon code for XOR from GitHub [mba]. *

WBC Network: consist of an input layer, 1 hidden layer and output layer with 1 output neuron. Gaussian receptive field or square cosine encoder is not necessary for breast cancer data as it has the same value range from 1 to 10. There are 9 input variables directly encoded to network and 2 different time slots are assigned for output spike times corresponded to benign and malignant class.

Iris Network: consists of an input layer with 4-encoding neurons, 1 hidden layer, and an output layer with the 1-output neuron. Input variables are directly encoded to network same as WBC network and three different time slots are assigned for output spike times corresponded to setosa, versicolor, and virginica class.

A comparison of two benchmark dataset with the result reported by Bothe et

*We had to make a number of changes to make this code work, in particular setting initial weight and target time. Latest version of our work is available to github [Mus]

Dataset	n	m	o	α	d	e	τ
WBC	9	9	1	0.01	16	800	10
Iris	4	12	1	0.05	14	100	9

Table 4.4: SNN simulation parameters

	Testing (Iris)	Testing (WBC)
SpikeProp ([BKLP02])	96.1% ± 0.1	97.0% ± 0.6
Configured SpikeProp	98.1% ± 0.2	98.5%± 0.8

Table 4.5: Comparison between original SpikeProp paper and SpikeProp we configured using ParamILS

al. [BKLP02] has been shown in table 4.5 and simulation setup for our experiment has been reported in table 4.4. All result are based on average 100 independent runs with a cutoff time 5.0.

Evaluating both outcomes as shown in table 4.5, it can be seen that our experiment outperform original SpikeProp in term of accuracy and smaller network architecture.

4.4 Investigating parameters in SNN configuration

In this section we will closely observe how output spike time related to other parameters such as a number of hidden neurons n, a number of delay terminals d_k, threshold voltage v, synaptic weights w_{ijk}, coding interval ΔT and membrane time constant τ. For example, it was shown that a τ needs to be larger than the relevant ΔT [BKLP02], but it is now clear by how much it needs to be larger for better accuracy. As SNNs have more configurable parameter (such as membrane time constant τ, threshold θ, target spike time t_j^{tg}, number of synapses d_k, increment rate of delay, weight range, and coding interval ΔT) than ANN, training SNN is more complicated than ANN.

4.4.1 Weight bounds

Setting initial weight of network have a significant impact to enable the network to converge rapidly. If a neuron does not fire at all, this neuron will have no contribution to entire training process [MLB06]. If the weights are too small, neurons might never fire. On the other hand, if the weights are too large neurons will fire earlier than expected and the network will not consistently converge. That's why it is important to set initial weight in such a way so that each neuron fire at least once. An approach to set weight range was proposed in [MLB06] However, it requires knowing target times. Instead, we set weights based on number of previous layer neurons n_i, maximum number of delay path d, $t_{max} = \tau + d$, and the time step t_{min} depend on data set. For Iris dataset $t_{min} = 0.1$ and for Wisconsin breast cancer dataset $t_{min} = 1$. It actually depends on value changing step of input data.

With that, we use the following formulas for computing minimum and maximum weights.

Minimum Weight

$$w_{min} = \frac{\tau v}{d n_i} e^{1-(t_{max}/\tau)} \tag{4.1}$$

Maximum Weight

$$w_{max} = \frac{\tau v}{d n_i} e^{1-(t_{min}/\tau)} \tag{4.2}$$

Weight is initialized by random value in range minimum and maximum weight associated with minimum weight.

$$weight = w_{min} + random(w_{min}, w_{max}, d) \tag{4.3}$$

4.4.2 Threshold, delays and membrane time constant τ

Membrane time constant (τ) have to be chosen in such way so that there will have a common range of firing time for all input values of the same class. A maximum number of delays (synapses) d and τ are not independent. The formulas for determining minimum and maximum weight in equations 4.1,4.2 goes reverse direction depending on t_{min} and t_{max}. To avoid minimum weight getting bigger than maximum weight, we need to balance values for time step and d: decrease time step and increase d. We looked at changing the delay increment, as well, but did not get conclusive results. Increasing the delay increment did not seem to affect the behavior of the network and varying the data set did not seem to affect this behavior.

Threshold does not make an effect on spike firing time. Because we use a threshold to set weight range. It just becomes a multiplicative factor on both sides inequality for checking if the membrane potential exceeds threshold. Therefore, it is possible to set threshold to 1 without affecting the performance.

There seems to be an interesting interplay between τ and the number of delays d with respect to accuracy. In our experiments, the isolines of the same accuracy on the 3D graphs with τ and d on x and y axes look essentially linear. That is, increasing d and τ proportionally to each other seems to keep the accuracy of the network the same. For a given τ, as d increases, the performance increases as well. However, in all our experiments there was another cut-off when the value of d became too large

More specifically, here are the accuracy landscapes for our datasets with respect to τ and d. Each point in this landscape is an average of 30 runs for the given values of τ and d. For each of the isolines, we used Matlab cftool to fit a line to describe that isoline as $d = p_1\tau + p_2$.

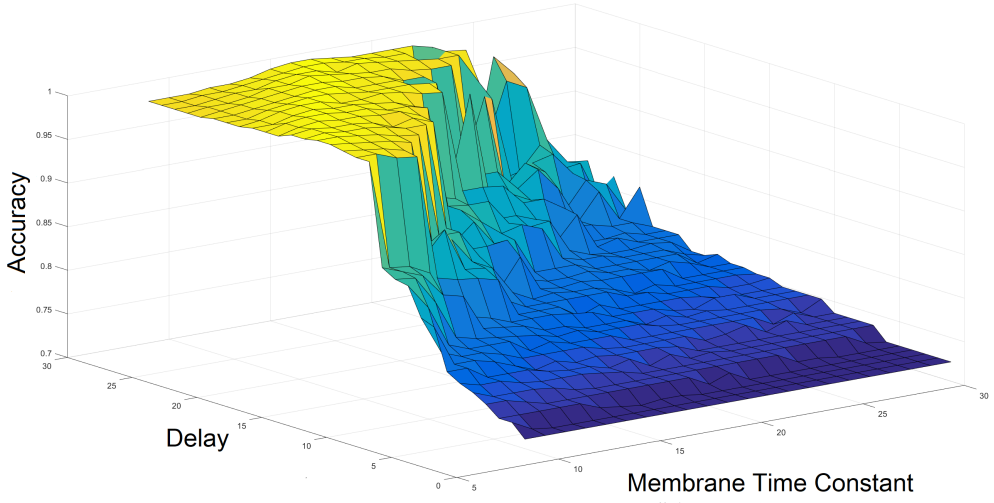

Figure 4.1: Accuracy as a function of τ and d for WBC dataset

Membrane time constant τ vs. number of delays d in WBC dataset

Accuracy for classifying WBC benchmark datasets with respect to delay and τ has been ploted in figure 4.1.

The good performance area is with d above the line between yellow and green (see figure 4.2), which is a drop in accuracy from 0.92 to 0.86. By using MATLab cftool to determine the parameters of this line, we obtain the following description of this isoline: $d = 1.583\tau + 1.861$. According to cftool, the SSE of this fit is 1.806, and adjusted R-square is 0.9064. See figure 4.4A for the plot of the data and the line. This isoline denotes the bottom boundary for the good accuracy region: accuracy drops sharply when d is below $1.583\tau + 1.861$.

Then, there is a milder cutoff as d grows too large with respect to τ, and the accuracy diminishes to below 0.98. The line (see figure 4.4B) corresponding to this cutoff is defined by $d = 12.95\tau + 0.8637$ (adjusted R-square: 0.9786).

If we zoom in on the good performance region (see figure 4.3), we will see a distinct "ridge" of the best performance, which also seems to follow a linear pattern

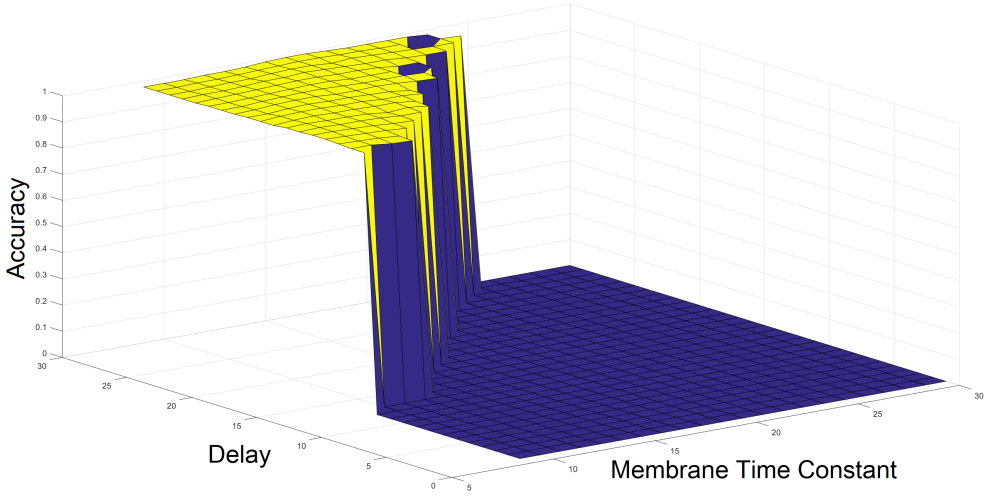

Figure 4.2: Yellow part of area accuracy of at least 97% and the rest set to zero of WBC data

More specifically, the points with accuracy > 0.99 form a line (see figure 4.4C) $d = 1.714\tau + 6.619$ (adjusted R-square: 0.9554). Thus, there is an optimal region of accuracy, bordered by isolines of d as a function of τ.

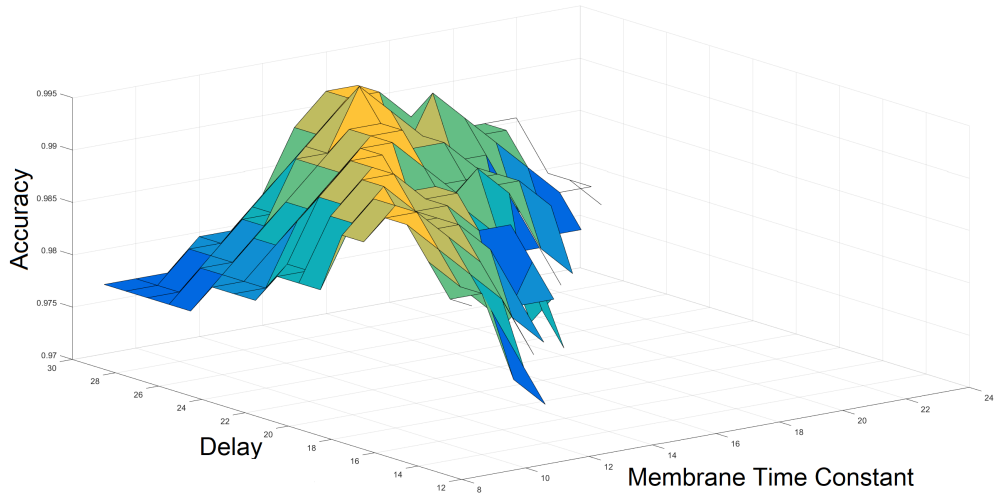

Figure 4.3: Zoom in picture on high accuracy region of WBC data

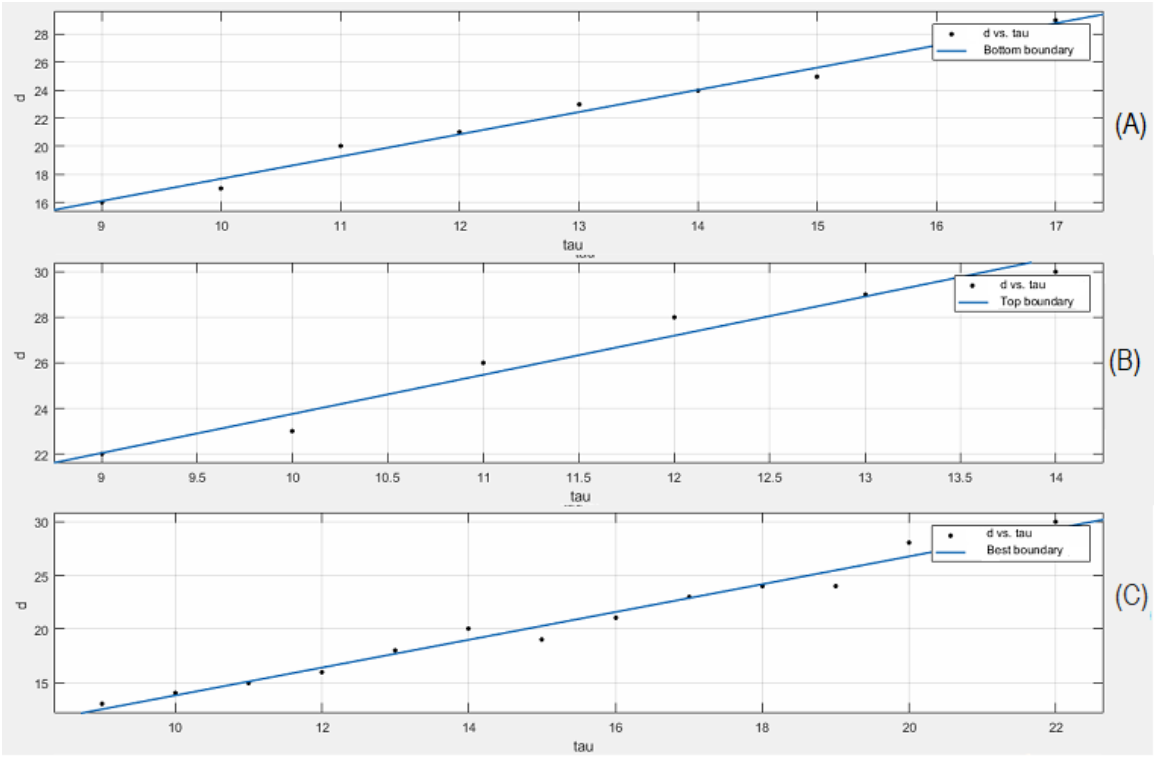

Figure 4.4: Fitted lines for borders of the good accuracy region and the "ridge of the best accuracy" in WBC

Iris dataset: τ vs. d

Similarly, performance variation in term of number of delays d and τ has been shown in figure 4.5 and 4.6 for Iris data set, as well as a permuted Iris dataset, respectively. As for WBC, there seems to be a line where accuracy is maximized for the permuted Iris dataset: namely, $d = 0.1593\tau + 16.31$. This behaviour is fairly resilient to permuting the dataset, as seen on figure 4.8.

Iris original dataset seems to have a plateu of best accuracy, bordered by a line $d = 0.9558 * \tau - 0.2052$ below, and $d = 2.464 * \tau - 4.107$ above. Similarly, for permuted Iris dataset there is a good accuracy plateau (with a faint "best accuracy ridge" close to the top border, see figures 4.7 and 4.9), bordered from above by a line $d = 2.464 * \tau - 4.107$, and from below by a line $d = 0.9558 * \tau - 0.2052$.

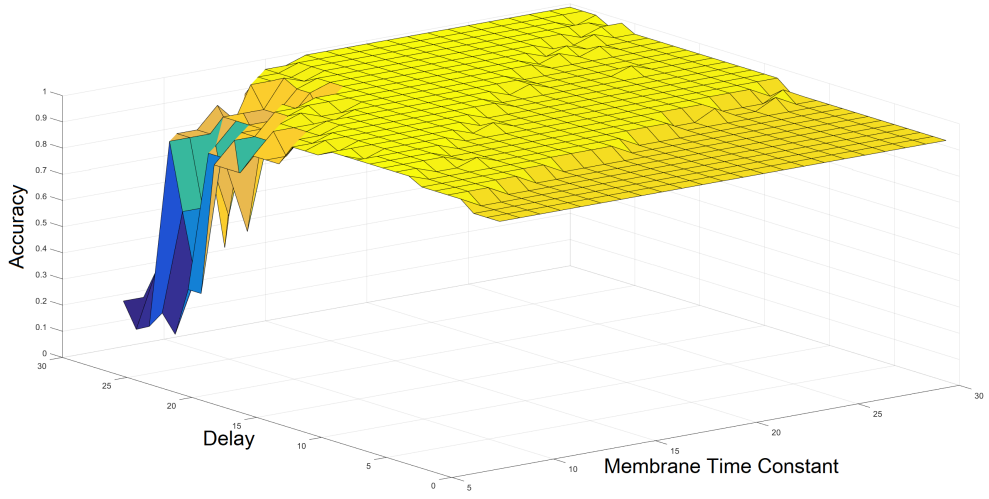

Figure 4.5: Accuracy as a function of τ and d for original Iris dataset

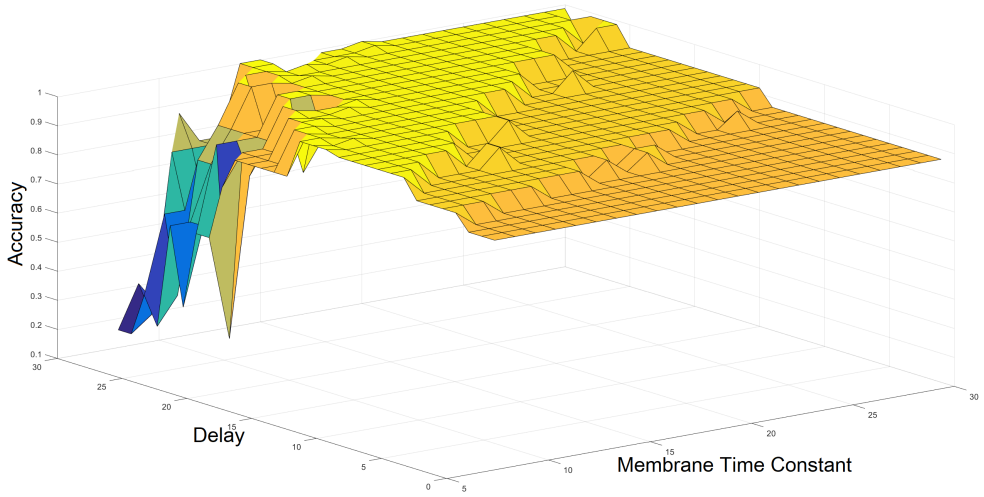

Figure 4.6: Accuracy as a function of τ and d for permuted Iris dataset

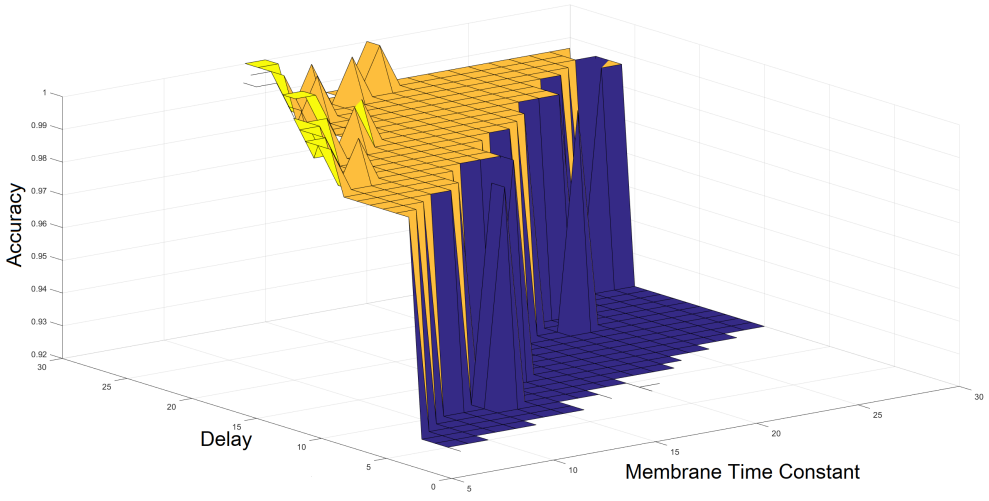

Figure 4.7: Permuted Iris dataset zoom in

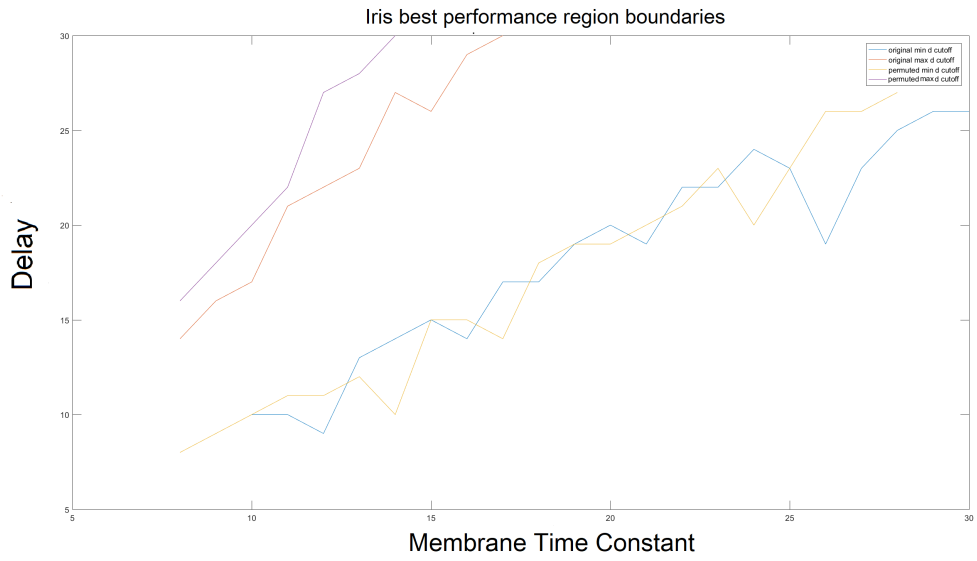

Figure 4.8: Boundaries of best performance region for Iris data set, original (blue and red lines) and permuted (orange and purple lines)

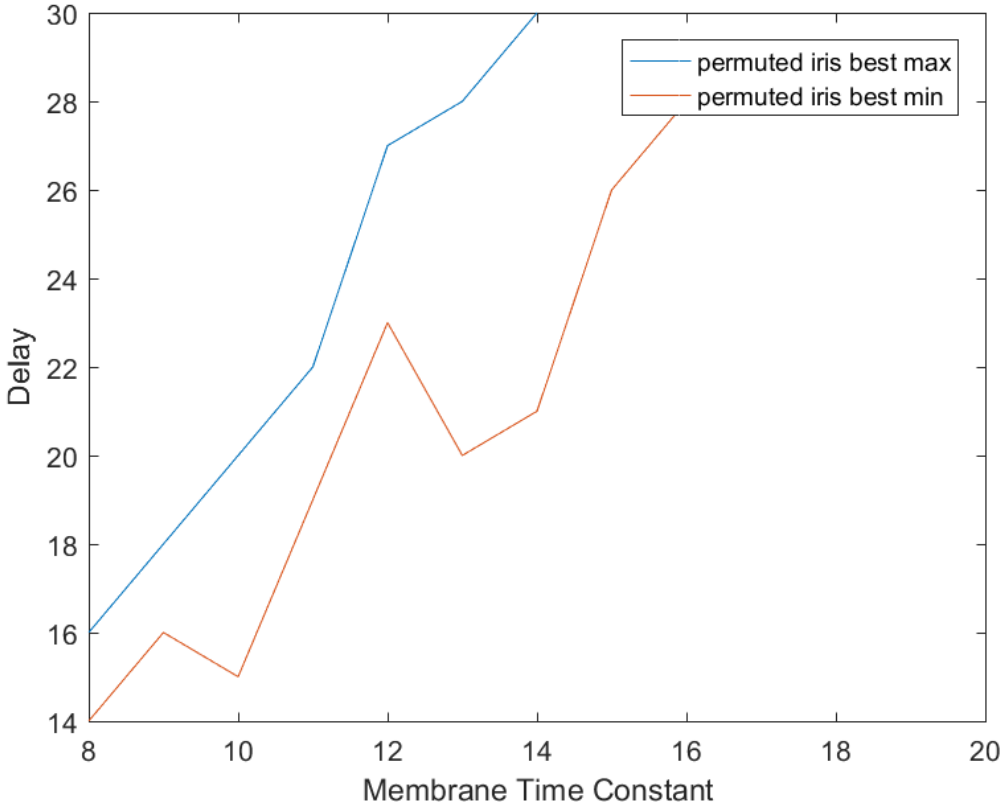

Figure 4.9: Region for permuted Iris dataset with accuracy > 99

4.4.3 Selecting Target Times

Correctly choosing target time is the biggest challenge for training spiking neural network. There is not much explanation in existing work for setting target times. For example standard XOR take inputs 0 for 0, 6 for 1 and targets are 10 and 16 without having a specific hypothesis. Automated algorithm configuration tools "ParamILS" has been used in our approach for selecting target time. First and last output spike time of network is determined which is used as a range of target time. First output spike time and last output spike time of network mainly depend on maximum weight and minimum connection weight respectively.

Earliest Spike Time:

The first spike depends on maximum weight and smallest input while latest spike time depends on minimum weight and largest input. To determine first firing time at hidden layer let us consider smallest input 0 and maximum weight between input layer and hidden layer w_{imax}.

$$v \leq n_i w_{imax} \sum_{k=1}^{d_{max}} H(t - 0 - d_k)\varepsilon(t - 0 - d_k) \tag{4.4}$$

Here, $H(s)$ denotes heavy-side step function, $H(s) = 0$ for $s < 0$; $H(s) = 1$ for $s >= 0$. The first firing time t_i at a hidden layer is considered as smallest input at output layer and maximum weight w_{hmax} between hidden and output layer for determining first firing time at the output layer.

$$v \leq n_h w_{hmax} \sum_{k=1}^{d_{max}} H(t - t_i - d_k)\varepsilon(t - t_i - d_k) \tag{4.5}$$

Latest Spike Time:

Latest spike time of network is determined as like earliest spike time with considering minimum weight instead of taking the maximum weight. The latest firing time at hidden layer is calculated by the following equation.

$$v \leq n_i w_{imin} \sum_{k=1}^{d_{max}} H(t - 0 - d_k)\varepsilon(t - 0 - d_k) \tag{4.6}$$

Similarly, the latest firing time at output layer is:

$$v \leq n_h w_{hmin} \sum_{k=1}^{d_{max}} H(t - t_i - d_k)\varepsilon(t - t_i - d_k) \tag{4.7}$$

4.4.4 Target times

When we first encountered the heuristic of choosing target times by taking the average in each class of the outputs of a randomly intialized network, it seemed a surprisingly ad hoc approach. However, our experiments with ParamILS seem to indicate that it actually works fairly well for the types of dataset we used. More specifically, even though the classes in WBC and Iris are not quite linearly separable, the instances from different classes look, on average, quite distinct.

Consider the following figures, which show a such a simple metric as a sum of input values for each class. From figure 4.10, it is clear that the range of this sum for the malignant class is mostly from 30 up, with an average around 55, whereas in benign class very few instances have values above 25, and none above 50. More precisely, the sum of the values in malignant class is from 22 to 88, while in benign class the sums range from 5 to 49. That is, it is natural to expect that on a randomly trained network the average output spike in the benign data will be much earlier than in malignant data. And indeed, ParamILS came up with values 7 and 8 for target

spikes for benign and malignant data, respectively.

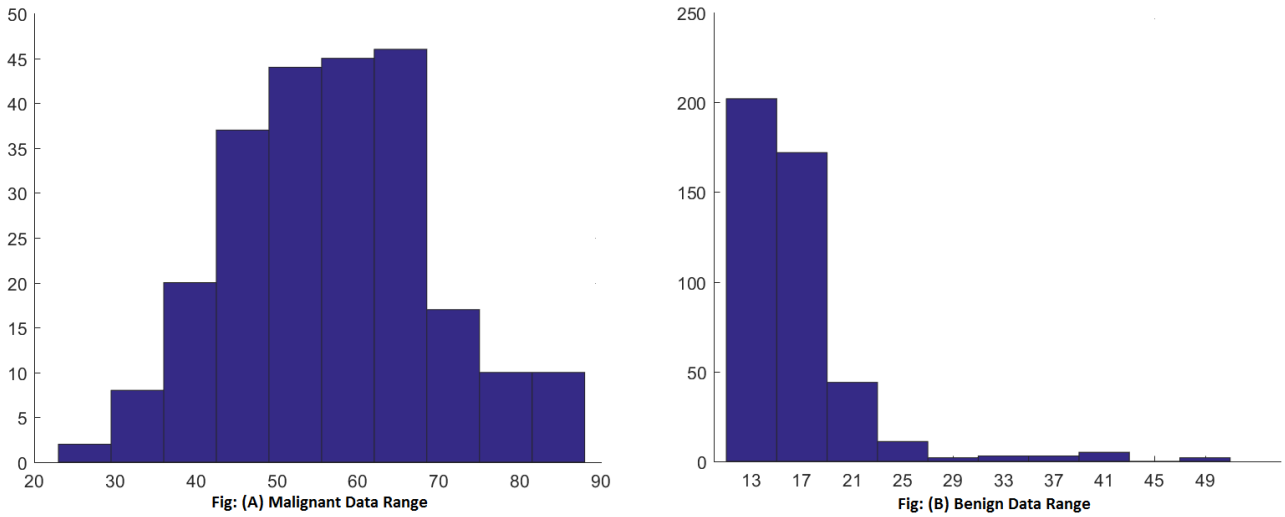

Figure 4.10: Sums of inputs for malignant and benign classes of Wisconsin Breast Cancer data

Similarly, for Iris dataset the classes have different average sums of inputs (see figure 4.11), though here the overlap between Versicolor and Virginica is more significant. For this dataset, ParamILS gave target values of 4 for Setosa, 5 for Versicolor and 6 for Virginica.

Input data range of Iris data is [0.1,7.9], while for WBC it is [1,10].

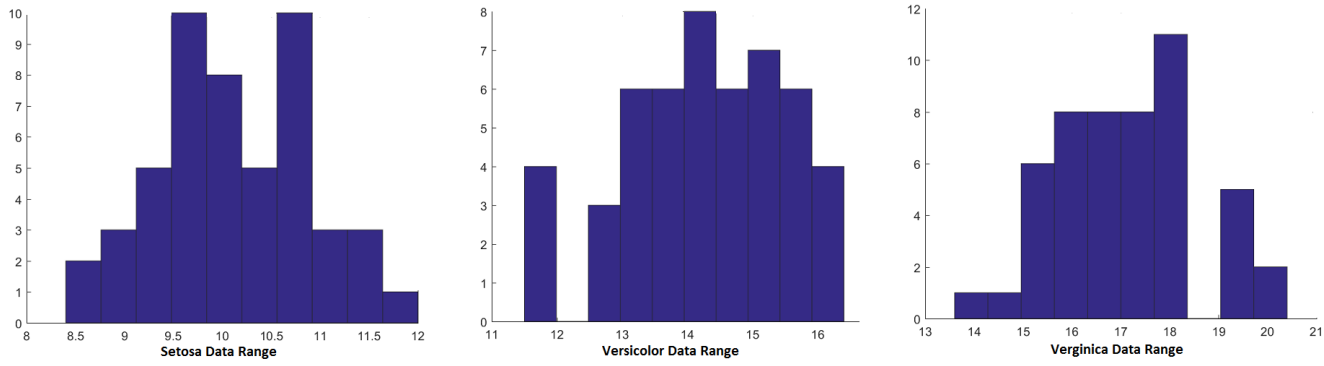

Figure 4.11: Sums of inputs for Setosa, Versicolor and Virginica classes in Iris dataset

Chapter 5

Conclusion

In this project, we have looked at configuring artificial neural network training algorithms, in particular the spiking neural network training algorithm SpikeProp due to Bohte [BKLP02]. We have shown that using automated algorithm configuration tools it is possible to achieve better accuracy than reported in the literature.

That raises the question of comparing the quality of different algorithms or even different modifications of the same algorithm: how can we be reasonably sure that the performance increase is due to the modification in question, as opposed to a tweak to parameters? We think that using automated algorithm configuration tools would allow for a more fair comparison of algorithms, and, though it does not constitute a formal proof, it would alleviate some of such concerns.

There is a number of directions that we are currently investigating. So far, we used the simplest possible encoding of the input: assigning a number to a corresponding discrete time slot in the coding interval. The next step would be to compare the direct encoding of the input with a more complex way such as Gaussian or cosine receptive fields encodings. These encodings come with their own sets of parameters to be optimized together with the rest of the network parameters, which we would do

using ParamILS.

Another project would be to change the way the output of the network is represented. In particular, rather than having a single output neuron and match classes to spike times, we would like to look at the winner-takes-all representation, where there is an output neuron for each class, and the output neuron that fires first is considered to be the answer. Using this representation would allow us to sidestep the question of target times, potentially just training for the earliest-possible versus latest-possible spike. So far, though, we were not able to achieve good results using this output representation.

Overall, we would like to understand the question of target times better. Ideally, given a data set, we would like to be able to analyse it and derive the desired target times just from the statistical properties of the datasets (there, we would need to calculate the other parameters, particularly τ and d, simultaneously, as the target times depend on these parameters). It would be good to understand spiking neural networks, at least of the delayed type used with SpikeProp, enough to derive formulas for setting the parameters of a training algorithms, instead of relying on heuristics.

Finally, we would like to return to the project from which we started: augmenting spiking neural networks with glia, in the style of [Saj14]. This would create artificial networks of cells of different types and functions, making them more biologically realistic. Additionally, we would like to leverage the other types of improvements in modern artificial neural networks, and work with more complex datasets. This thesis makes a step towards understanding how to configure and compare the existing algorithms, and gives us ways to make such future comparisons more meaningful.

Bibliography

[AGPPP12] Alberto Alvarellos-González, Alejandro Pazos, and Ana B Porto-Pazos. Computational models of neuron-astrocyte interactions lead to improved efficacy in the performance of neural networks. *Computational and mathematical methods in medicine*, 2012, 2012.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[BKLP02] Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48:17–37, 2002.

[Boh] Sander M Bohte. Artificial Spiking Neural Networks. `https://slideplayer.com/slide/4642397/`. Accessed: 2017-10-18.

[BSA89] Tobias Bonhoeffer, Volker Staiger, and AMHJ Aertsen. Synaptic plasticity in rat hippocampal slice cultures: local" Hebbian" conjunction of pre-and postsynaptic stimulation leads to distributed synaptic enhancement. *Proceedings of the National Academy of Sciences*, 86(20):8113–8117, 1989.

[Dav13] Sergio Davies. *Learning in spiking neural networks*. PhD thesis, University of Manchester, 2013.

[DSMH85] Marian C Diamond, Arnold B Scheibel, Greer M Murphy, and Thomas Harvey. On the brain of a scientist: Albert Einstein. *Experimental neurology*, 88:198–204, 1985.

[GK02a] Wulfram Gerstner and Werner M Kistler. Mathematical formulations of Hebbian learning. *Biological cybernetics*, 87(5-6):404–415, 2002.

[GK02b] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge University Press, 2002.

[GKvHW96] Wulfram Gerstner, Richard Kempter, J Leo van Hemmen, and Hermann Wagner. A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383(LCN-ARTICLE-1996-002):76–78, 1996.

[GS06] Robert Gütig and Haim Sompolinsky. The tempotron: a neuron that learns spike timing–based decisions. *Nature neuroscience*, 9:420–428, 2006.

[HHLBS09] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[HHS07] Frank Hutter, Holger H Hoos, and Thomas Stützle. Automatic algorithm configuration based on local search. In *AAAI*, volume 7, pages 1152–1157, 2007.

[HLQ+16] Hong, Xiurui Liu, Xie, Guisong Qu, Malu Zhang, and Jürgen Kurths. An Efficient Supervised Training Algorithm for Multilayer Spiking Neural Networks. *PloS one*, 11:e0150329, 2016.

[HOT06] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[HZHS13] Shicheng Hu, Zhaoze Zhang, Qingsong He, and Xuedong Sun. An iterated local search algorithm for a place scheduling problem. *Mathematical Problems in Engineering*, 2013, 2013.

[IRMGM+15] Taras Iakymchuk, Alfredo Rosado-Muñoz, Juan F Guerrero-Martínez, Manuel Bataller-Mompeán, and Jose V Francés-Víllora. Simplified spiking neural network architecture and STDP learning algorithm applied to image classification. *EURASIP Journal on Image and Video Processing*, 2015:4, 2015.

[Izh03] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14:1569–1572, 2003.

[JTG03] Renaud Jolivet, J Timothy, and Wulfram Gerstner. The spike response model: a framework to predict neuronal spike trains. In *Artificial Neural Networks and Neural Information Processing—ICANN/ICONIP 2003*, pages 846–853. Springer, 2003.

[Kis02] Werner M Kistler. Spike-timing dependent synaptic plasticity: a phenomenological framework. *Biological cybernetics*, 87(5):416–427, 2002.

[KP05] Andrzej Kasinski and Filip Ponulak. Experimental demonstration of learning properties of a new supervised learning method for the spiking

neural networks. *Artificial Neural Networks: Biological Inspirations–ICANN 2005*, pages 145–152, 2005.

[KP06] Andrzej Kasiński and Filip Ponulak. Comparison of supervised learning methods for spike time coding in spiking neural networks. *International Journal of Applied Mathematics and Computer Science*, 16:101–113, 2006.

[KPK06] M Kraft, F Ponulak, and A Kasinski. FPGA implementation of ReSuMe learning in Spiking Neural Networks. In *Proceedings of EPFL LATSIS Symposium 2006, Dynamical Principles for Neuroscience and Intelligent Biomimetic Devices*, pages 97–98, 2006.

[KvRST02] Adam Kepecs, Mark CW van Rossum, Sen Song, and Jesper Tegner. Spike-timing-dependent plasticity: common themes and divergent vistas. *Biological cybernetics*, 87(5-6):446–458, 2002.

[Maa95] Wolfgang Maass. On the computational complexity of networks of spiking neurons. In *Advances in neural information processing systems*, pages 183–190, 1995.

[Maa96] Wolfgang Maass. Lower bounds for the computational power of networks of spiking neurons. *Neural computation*, 8:1–40, 1996.

[Maa97] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10:1659–1671, 1997.

[Maz] Matt Mazur. A Step by Step Backpropagation Example. `https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/`. Accessed: 2017-08-8.

[mba] mbax9an4. https://github.com/mbax9an4/SpikeProp. Accessed: 2017-08-8.

[MLB06] Sam McKennoch, Dingding Liu, and Linda G Bushnell. Fast modifications of the spikeprop algorithm. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 3970–3977. IEEE, 2006.

[MLFS97] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275(5297):213–215, 1997.

[Mus] Sabnam Mustari. https://github.com/Sabnam-Mustari/SpikingNet. Accessed: 2017-09-20.

[Ng] Andrew Ng. Machine Learning. https://www.coursera.org/learn/machine-learning/home/week/6. Accessed: 2017-09-07.

[PA10] Filip Ponulak and Kasínski Andrzej. Supervised Learning in Spiking Neural Networks with ReSuMe: Sequence Learning, Classification, and Spike Shifting. *Neural Computation*, 22:467–510, 2010.

[PBR08] F Ponulak, D Belter, and S Rotter. Adaptive movement control with spiking neural networks, Part I: feedforward control. In *Proceedings of Recent Advances in Neuro-Robotics, Symposium: Sensorimotor Control*. Freiburg University Freiburg, 2008.

[PF10] Alfredo Pereira and Fabio Augusto Furlan. Astrocytes and human cognition: modeling information integration and modulation of neuronal activity. *Progress in neurobiology*, 92:405–420, 2010.

[PGP09] Ana Belén Porto Pazos, Alberto Alvarellos González, and Félix Montañés Pazos. Artificial neuroglial networks. In *Encyclopedia of Artificial Intelligence*, pages 167–171. IGI Global, 2009.

[Pon05] Filip Ponulak. ReSuMe-new supervised learning method for Spiking Neural Networks. *Institute of Control and Information Engineering, Poznan University of Technology.*, 2005.

[Pon06] Filip Ponulak. Supervised learning in spiking neural networks with ReSuMe method. *Phd, Poznan University of Technology*, 46:47, 2006.

[PSA14] Gertrudis Perea, Mriganka Sur, and Alfonso Araque. Neuron-glia networks: integral gear of brain function. *Frontiers in cellular neuroscience*, 8, 2014.

[Saj13] Zahra Sajedinia. Artificial Glia Astrocytes; a New Element in Adaptive Neuro Fuzzy Inference Systems. In *22th Annual IEEE Newfoundland Conference proceedings (NECEC 2013)*, 2013.

[Saj14] Zahra Sajedinia. Artificial Astrocyte Networks, as Components in Artificial Neural Networks. In *International Conference on Unconventional Computation and Natural Computation*, pages 316–326. Springer, 2014.

[WMH+11] John J Wade, Liam J McDaid, Jim Harkin, Vincenzo Crunelli, and JA Scott Kelso. Bidirectional coupling between astrocytes and neurons mediates learning and dynamic coordination in the brain: a multiple modeling approach. *PloS one*, 6:29445, 2011.

[WMM+06] QX Wu, T Martin McGinnity, Liam P Maguire, B Glackin, and Ammar Belatreche. Learning under weight constraints in networks of temporal encoding spiking neurons. *Neurocomputing*, 69:1912–1922, 2006.

[WMSS10] John J Wade, Liam J McDaid, Jose A Santos, and Heather M Sayers. SWAT: a spiking neural network training algorithm for classification problems. *IEEE Transactions on Neural Networks*, 21:1817–1830, 2010.

[XQYK17] Xiurui Xie, Hong Qu, Zhang Yi, and Jürgen Kurths. Efficient Training of Supervised Spiking Neural Network via Accurate Synaptic-Efficiency Adjustment Method. *IEEE transactions on neural networks and learning systems*, 28:1411–1424, 2017.

[XZHY13] Yan Xu, Xiaoqin Zeng, Lixin Han, and Jing Yang. A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks. *Neural Networks*, 43:99–113, 2013.

[XZZ13] Yan Xu, Xiaoqin Zeng, and Shuiming Zhong. A new supervised learning algorithm for spiking neurons. *Neural computation*, 25:1472–1511, 2013.

[Yad15] Yadav Anupam Kumar Manoj Yadav, Neha. *An Introduction to Neural Network Methods for Differential Equations*. Springer, 2015.