Supervised Spike-Time Learning with an Adaptive Learning Rate in Spiking Neural Networks

A Thesis

submitted to

Indian Institute of Science Education and Research Pune in partial fulfillment of the requirements for the BS-MS Dual Degree Programme

by

Vaishnavi V

Indian Institute of Science Education and Research Pune

Dr. Homi Bhabha Road,

Pashan, Pune 411008, INDIA.

Date: 1 April, 2023

Under the guidance of

Supervisor : Dr. Venkatakrishnan Ramaswamy,

Assistant Professor,

Department of Computer Science,

BITS Pilani(Hyderabad Campus)

From May 2022 to Mar 2023

INDIAN INSTITUTE OF SCIENCE EDUCATION AND RESEARCH PUNE

Certificate

This is to certify that this dissertation entitled **Supervised Spike-Time Learning with an Adaptive Learning Rate in Spiking Neural Networks** towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Vaishnavi V at BITS Pilani(Hyderabad Campus) under the supervision of Dr. Venkatakrishnan Ramaswamy, Assistant professor, Department of Computer Science, during the academic year 2022-2023.

Dr. Venkatakrishnan Ramaswamy

Committee:

Name of your Guide : Dr. Venkatakrishnan Ramaswamy

Name of Your TAC : Dr. Arunava Banerjee

Dr. Suhita Nadkarni

Dedicated to Mom and Dad

Declaration

I hereby declare that the matter embodied in the report entitled **Supervised Spike-Time Learning with an Adaptive Learning Rate in Spiking Neural Networks** are the results of the work carried out by me at BITS Pilani(Hyderabad Campus) under the supervision of Dr. Venkatakrishnan Ramaswamy, Assistant professor, Department of Computer Science, for the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune and the same has not been submitted elsewhere for any other degree.

Vaishnavi V

Date: 10/04/2023

Table of Contents

List of Figures

List of Tables

Abstract

Reliable communication of neuronal information by neurons of the central nervous system to its downstream neurons involves transformation of input spike trains to specific output spike trains. The spike train to spike train transformation problem has been addressed by numerous studies in the past but we focus our attention on the synaptic weight update rule proposed in Banerjee(2016) which aligns two spike trains using only the spike time disparities. We implement the synaptic weight update rule on a single neuron receiving multiple synaptic inputs and re-evaluate the results of Banerjee(2016). We identify the problems that are faced during implementation of the rule and suggest methods to address these problems.

During implementation, we identified that learning slows down due to silent synapses or (synapses whose weights do not change much) or quiescent neurons and manual tuning of hyperparameters - learning rate and cap on update vector. The first problem is difficult to solve but we suggest a potential solution to the problem in the Discussion section. The problem due to a fixed learning rate and update vector cap is solved by using gradient descent with momentum and other adaptive gradient based optimisers - AdaGrad, RMSProp and Adam. The choice of optimiser is very important especially when dealing with sparse gradient tasks and large spiking neural networks because optimisers take into account the characteristics of the data and assign a per-parameter learning rate and accelerate the learning process. Out of gradient descent with momentum and other three optimisers used, Adam performed remarkably well in converging the weights of the learning neuron towards the target weights, which is used as a measure of effectiveness of the learning rule.

Acknowledgements

It is my privilege and pleasure to express my deep sense of gratitude, sincere and respectful thanks to my supervisors **Dr. Venkatakrishnan Ramaswamy**, Assistant Professor, Department of Computer Science, BITS Pilani (Hyderabad campus) and **Dr. Arunava Banerjee**, Associate Professor, Department of Computer & Information Science & Engineering, University of Florida, who have been a source of inspiration for me. Their valuable guidance, kind advice, constant encouragement and timely suggestions throughout the course of this project have helped me immensely.

I am thankful to **Dr. Suhita Nadkarni**, Associate Professor, Department of Biology, IISER Pune for her guidance and advice during the mid-year evaluations that helped me charter the course of this project and motivated me to ask the right questions and search for the right answers.

I take this opportunity to thank BITS Pilani(Hyderabad campus) for giving me access to Sharanga, the high performance computing cluster and Brain server during the course of this project.

I express my thanks to the library of Indian Institute of Science Education and Research Pune for their help in providing me with reading materials for my project.

I express my thanks to my parents and my friends who have been a source of endless motivation and support throughout the project period.

Vaishnavi V.

Contributions

Contributor name	Contributor role
Vaishnavi V, Dr. Venkatakrishnan Ramaswamy and Dr. Arunava Banerjee	Conceptualization Ideas
Vaishnavi V, Dr. Venkatakrishnan Ramaswamy and Dr. Arunava Banerjee	Methodology
Vaishnavi V	Software
Vaishnavi V	Validation
Vaishnavi V	Formal analysis
Vaishnavi V	Investigation
Dr. Venkatakrishnan Ramaswamy	Resources
Vaishnavi V	Data Curation
Vaishnavi V	Writing - original draft preparation
Vaishnavi V	Writing - review and editing
Vaishnavi V	Visualization
Dr. Venkatakrishnan Ramaswamy and Dr. Arunava Banerjee	Supervision
Dr. Venkatakrishnan Ramaswamy and Dr. Arunava Banerjee	Project administration

ABBREVIATIONS

AdaGrad	Adaptive Gradient Algorithm
AHP	Afterhyperpolarization
ANN	Artificial Neural Network
CNS	Central Nervous System
DNN	Deep Neural Network
GD	Gradient Descent
MAPE	Mean Absolute Percentage Error
PSP	Postsynaptic Potential
RMSProp	Root Mean Square Propagation
SNN	Spiking Neural Network
SRM	Spike Response Model
VPD	Victor-Purpura Distance

1. Introduction

1.1 Background

1.1.1 Rate based and spike based coding

In the central nervous system, neurons communicate with each other through electrical signals called action potentials or spikes. Spikes can be considered as precisely timed discrete events occurring in the brain. There are two lines of neural theory that describe the basis of computation in the neural networks of the brain. The classical view of neural computation suggests that the information related to a stimulus is conveyed through the firing rate of neurons. Firing rate is an abstract mathematical concept defined in the limit of an infinite number of spikes (Brette, 2015). In practice, in order to reliably estimate the frequency of neuron firing, it is typically necessary to observe at least two spikes per neuron, and neural responses need to be averaged over multiple trials and a significant time window. In the rate coding perspective, neural activity can be described sufficiently by the frequency of firing, and the precise timing of individual spikes is not regarded as highly significant.

According to spike based neural coding theories, neurons use the precise timing of individual spikes to communicate information.The question of whether precise timing of spikes with a precision of milliseconds is relevant to neural computation is still being debated. A number of scientists have presented experimental evidence that in many animal sensory pathways, precise patterns of spikes encode temporal structure of the stimulus especially for stimuli with a high temporal resolution (30-300ms). Nemenman, et al. (2008) while investigating the motion sensitive neurons of the fly visual system discovered that a considerable amount of visual information is conveyed through specific patterns of action potentials, with a precision of fractions of a millisecond. The varying patterns of spike timings represent slightly distinct trajectories of flight extracted from the stimulus ensemble. Bialek et al.(1991) and De Ruyter van Steveninck et al. (1997) conducted extensive research on the time-response of the H1 neuron in flies

during flight. They developed a method to decode signals from H1 and reconstruct the original environment experienced by the fly. They discovered that individual spikes from H1 were essential in determining the velocity estimate at each time point during decoding. Their findings revealed that observing spikes with greater temporal precision led to an increase in the accuracy of the decoded signal. Johansson and Birznieks (2004) proposed that the relevant features of tactile sensory information regarding mechanical events at the fingertips could be effectively communicated through the initial timing of the first spike generated by groups of sensory neurons at the fingertips. This empirical evidence has been confirmed by a different study on rat barrel cortex in which computational analysis of information from neurons in the barrel cortex confirmed that stimulus information is encoded in the precise timing of first-spike from the sensory neurons (Panzeri et al., 2001). Other ways in which precise spike timing of the spikes is used is in latency coding where the spike times relative to each other is used (Gawne et al. 1996; Middlebrooks et al. 1994), coding by synchrony where groups of neurons that encode different pieces of information about the same object fire in synchrony (Izhikevich, 2006; Brette, 2012) or resonant burst coding where the frequency of the burst allows for selective activation of specific neurons that are attuned to that frequency, thus conveying information in a precise manner (Zeldenrust, 2018). Rank order coding (Thorpe et al., 2001) and predictive spike coding (Deneve, 2008) are temporal coding theories that rely on the asynchronous firing of neurons

1.1.2 Spiking neural networks

In the past decade, deep neural networks have achieved enormous success in various fields attributed to their ability to learn and predict based on inputs of large amounts of data. However, these networks are highly energy-intensive, require large amounts of data and have high computational costs, especially when carrying out real-time tasks. Biologically plausible spiking neural networks are able to overcome the bottlenecks of artificial neural networks and can perform the same type of computations as the traditional ANNs (Maass, 1997). Unlike ANNs which use single, static and continuous valued activation functions, SNNs use discrete spike events for computation. A spiking

neural network architecture consists of spiking neurons based on various neuron models such as Hodgkin-Huxley (Hodgkin & Huxley, 1952), spike response model(SRM) (Gerstner et al., 1993 ; Gerstner 1995), Izhikevich model (Izhikevich, 2003) or Leaky Integrate and Fire model (Lapicque, 1907) connected to each other through synapses with adjustable scalar weights.. They are functionally similar to the networks found in the central nervous system and have the properties of sparse representation and temporal coding. Maass (1997) demonstrated that for spiking neurons, when the input is made up of spikes that occur at different times (i.e., asynchronous spiking), the specific timing of the resulting output spike can be understood as a computation performed on the input i.e., under certain parameter ranges, spiking neurons are capable of calculating a linear combination of the spike times. Maass (1997) also proved theoretically that SNNs are more computationally powerful than ANNs. However, training spiking neural networks is particularly challenging due to a number of factors. First, in contrast to the basic spatial activation vectors used in DNNs, spatiotemporal spike patterns are utilized as input stimuli and output in the spiking neural network. Therefore the cost function used for training should cater specifically for the SNN. The second challenge with many neuron models is that they lack differentiability at the time of a spike, making it tough to use techniques based on gradients. Additionally, the spike reset mechanism in many spiking neurons creates an inherent self-memory, which presents a significant challenge when attempting to address it analytically (Zenke & Ganguli, 2018). The learning rules in networks whether it be artificial neural networks or spiking neural networks, involve adjusting the scalar weights of synapses. The training methods for spiking neural networks include supervised training with gradient descent and spike backpropagation, unsupervised learning with synaptic learning rules and reinforcement learning. Here we discuss and implement supervised learning in spiking neural networks.

1.1.3 Supervised learning in SNNs

Spiking neural networks utilize supervised learning, which entails modifying the weights through gradient descent on a cost function that evaluates the similarity between the

observed output of the network and the desired output. The objective is to minimize the error, often referred to as readout error, between the expected and actual output spike trains in response to various inputs. By doing so, the spiking neural network can learn to classify inputs with precision and generate the appropriate output spikes in response to the input spike train.

1.1.4 Backpropagation in SNNs

The following formula is a core expression derived from the chain rule , used in all types of backpropagation algorithms (Rumelhart, 1986; Lee, 2016).

$$\delta_j^\mu = g'(a_j^\mu) \sum_k w_{ij} \delta_i^\mu$$

δ_j^μ and δ_i^μ are partial derivatives of the cost function at neuron j and i respectively with respect to the input to the neurons. w_{ij} is the weight of the feedforward connections from neuron i to neuron j. $g(.)$ is the activation function applied to a_j^μ the net input to neuron j.

There are notable issues with implementing this backpropagation algorithm in the brain and the bio-plausible spiking neural networks. To compute $g'(.)$ with respect to w_{kj}, we need to take the derivative of $g(.)$. However, in the case of a spiking neuron where $g(.)$ is represented by a sum of Dirac delta functions, the derivative of $g(.)$ does not exist. This means that calculating the derivative of $g(.)$ with respect to w_{kj} is not possible. This problem is addressed by using substitute derivatives. To enable backpropagation in multi-layer spiking neural networks (SNNs), it is necessary to find a proxy, which is a real-valued function that is almost everywhere differentiable. This proxy function is used as a substitute for the actual spiking activity in the network and enables the use of spike-based learning rules. By using this proxy, backpropagation can be applied to train the network and adjust the weights of its connections. The proxy function should accurately capture the behavior of the spiking neurons and their interactions, so that the resulting weights can effectively support the network's learning and prediction tasks.

Next we look at prior works on the supervised learning of precisely timed spikes.

1.1.5 Previous work

SpikeProp (Bohte et al., 2002) is a supervised learning algorithm with a temporal coding paradigm that uses the timing of single spikes for backpropagation. It overcomes the discontinuous nature of spikes by approximating the threshold function around the spike time. Bohte et. al.(2002) was able to show that the algorithm can learn complex non-linear functions, such as XOR classification. Booij and tat Nguyen (2005) expanded Spikeprop to include multiple spiketimes from a single neuron. However, the network error remained the sum of squared differences between the desired spike time and the first spike time of the output neurons. Since Spikeprop is computationally expensive, it has not been used in large-scale deep learning applications as of now.

The Tempotron proposed by Gutig & Sompolinsky(2006) has been another successful supervised learning rule. Tempotron learning is useful in categorizing different input classes. Out of two different classes of input spike patterns, tempotron classifies these inputs into two class labels + and -, depending on whether the output neuron generates a spike or remains quiescent. It uses a gradient descent approach on a cost function measuring the difference between the maximum potential of the neuron and its firing threshold. However, the precise timing of the output spiketrain from the tempotron does not carry any information. Chronotron(Florian, 2012) classifies the input spike trains to multiple categories based on the output spike times with millisecond precision. Chronotron utilizes a variation of Victor & Purpura (VP) distance (Victor & Purpura, 1996) as its error function. To determine the VPD, a metric used to quantify the temporal disparity between two neural spike trains, the minimum expenditure required to transform one spike train into the other is computed. This expenditure encompasses the addition, removal, or shifting of individual spikes. However, E-learning in chronotron is not biologically plausible since it does not allow online learning and because of its dependence on time delocalised synaptic variables.

Another recent alternative approach is remote supervised learning(ReSuMe) proposed by Ponulak(2005) which employs the interaction between two spike-timing dependent plasticities. It can be seen as a spiking analogy to the Widrow-Hoff rule (Widrow &

Hoff,1960). The Widrow-Hoff rule reduces the cost function without gradient calculations. This helps to bypass the first issue discussed earlier with backpropagation in spiking neurons. However, the rule assumes that the neuron's response is linear which is highly problematic because even though the subthreshold potential of the neuron has a linear dependence on synaptic weights, the suprathreshold potential has a non-linear dependence owing to the membrane potential reset after a spike.The inherent sensitivity of the precise spike times due to the non-linearity could lead to appearance and disappearance of spikes with small changes to one of the output spike times.

One problem with spike timing-based approaches is they cannot learn when the neuron is quiescent, because then the spike times are not defined. Superspike circumvents this problem without injecting noise even when the hidden layer neurons are non-spiking. Zenke and Ganguli (2018) use an estimation technique where the partial derivative of hidden units is approximated as the product of the presynaptic spike train and a nonlinear function of the postsynaptic voltage ie,

$$\frac{\partial S}{\partial w_{i,j}} \rightarrow \sigma'(U_i)\frac{\partial U_i}{\partial w_{i,j}}$$

Many of these algorithms are also not physiologically realistic because their applicability is restricted to specific spiking neuron models that can be analyzed mathematically. Additionally, many learning algorithms are limited to solving particular information coding schemes, such as single spike time coding. As a result, these models and algorithms may not fully represent the complex mechanisms of the brain.

1.2 Existing Problems

1.2.1 Silent synapses and saturated synapses

A dead or silent synapse is a concern in spiking neural networks. If a synapse of the postsynaptic neuron is silent i.e, has little or no contribution to the spiking of the neuron, the algorithm will produce a negligible gradient value for the weight of that synapse in

the error function(Banerjee, 2016). As a result, the learning at that synapse can become slow because the update vector for that synapse will also be small.

Saturated neuron problem occurs when the neurons in the hidden layer fire all the time, at very high rates irrespective of the input to the neuron. When spike rates are high, there are several possible combinations of synaptic weights that can generate the same spike pattern. This makes it difficult for the gradient descent method to identify a unique set of synaptic weights towards which to converge (Banerjee,2016). As a result, the update rule can cause the synaptic weights to move randomly from one set of target weights to another, without actually converging towards a specific set of weights. This means that the synaptic weights do not reach a steady state, and the error $E(\cdot)$ remains unstable and high throughout the process.

1.2.2 Fixed learning rate and update vector cap

Selecting an optimal learning rate for training a network can be a difficult task when the learning rate remains constant. When the learning rate is too low, convergence can be slow, but if it is too high, it can impede convergence and cause the loss function to fluctuate or deviate from its minimum value. Additionally, using the same learning rate for all parameter updates can be problematic when dealing with sparse data or synapses with varying rates of learning. In such instances, updating all synapses equally may not be desirable, and it may be advantageous to provide a larger update to infrequently occurring features, i.e., synapses with slow learning.

1.3 Addressing the problems

1.3.1 Adaptive gradient methods

The properties of the data being viewed are not taken into account by conventional gradient approaches. The reason adaptive gradient methods are distinct is that they take into account the geometry of the data from earlier iterations and give rarely

occurring features a greater learning rate and frequently occurring features a lower learning rate.

In this paper, we use adaptive gradient methods in an attempt to solve the problem of manually changing the learning rate and the update vector cap when the learning slows down. When most of the update vector is getting capped it is necessary to adjust the cap or else the weights will be changed only in the direction of the cap vector and no learning happens.

a) AdaGrad

One such adaptive gradient method is called AdaGrad. AdaGrad performs informative gradient descent learning by taking into account the geometry of the data in the earlier iterations. Let f be a function and ∂f be the subdifferential set of function f. Let's say $g = f'(w) \quad \epsilon \quad \partial f(w)$ is the gradient of w with respect to time t and $g_t \quad \epsilon \quad \partial f_t(w_t)$. In an online learning scenario with a regret bound model , the learner predicts the weight vector $w_t \quad \epsilon \quad W \subseteq \mathcal{R}^d$. The goal of AdaGrad is to minimize the regret

$$R(T) = \sum_{t=1}^{T} f_t(w_t) - inf_{w\epsilon W} \sum_{t=1}^{T} f_t(w*)$$

where $w*$ is the set of desired weights.

At every time step the learner receives $g_t \epsilon \partial f_t(w_t)$. We perform a project gradient update

$$w_{t+1} = \Pi_W(w_t - \eta g_t) = argmin||w - (w_t - \eta g_t)||_2^2$$

where $||.||_A$ is the Mahalanobis norm denoting the projection of a point y onto X. This can also be written as :

$$w_{t+1} = \Pi_W^{G_t^{1/2}}(w_t - \eta G_t^{-1/2} g_t)$$

For a neuron i at time t+1, AdaGrad modifies the learning rate η at each time step for each parameter w_i based on the past gradients of w_i according to the equation:

$$w_{t+1,i} = w_{t,i} - g_{t,i}\frac{\eta}{\sqrt{G_{t,ii} + \epsilon}}$$

Where $g_{t,i}$ is the gradient of w with respect to time t, η is the learning rate, G is the outer product of all previous gradients $G = \sum_{\tau=1}^{T} g_\tau g_\tau^T$ and ϵ is a constant with a small value to avoid a zero in the denominator.

The primary advantage of AdaGrad is that it removes the requirement for manual tuning of the learning rate. Usually, a default value of 0.01 is utilized, and no further adjustment is necessary. AdaGrad automatically modifies the learning rates for stochastic gradient descent and online learning on a feature-by-feature basis. However, AdaGrad's principal weakness is the accumulation of squared gradients in the denominator. Due to the addition of positive terms, the accumulated sum grows during training, resulting in a reduction in the learning rate to the point where it becomes exceptionally small, preventing the algorithm from learning further.

b) RMSprop

RMSprop is a widely-used optimization algorithm in neural networks for gradient descent that stands for Root Mean Square Propagation. Its objective is to adjust the learning rate of each weight parameter by taking into account the historical average of the squared gradients for that weight. The calculation of the learning rate for each parameter in RMSProp can be seen as an extension of AdaGrad as it substitutes a decaying average or moving average of the partial derivatives for the sum of partial derivatives seen in AdaGrad.

The algorithm works by keeping track of a running average of the squared gradients $E[g_{t,i}^2]$ for each weight parameter. This running average is used to normalize the gradient at each iteration, which helps prevent oscillations and divergence in the optimization process. Specifically, RMSprop updates the weight parameters using the following formula:

$$\mathbb{E}[g_{t,i}^2] = \beta\mathbb{E}[g_{t-1,i}^2] + (1-\beta)g_{t,i}^2$$

$$w_{t,i} = w_{t-1,i} - \frac{\eta}{\sqrt{\mathbb{E}[g^2_{t-1,i}] + \epsilon}} * \frac{\partial E}{\partial w_{t,i}}$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{\mathbb{E}[g^2_{t-1}] + \epsilon}} * \frac{\partial E}{\partial w_t}$$

where, w_i is the weight of the synapse i, η is the learning rate and $E[g^2]$ is moving average of squared gradient and E is the error function.The ϵ term is added for numerical stability and prevents division by zero.

c) Adam

Adam (Adaptive Moment Estimation) is an optimization algorithm that is commonly used in stochastic gradient descent (SGD) for training deep neural networks. Adam is a combination of two other optimization techniques, namely, RMSprop and momentum.

Adam utilises the estimates of first and second moment of gradients: the first is the exponentially decaying average of the gradient itself, and the second is the exponentially decaying average of the squared gradient.

At each iteration of the training process, Adam computes an effective learning rate for each weight parameter. This effective learning rate is a function of the running averages of the gradient and the squared gradient, and is normalized by a factor that accounts for the initial values of these moving averages.

If $f(W)$ is a noisy objective function, the goal of ADAM is to minimize the expected value of $f(W)$ with respect to parameter W. Let $g_t \epsilon \partial f(w_t)$. m_t is the exponential decaying moving average of the gradient (first moment).

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$$

v_t is the exponential decaying moving average of the squared gradient (second moment).

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$$

where β_1 and β_2 are hyperparameters $\epsilon[0, 1)$. m_t and v_t are biased towards zero during the initial steps and when decay rates are small. To bias correct the two moments, we perform:

$$\hat{m}_t = m_t/(1 - \beta_1^t)$$

$$\hat{v}_t = v_t/(1 - \beta_2^t)$$

The Adam update rule then looks like this:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

where m and v are the moving averages of the gradient and squared gradient, respectively, $\hat{m}$ and $\hat{v}$ are the bias-corrected first and second moment estimates, w is the weight parameter, η is the learning rate, ϵ is a small constant added for numerical stability, E is the error function and t is the current iteration number.

In summary, Adam is an algorithm that combines the advantages of both RMSprop and momentum techniques to optimize the gradient descent process in deep learning. It is effective in dealing with noisy or sparse gradients, and can converge to a good solution more quickly than other optimization algorithms.

1.4 Scope of work

We implement an algorithm that operates on a neuron which receives input spikes and produces output spikes, with the goal of training this neuron to learn a transformation between its input and output spikes.The nature of this transformation is not specified or assumed, as theoretically it is not yet fully understood although attempts have been made in Ramaswamy & Banerjee (2014).The significance of this project lies in the ability of the neuron to reliably produce output spike trains from input spike trains, which is essential for conveying information without loss of any kind to downstream neurons. We implement the algorithm proposed in Banerjee(2016) that can accurately map patterns of input spikes to output spikes in a multi-layered feed-forward network of deterministic spiking neurons, while ensuring that the learning rule is completely spike-based and does not involve mean rates or probabilistic models. We identify the

problems faced during implementation of the learning rule and address these problems using momentum and adaptive gradient methods.

2. Materials and Methods

This section provides an overview of the neuron model adopted in the algorithm called the spike response model (SRM) and the error function that measures the disparity between actual output and desired output spike trains. Next, we compute the gradients of the error functional with respect to the weights of synapses through perturbation analysis and chain rule. The witness based evaluation method used to evaluate the efficacy of the learning rule is discussed in detail. At the end we describe and compare the adaptive gradient optimization methods AdaGrad, RMSProp and Adam.

2.1. Spike Response Model

The Spike Response Model (SRM) (Gerstner et al., 1993) is a simplification of the Hodgkin and Huxley equations (Hodgkin & Huxley, 1952) by reduction of parameters to a single variable, the membrane potential $u(t)$. The SRM views spiking as a process that involves setting a threshold value ν for the membrane potential. If the potential exceeds this value from below, an action potential is triggered. This phenomenon can be represented by the conditions:

$$u(t^{(f)}) = \nu \quad \text{and} \quad \frac{d}{dt}u(t^{(f)}) > 0$$

where $t^{(f)}$ signifies the firing time of the neuron. Assuming that the input current remains in a biologically realistic range, the potential trajectory of the neuron during a spike is predictable. Following the last firing time of the neuron, denoted as $\hat{t}$, for $t > \hat{t}$,

the voltage trajectory spikes and resets to resting potential. Therefore, for $t > \hat{t}$, the membrane potential can be expressed as $u(t) = \eta(t - \hat{t}) + u_{rest}$, where η is the typical shape of a spike and u_{rest} is the resting membrane potential of the neuron. Devoid of input, u approach the u_{rest} and $\eta(t - \hat{t}) \to 0$ for $t - \hat{t} \to \infty$.

If at $t' > \hat{t}$, an additional input current I is applied to the neuron. The membrane potential u is perturbed by a impulse response function or postsynaptic potential function, $\varepsilon(t - \hat{t})$.

Therefore for $t' > t$,

$$u(t') = \eta(t' - \hat{t}) + \int_0^{(t'-\hat{t})} \varepsilon(t' - \hat{t}, s) I(t' - s) ds + u_{rest} \tag{1}$$

This equation is called SRM.

In networks of neurons, the input generally comes in the form of spikes produced by other presynaptic neurons. Let's assume that a presynaptic neuron releases a spike at time t^f and this generates a current input $I(t) = \alpha(t - t^f)$ for $t > t^f$ in a postsynaptic neuron i. Here, $\alpha(t - t^f)$ is a function which describes how the post synaptic potential evolves with time. As a result of this, the voltage of the postsynaptic neuron changes, as given by equation (2):

$$\Delta u_i(t) = \int_0^{t-\hat{t}_i} \varepsilon(t - \hat{t}_i, s) \alpha(t - t_j^{(f)} - s) ds \tag{2}$$

Where, $\hat{t}_i$ is the last output spike time of neuron i.

We define

$$\varepsilon(t - \hat{t}_i, t - t_j^{(f)}) = \Delta u_i(t) = \int_0^{t-\hat{t}_i} \varepsilon(t - \hat{t}_i, s) \alpha(t - t_j^{(f)} - s) ds \tag{3}$$

Suppose that the postsynaptic neuron had its last output spike a while ago. The response in voltage, denoted as $\varepsilon(\infty, t - t_j^{(f)})$, represents the effect of the presynaptic neuron j's firing on neuron i's potential.

The function ε is used to depict how the system reacts over time in response to an incoming spike. When the effects of multiple incoming spikes are added up, and the

resulting potential of neuron i reaches the threshold value, denoted as ν, it will generate an output spike. The shape of the output spike, which returns below the resting membrane potential value after the pulse, is described by a function η . After firing the spike at $\hat{t}_i$, u_i evolves at:

$$u_i(t) = \eta_i(t - \hat{t}_i) + \sum_{j \epsilon \tau_i} w_{ij} \sum_{t_j^{(f)} \epsilon F_j} \varepsilon_{ij}(t - \hat{t}_i, t - t_j^{(f)}) \quad (4)$$

where $\hat{t}_i$ is the last spike of neuron i, $t_j^{(f)}$ is the spikes of presynaptic neurons j, w_{ij} is the synaptic efficacy. $\tau_i = \{j | j$ presynaptic to $i\}$ and F_j is the set of all firing times $t_j^{(f)} < t$ of neuron j. This has three components:

1. η kernel

   $$\eta(t) = -A \times e^{\frac{-t}{\tau_m}} \times \mathcal{H}(t)$$

   The response kernel η represents the typical shape of an action potential and the spike after potential that follows a spike. Each time u reaches the threshold ν , η is added to u. Our choice of η, adopted from the original paper Banerjee(2016), is given by:

   (5)

   where, A is the maximum potential drop during an after spike reset, τ_m is the rate of decay of after hyperpolarization potential and $\mathcal{H}(t)$ is the Heaviside step function, $\mathcal{H}(t) = 1$ for $t > 0$ and 0 everywhere else.

2. ε kernel

   ε kernel is the linear response of the membrane potential to input spikes. Biologically, after a spike spike occurs, ion channels open and membrane resistance is $\varepsilon(t) = \frac{1}{\alpha\sqrt{(t)}} \times e^{\frac{-\beta\alpha^2}{t}} \times e^{\frac{-t}{\tau_c}}$ reduced and the neuron enters a refractory state. This is incorporated into the ε kernel by the dependence on $t - \hat{t}_i$. ε_{ij} is called excitatory postsynaptic potential if the synapse from j to i is excitatory and inhibitory postsynaptic potential if the synapse from j to i is inhibitory. The ε function is given by:

   (6)

where, α, a dimensionless quantity is chosen to replicate the distance of the synapse from the soma, β, another dimensionless quantity describes the rate of rise of postsynaptic potential. τ_c is the rate of decay of PSP.

3. Threshold ν

   Threshold is defined as a free parameter. For the purpose of our algorithm, we adopt hard thresholding.

Parameter	**Value of parameter**
Number of synapses	8 or 15
Number of output neurons	1
ν - threshold	-$20 * 10^{-3}$ V
τ_{ce}- excitatory synaptic time constant	$20 * 10^{-3}$ s
τ_{ci}- inhibitory synaptic time constant	$10 * 10^{-3}$ s
τ_m- membrane potential constant	$1.2 * 10^{-3}$ s
A- drop in potential after a spike	$400 * 10^{-3}$ V
γ- simulation window size	$500 * 10^{-3}$ s
Γ	$150 * 10^{-3}$ s
α_{exc}-dimensionless constant	1.5
α_{inh}-dimensionless constant	1.2
β- dimensionless constant	1

Table 2.1 : The parameters used in implementation of spike response model

2.2 Error functional

We use the error functional described in Banerjee(2016) as a cost function for training the SNN. It fulfills the following requirements:

1. The functional should focus more on the recent spikes than the past ones. If the spikes are misaligned in the far past but more closely aligned in the recent past, we want the effect of the far past to be diminished in our error function. We achieve this by defining a parameter γ that acts as a bound and spikes that have aged beyond it have no contribution to the present membrane potential of the neuron. Γ ensures a temporal asymmetry reflects the exponential decay of all PSPs and AHPs towards the resting membrane potential.
2. The function must be capable of explicitly depicting how a spike train impacts the membrane potential of a neuron.
3. The functional should be easy to manipulate, and its derivatives should be obtained in closed form.

We start with a parametric function that resembles PSP of the neuron:

$$f_{\beta,\tau}(t) = 1/\tau e^{-\beta/t} e^{-t/\tau} \tag{7}$$

Let O be the finite vector of output spike times and D be the finite vector of desired spike times.

The impact of the output spike train on the membrane potential can be described as:

$$\sum_{i=1}^{N} f_{\beta,\tau}(t_i^O) \tag{8}$$

The measure of disparity is therefore defined as the square of the difference in the impact of the output spike train and the desired spike train in the membrane potential of a neuron:

$$\left(\sum_{i=1}^{N} f_{\beta,\tau}(t_i^D) - \sum_{i=1}^{N} f_{\beta,\tau}(t_i^O) \right)^2 \tag{9}$$

To remove the dependence on the parameters β and τ, we integrate the measure over all the values of β and τ. However, since, integrating τ over 0 to ∞ will make the effect of the spikes aged past γ significant, we choose a value T significantly greater

than γ such that the effect of the spike aged γ will have negligible impact on the potential.

$$E(t^D, t^O) = \int_0^T \int_0^\infty \left(\sum_{i=1}^{N} f_{\beta,\tau}(t_i^D) - \sum_{i=1}^{N} f_{\beta,\tau}(t_i^O) \right)^2 d\beta d\tau$$

$$= \sum_{i,j=1}^{M,M} \frac{t_i^D * t_j^D}{(t_i^D + t_j^D)^2} \quad e^{-\frac{t_i^D + t_j^D}{T}} + \sum_{i,j=1}^{N,N} \frac{t_i^O * t_j^O}{(t_i^O + t_j^O)^2} \quad e^{-\frac{t_i^O + t_j^O}{T}} - 2 \sum_{i,j=1}^{M,n} \frac{t_i^D * t_j^O}{(t_i^D + t_j^O)^2} \quad e^{-\frac{t_i^D + t_j^O}{T}} \tag{10}$$

Taking a partial derivative of E with respect to the output spike times:

$$\frac{\partial E}{\partial t_i^O} = 2 \left(\sum_{j=1}^{N} \frac{t_j^O((t_j^O - t_i^O) - \frac{t_i^O}{T}(t_j^O + t_i^O))}{(t_j^O + t_i^O)^3} \quad e^{-\frac{(t_j^O + t_i^O)}{T}} - \sum_{j=1}^{M} \frac{t_j^D((t_j^D - t_i^O) - \frac{t_i^O}{T}(t_j^D + t_i^O))}{(t_j^D + t_i^O)^3} \quad e^{-\frac{(t_j^D + t_i^O)}{T}} \right) \tag{11}$$

2.3 Perturbation Analysis

To derive the partial derivatives of E with respect to the input spike times and input weights we perform a perturbation analysis. Let us assume the neuron produces its l^{th} output spike at time t_l^O. The state of the neuron at the time can be described as:

$$\Theta = \sum_{i \epsilon \Gamma} \sum_{j \epsilon F_i} (w_{i,j}) \varepsilon_i (t_{i,j}^I - t_l^O) + \sum_{k \epsilon F} \eta (t_k^O - t_l^O) \tag{12}$$

.

If we perturb the input spike times by $\Delta t_{i,j}^I$ and input weights by $\Delta w_{i,j}$, the neuron spikes at $t_l^O + \Delta t_l^O$. The state of the neuron can then be described as:

$$\Theta = \sum_{i \epsilon \Gamma} \sum_{j \epsilon F_i} (w_{i,j} + \Delta w_{i,j}) \varepsilon_i (t_{i,j}^I + \Delta t_{i,j} - t_l^O - \Delta t_l^O) + \sum_{k \epsilon F} \eta (t_k^O + \Delta t_k^O - t_l^O - \Delta t_l^O) \tag{13}$$

.

From equation (13), we can derive:

$$\Delta t_l^O = \frac{\sum_{i \epsilon \Gamma} \sum_{j \epsilon F_i} \Delta w_{i,j} \varepsilon_i (t_{i,j}^I - t_l^O) + \sum_{i \epsilon \Gamma} \sum_{j \epsilon F_i} w_{i,j} \frac{\partial \varepsilon_i}{\partial t}|_{(t_{i,j}^I - t_l^O)} \Delta t_{i,j}^I + \sum_{k \epsilon F} \frac{\partial \eta}{\partial t}|_{(t_k^O - t_l^O)} \Delta t_k^O}{\sum_{i \epsilon \Gamma} \sum_{j \epsilon F_i} w_{i,j} \frac{\partial \varepsilon_i}{\partial t}|_{(t_{i,j}^I - t_l^O)} + \sum_{k \epsilon F} \frac{\partial \eta}{\partial t}|_{(t_k^O - t_l^O)}} \tag{14}$$

$$\frac{\partial t_l^O}{\partial t_{i,j}} = \frac{w_{i,j}\frac{\partial \varepsilon_i}{\partial t}|_{(t_{i,j}^I - t_l^O)} + \sum_{k\epsilon F}\frac{\partial \eta}{\partial t}|_{(t_k^O - t_l^O)}\frac{\partial t_k^O}{\partial t_{i,j}}}{\sum_{i\epsilon\Gamma}\sum_{j\epsilon F_i} w_{i,j}\frac{\partial \varepsilon_i}{\partial t}|_{(t_{i,j}^I - t_l^O)} + \sum_{k\epsilon F}\frac{\partial \eta}{\partial t}|_{(t_k^O - t_l^O)}} \tag{15}$$

(16)

2.4 Gradient Descent

$$w_{i,j} \leftarrow w_{i,j} - \mu\frac{\partial E}{\partial w_{i,j}} \tag{17}$$

The expression presented is a typical update using gradient descent. However, rather than updating the weights in the past, the approach advocated in Banerjee (2016) involves a delayed update. In this approach, the weight at synapse i is updated based on the contributions of a finite number of past spikes.The below expression describes the gradient descent update that we have used:

$$w_i \quad \leftarrow \quad w_i - \sum_{j\epsilon\mathcal{F}_i}\mu\frac{\partial E}{\partial w_{i,j}} \tag{18}$$

$\frac{\partial E}{\partial w_{i,j}}$ is obtained by chain rule.

$$\frac{\partial E}{\partial w_{i,j}} = \sum_{k\epsilon F}\frac{\partial E}{\partial t_k^O}\frac{\partial t_k^O}{\partial w_{i,j}} \tag{19}$$

2.5 Experimental Validation

$$\frac{\partial t_l^O}{\partial w_{i,j}} = \frac{\varepsilon_i(t_{i,j}^I - t_l^O) + \sum_{k\epsilon F}\frac{\partial \eta}{\partial t}|_{(t_k^O - t_l^O)}\frac{\partial t_k^O}{\partial w_{i,j}}}{\sum_{i\epsilon\Gamma}\sum_{j\epsilon F_i} w_{i,j}\frac{\partial \varepsilon_i}{\partial t}|_{(t_{i,j}^I - t_l^O)} + \sum_{k\epsilon F}\frac{\partial \eta}{\partial t}|_{(t_k^O - t_l^O)}}$$

2.5.1 Witness-based evaluation framework

We use a conservative evaluation criterion derived from witness-based evaluation framework to evaluate the results because:

1. The state of the current knowledge on which transformations can be carried out by a feedforward network of spiking neurons with a particular architecture and complexity is very limited.
2. We don't make assumptions about what the desired spike train is supposed to look like.

According to the evaluation framework,

- We generate a witness network of a particular architecture with weights randomly assigned and later fixed.
- We drive the network with input spike trains generated by the Poisson process.
- We record the precise input and output spike trains
- We generate learning networks of the same architecture initialized with random synaptic weights. Can the learning networks learn the input-output transformation of the witness network by using the synaptic weight update rule proposed above?

2.5.2 Mean Absolute Percentage Error

Instead of examining whether the learning network has learned the transformation, we check if the weights of the learning network have converged to that of the witness network. We use a conservative criterion called Mean Absolute Percentage Error(MAPE).

$$MAPE = \frac{100\%}{n} \sum_{t=1}^{n} \left| \frac{CorrectWeight - LearnedWeight}{CorrectWeight} \right| \quad (20)$$

where: n is the number of synapses, CorrectWeight is the weight of the witness network synapse, LearnedWeight is the weight of the learning network synapse. A MAPE of 1 or 100% shows convergence of the learning network weights to the witness network weights.

2.6 Momentum

Require :

Parameter	**Value of parameter**
η - step size	0.01
β - decay rate of first moment	0.9

Table 2.2 : Parameters for gradient descent with momentum

Require : $E(w_{t,i})$ The error function at time t for neuron i

Initialise : w_0 - initial parameter value

$g_t \leftarrow$ 0 initial derivative of error function wrt w

$b_t \leftarrow$ 0 initial momentum vector

$t \leftarrow$ 0 timestep

while w_t not converged **do**:

$t \leftarrow t + 1$ (update timestep)

$g_t \leftarrow \frac{\partial E}{\partial w_t}$ (update derivative)

$b_t \leftarrow \beta b_{t-1} + g_t$ (update the momentum vector)

$w_{t,i} \leftarrow w_{t-1,i} - \eta b_t$ (update parameters)

end while

return w_t

2.7 Adaptive gradient Methods

2.7.1 AdaGrad

Require :

Parameter	**Value of parameter**
η - step size	0.01
ε - constant for numerical stability	10^{-8}

Table 2.3 : **Parameters used in Adagrad**

Require : $E(w_{t,i})$ The error function at time t for neuron i

Initialise : w_0 - initial parameter value

$g_t \leftarrow$ 0 initial derivative of error function wrt w

$G_0 \leftarrow$ 0 initial outer product matrix

$t \leftarrow$ 0 timestep

while w_t not converged **do**:

$t \leftarrow t + 1$ (update timestep)

$g_t \leftarrow \frac{\partial E}{\partial w_t}$ (update derivative)

$G_t \leftarrow \sum_{\tau=1}^{t} g_\tau g_\tau^T$ (update the outer product)

$w_{t,i} \leftarrow w_{t-1,i} - g_{t,i}\frac{\eta}{\sqrt{G_{t,ii} + \epsilon}}$ (update parameters)

end while

return w_t

2.7.2 RMSProp

Require :

Parameter	**Value of parameter**
η - learning rate	0.01
β - decay rate of second moment	0.9

Table 2.4: **Parameters used in RMSprop**

Require : $E(w_{t,i})$ The error function at time t for neuron i

Initialise : w_0 - initial parameter value

$g_t \leftarrow$ 0 initial derivative of error function wrt w

$E[g^2]0 \leftarrow$ 0 initial moving average of squared gradients

$t \leftarrow$ 0 timestep

while $w(t)$ not converged **do**:

$E[g^2](t) \leftarrow \beta E[g^2](t-1) + (1-\beta) * (\frac{\partial E}{\partial w})^2$ (update decaying moving average of squared gradients)

$w_{t+1} \leftarrow w_t - \frac{\eta}{\sqrt{E[g_t^2] + \epsilon}} * \frac{\partial E}{\partial w_{t+1}}$ (update parameters)

end while

return w_t

2.7.3 Adam

Require :

Parameter	**Value of parameter**
η - step size	0.01
β_1 - decay rate of first moment	0.9
β_2 - decay rate of second moment	0.999
ε - constant for numerical stability	10^{-8}

Table 2.5 : **Parameters used in Adam**

Require : $E(w_{t,i})$ The error function at time t for neuron i

Initialise : w_0 - initial parameter value

$m_0 \leftarrow$ 0 the first moment vector

$v_0 \leftarrow$ 0 the second moment vector

$t \leftarrow$ 0 timestep

while w_t not converged **do**:

$t \leftarrow t + 1$ (update time step)

$g_t \leftarrow \partial E_t / \partial w_{t-1}$ (update the derivative of error function wrt w)

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ (update the first moment)

$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ (update the second moment)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (calculate bias corrected first moment)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (calculate bias corrected second moment)

$w_t \leftarrow w_{t-1} - \eta . \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$ (update parameters)

end while

return w_t

2.8 Programming and Hardware

All learning experiments were run using custom built code in Python with the Pytorch library. Few of them were run on CPU and a few on GPU. The code was run on the Brain server at Department of COmputer Science, BITS Pilani(Hyderabad campus). Specifications: CPU: 32 processors of AMD Ryzen Threadripper Pro 3955WX with 16 CPU cores each and 512 GB RAM. GPU: 2 NVIDIA GeForce RTX 3090 GPU cards with 10496 CUDA cores and 24 GB memory each.

3. RESULTS

3.1 Homogeneous Poisson input spike train

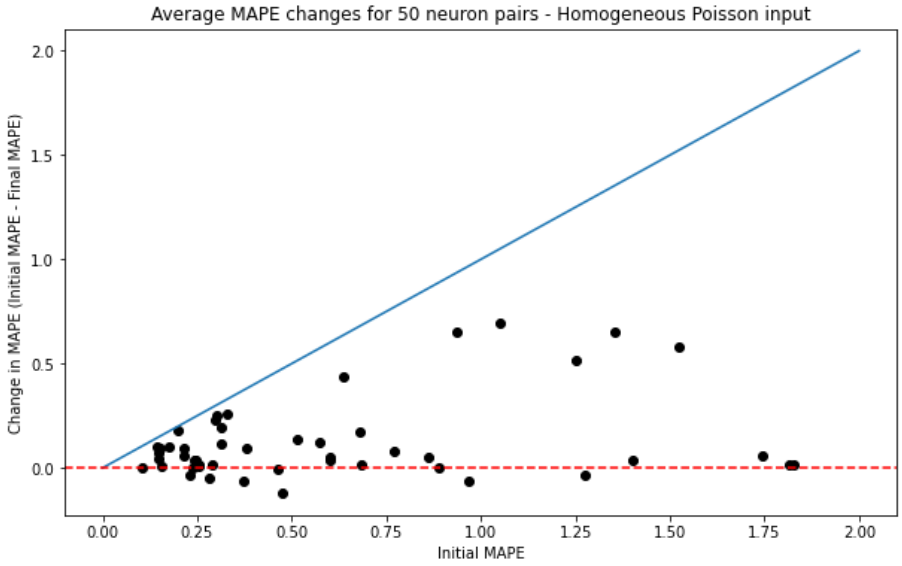

Figure 3.1 : Scatter plot depicting the initial MAPE versus change in MAPE for 50 witness-learning neuron pairs driven by homogeneous Poisson input spike train after 1000 gradient descent steps. Each neuron has 15 excitatory synapses each. The learning rate and the cap is not constant across each pair.

To assess the overall effectiveness of the algorithm in learning the spike train to spike train transformation, we created 50 pairs of neurons, each with 15 excitatory synapses that were randomly assigned weights. These pairs were then given a 10 Hz

homogeneous Poisson input spike train, and one neuron of the pair was subjected to 1000 gradient descent updates with a small learning rate and cap. We measured the disparity between the initial and final performance of the learning neuron compared to its corresponding witness neuron, and plotted this as a scatter plot in Figure 3.1.
Out of 50 learning neurons, 41 neurons showed convergence towards its corresponding witness neuron. This corresponds to 83.67% (exact binomial test, p = 2.81*10^-6) of the neuron pairs showing improvement in MAPE.

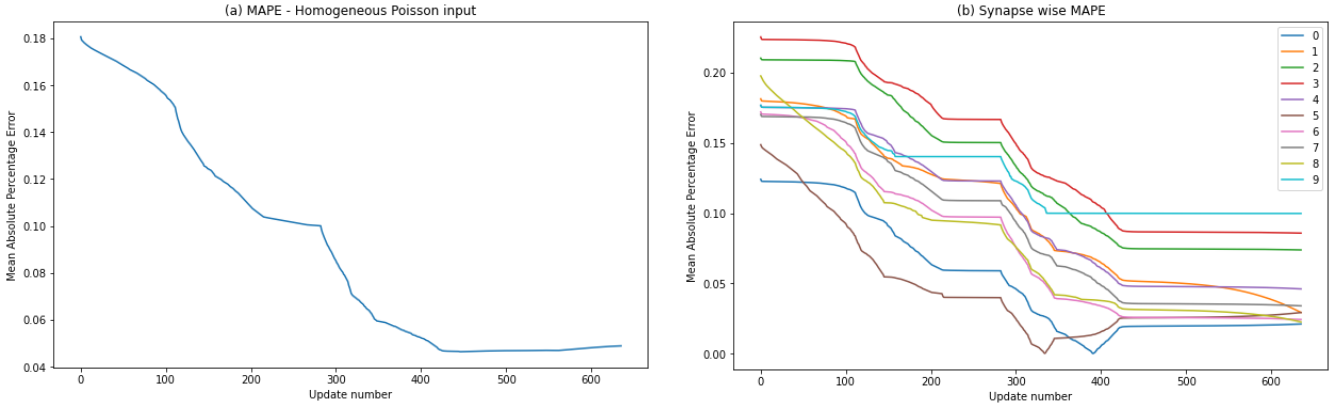

Figure 3.2 : A witness-learning neuron pair with 10 synapses each were driven with a homogeneous Poisson spike train and driven until convergence. (a) depicts the evolution of MAPE with the gradient descent on the weights. 74.08% convergence was achieved in 630 gradient descent steps. (b) depicts the synapse-wise MAPE versus the update number.

We ran a few pairs of witness-learning neuron pairs until convergence and were able to achieve a maximum of 70-75% convergence. One such pair is depicted in Figure 3.2. We gave the same homogeneous Poisson spike train as input to both neurons in a witness learning neuron pair. The convergence of the weights of the learning network towards the weights of the witness network is plotted as a function of the number of gradient descent steps. We were able to achieve 73% convergence in 630 gradient descent steps. The initial average MAPE was 0.1806 and the average MAPE after 630

gradient descent steps is 0.0489. The synapse wise MAPE shows that all the synapses were converging at the beginning but learning slows down and stagnates after 400 gradient descent steps.

3.2 Inhomogeneous Poisson input spike train

To create diversity in the input spike trains, we created 50 pairs of neurons, each with 15 excitatory synapses that were randomly assigned weights. These pairs were then given an inhomogeneous Poisson input spike train where the frequency of spikes varied sinusoidally between 5 Hz and 10 Hz at a frequency of 2 Hz. The learning neuron underwent 1000 updates through gradient descent, with a small learning rate and cap. The change in the MAPE of the learning neuron compared to its corresponding witness neuron was calculated and plotted as a scatter plot in Figure 3.3.

Out of the 50 learning neurons, 43 of them showed improvement in MAPE. This corresponds to 86% (exact binomial test, p = 5.81*10^-6) of witness-learning neuron pairs converging, which is marginally more than the percentage of pairs converging when driven by homogeneous poisson input.

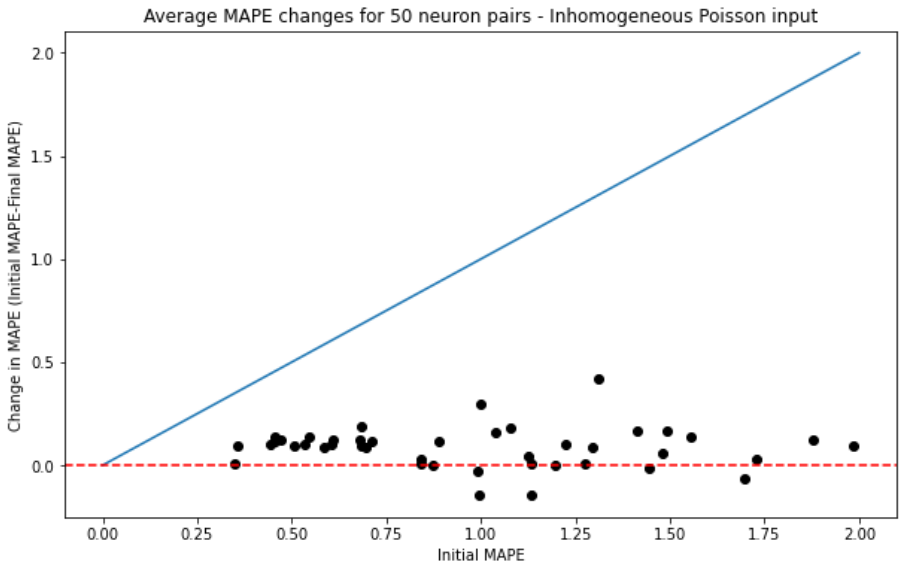

Figure 3.3 : Scatter plot depicting the initial MAPE versus change in MAPE for 50 witness-learning neuron pairs driven by inhomogeneous Poisson spike train after 1000 gradient descent steps. Each neuron has 15 excitatory synapses each. The learning rate and the cap is not constant across each pair.

We ran gradient descent on a few learning-witness neuron pairs until convergence. We observed 80-85% convergence in most of the pairs. The hindrance in reaching 100% convergence can be explained by the example below.

The witness-learning neuron pair in Figure 3.4(a) has been run for more than 1075 gradient descent steps. The gradient descent was stopped when the learning slowed down considerably. The initial average MAPE was 4.4481 and the average MAPE after 1075 gradient descent steps is 0.8191. This example shows a percent MAPE convergence of 81.6%.

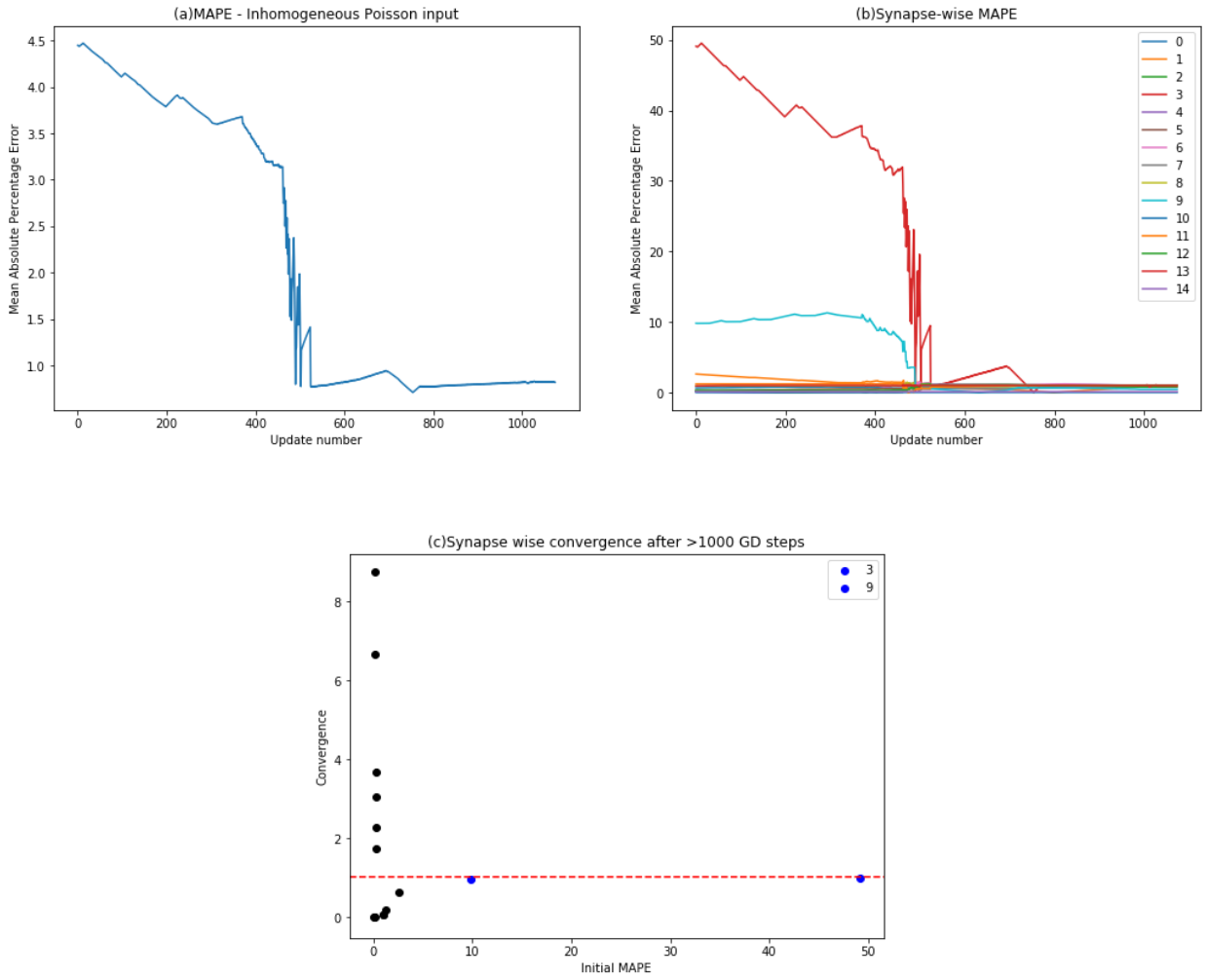

Figure 3.4 : A witness-learning neuron pair with 15 synapses each was driven with an inhomogeneous Poisson spike train and driven until 81.6% convergence. (a) depicts the evolution of MAPE with the gradient descent on the weights.81.6% convergence was achieved in 1075 gradient descent steps. (b) depicts the synapse-wise MAPE versus the update number. Notice that synapses 3 and 9 with a higher initial MAPE converge faster compared to other synapses. This is also depicted in (c) which shows the convergence(change in MAPE/initial MAPE) after 1075 gradient descent steps versus the initial MAPE for each synapse.

However, when we observe the synapse-wise MAPE trajectory of the learning neuron, we can conclude that only two synapses i.e, synapse 3 and synapse 9, have shown

more than 95% convergence (98.15% and 95.40% respectively). Other synapses are either diverging or converging very slowly towards the desired weights as can be observed from Figure 3.4(b). The two synapses that showed rapid convergence had a very high initial MAPE error(49.1308 and 9.8124 respectively). This trend has been observed in other cases as well where synapses with very high initial MAPE converged faster at the cost of other synapses.

3.3 Effect of inhibitory synapses

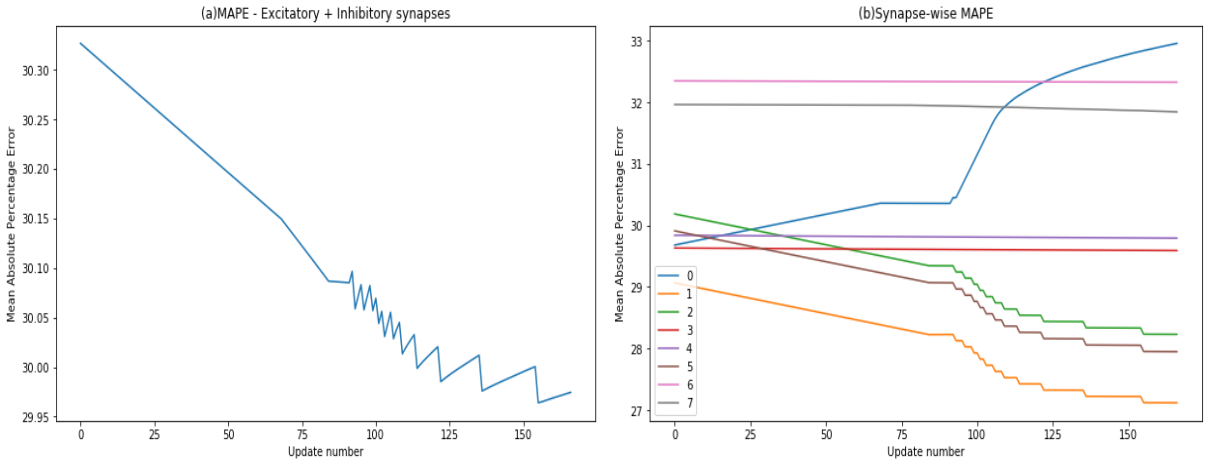

Figure 3.5 : A witness-learning neuron pair with 8 synapses in total, 6 of which are excitatory and 2 are inhibitory. (a) depicts how the MAPE changes or the learning weight converges to the witness weight. The learning started from a very high MAPE value and therefore, in 150 gradient descent steps shows very low convergence. The rate of convergence at 150 updates is around 0.21 per 100 gradient descent steps. (b) shows the synapse-wise MAPE versus the update number. The pink(6) and gray (7) traces are the inhibitory neurons.

We created a witness learning neuron pair with both excitatory and inhibitory synapses to see how the weight update rule influences the convergence of inhibitory weights. The neurons were driven by 6 excitatory synapses and 2 inhibitory synapses. The learning neuron's synaptic weights converged 1.16% in 150 gradient descent updates. The

synapse wise MAPE changes are quite interesting. The synapse-wise MAPEs of the two inhibitory synapses did not change at all during gradient descent i.e, the inhibitory synapses are silent.

The silent inhibitory synapses can be explained using the nature of the inhibitory PSPs. As discussed in Methods, $\frac{\partial E}{\partial w}$ is influenced by a key factor, $\frac{\partial t}{\partial w}$ given by the below equation

$$\frac{\partial E}{\partial w_{i,j}} = \sum_{k \epsilon F} \frac{\partial E}{\partial t_k^O} \frac{\partial t_k^O}{\partial w_{i,j}}$$

This refers to how a small adjustment in weight would affect the timing of output spikes. This factor is determined by the gradient of the postsynaptic potential (PSP) generated by the inhibitory neuron. The output spikes occur during the ascending phase of the inhibitory PSP , which has a much lower derivative than the ascending phase of an excitatory PSP. Therefore the value of $\frac{\partial E}{\partial w}$ is low and therefore learning in those synapses is very low.

3.4 Effect of simulation window size γ

The hyperparameter γ In our synaptic weight update rule, the window size within which the input spikes to and output spikes from a neuron is significant to the membrane potential of the neuron. Biologically, the parameter γ can be explained as follows: Once they initially increase or decrease, both postsynaptic potentials (PSPs) and afterhyperpolarization (AHPs) decay rapidly to the resting potential, in an exponential manner. During the absolute refractory period r, regardless of the strength of incoming stimuli, the neuron is unable to fire another action potential. $n_i = [\Gamma / r_i]$ is the The maximum limit on the number of effective spikes that neuron i can possess at any particular moment (Banerjee, 2001). This shows that $\mathcal{F}_i$ and $\mathcal{F}$ are both finite sets of afferent and efferent spikes respectively.

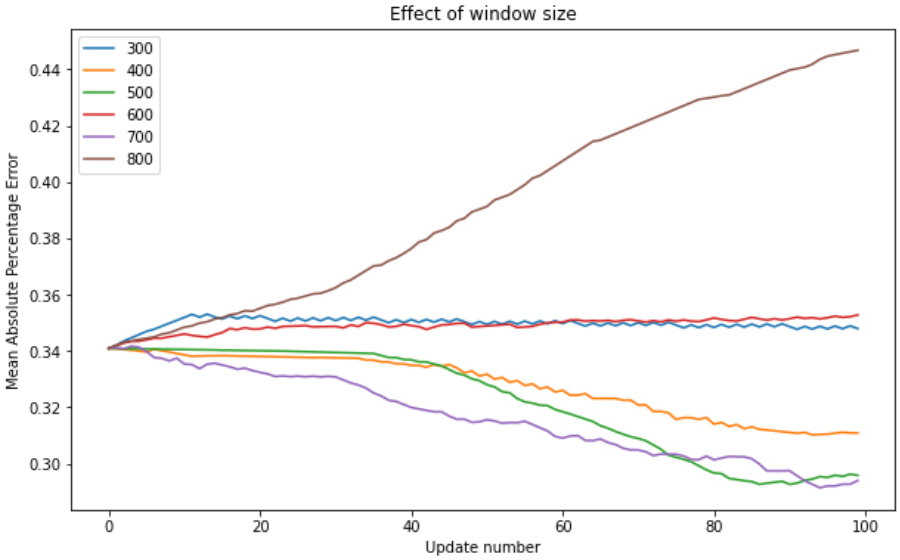

Figure 3.6 : MAPE convergence for different values of the window size γ

Optimizing the value of γ is important for the weight update rule. Having a higher γ seems to speed up the learning process and a lower γ shows very slow learning. Logically, having a higher γ would mean a larger number of spikes that contribute to the present membrane potential of the neuron and therefore, higher computation cost and time. However, at an optimum value of γ, the error calculated at a moment in time is not skewed by the low error of the recent past and the high error values of the far past contribute to the calculation therefore, learning does not slow down. However, when the error is high and erratic, there is a higher chance of divergence from the desired values.This is the reason we see a more convergence in the same number of update steps for higher values of γ.

3.5 Effect of momentum

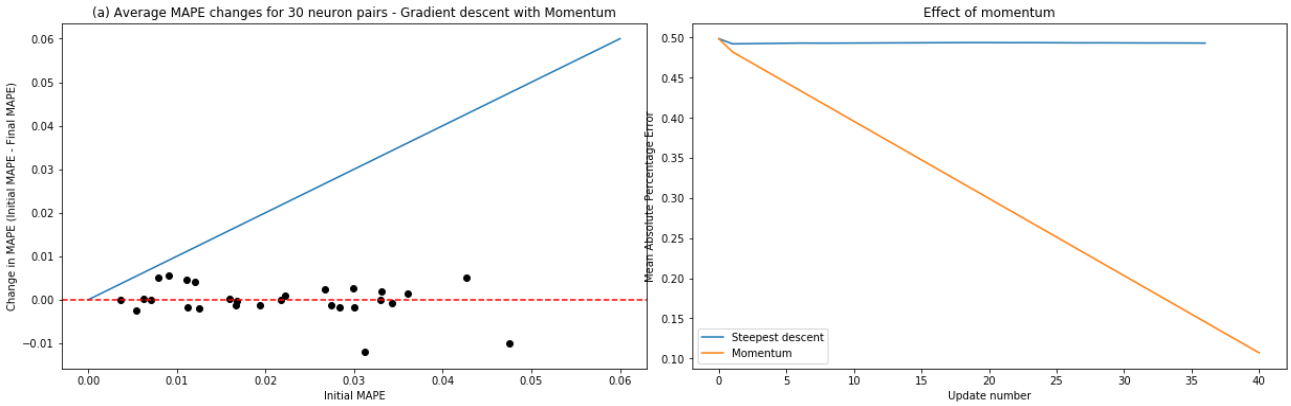

Figure 3.7 : The figure(a) depicts 30 learning neurons subjected to gradient descent with momentum. Scatter plot of initial MAPE versus change in MAPE for these pairs of neurons, (b) depicts the performance of gradient descent with or without momentum. WIthout momentum, the learning is very slow and convergence stagnates.With momentum, learning is accelerated and there is a very fast convergence of learning weights towards the witness weights.

30 pairs of neurons, each with 15 excitatory synapses with randomly assigned weights were created. These pairs were then given an inhomogeneous Poisson input spike train where the frequency of spikes varied sinusoidally between 5 and 10 Hz at a frequency of 2 Hz. The learning neuron was subjected to 500 gradient descent updates with momentum. 12 out of the 30 pairs showed improvement in MAPE value. This corresponds to 43.33% (exact binomial test, p = 0.82) of the neuron pairs showing convergence of learning weights towards witness weights. This is less than expected since momentum accelerates the gradient descent. We speculate that this may be due to the divergence of the weights before reaching the maximum convergence which has been observed while using momentum.

We created a witness learning neuron pair with 15 synapses and the learning neuron was subjected to gradient descent with momentum. The initial MAPE was 0.4985. The final MAPE after 40 gradient descent steps is 0.1069 for momentum compared to 0.4930 for steepest descent algorithm. The convergence percentage after 40 gradient

descent steps is 78.54% for gradient descent with momentum and 1.1014% for gradient without momentum. Momentum shows highly accelerated convergence compared to traditional gradient descent.

3.6 Gradient Descent with different optimisers

We created a witness learning neuron pair with 15 synapses and the learning neuron was subjected to gradient descent with different optimisers. We ran four different variants of gradient descent. First, traditional gradient descent following the steepest descent direction with a fixed learning rate. We observed that the weights of the learning neuron diverged from the witness weights from the start of the learning process. RMSprop also showed the same trend. During gradient descent with AdaGrad, the weights converged in the beginning, but the learning slowed down after a few steps and weights stagnated.

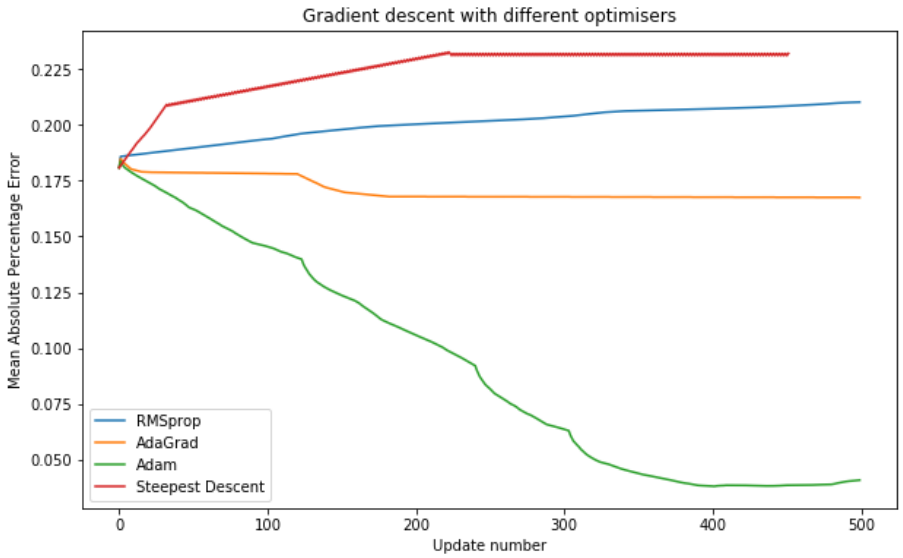

Figure 3.8: Gradient descent - Steepest descent and with different optimisers, AdaGrad, RMSprop, and Adam.

Out of the three optimisers, Adam showed the best results. With Adam, the neuron went from an initial MAPE of 0.1808 to a MAPE of 0.0404 in 502 gradient descent steps, which is equal to 77.67% convergence. Learning slowed down considerably after 500

gradient descent steps. Even though this is not the highest MAPE convergence obtained, achieving such a high convergence in fewer gradient descent steps is commendable. Also taking into consideration, divergence of the traditional gradient descent from the start this result appears significant.

3.7 Effect of RMSProp

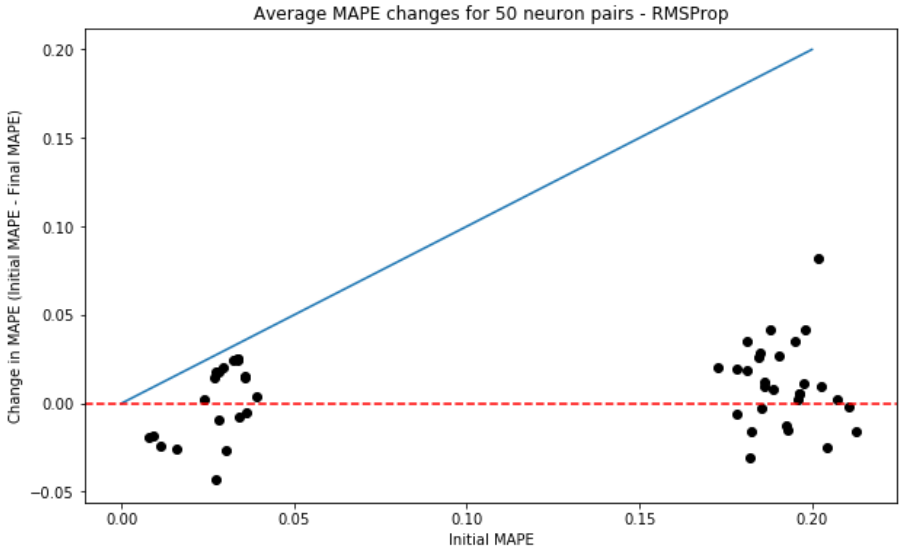

Figure 3.9 : Fifty pairs of witness-learning neurons were subjected to gradient descent with RMSProp. The initial MAPE versus change in MAPE after 100 gradient descent steps is plotted in the figure.

We created 50 witness-learning neurons that received inhomogeneous Poisson input. The learning neuron underwent 100 updates of gradient descent with the RMSProp algorithm. After the 100 steps, 32 of the 50 learning neurons improved their MAPE score, while 6 showed a higher MAPE error than at the beginning. Therefore, 64 % of the learning neurons (exact binomial test, p = 0.032) converged to the weights of their respective witness neurons.

3.8 Effect of AdaGrad

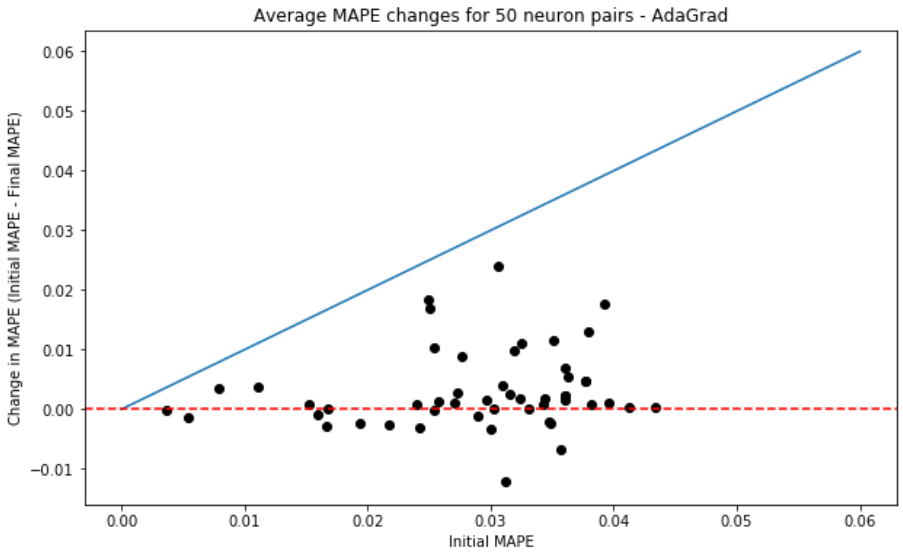

Figure 3.10 : Fifty pairs of witness-learning neurons were subjected to gradient descent with AdaGrad. The initial MAPE versus change in MAPE after 100 gradient descent steps is plotted in the figure.

50 pairs of witness learning neurons were created to receive input from inhomogeneous Poisson sources. Each learning neuron underwent 100 gradient descent updates with the AdaGrad algorithm. After the 100 iterations, 35 of the 50 learning neurons improved their MAPE score, while 15 neurons exhibited a higher MAPE error than at the beginning. As a result, 70% (exact binomial test, p = 0.003) of the learning neurons converged to the weights of their corresponding witness neurons.

3.9 Effect of Adam

We created 50 witness-learning neurons which were driven by inhomogeneous poisson input. The learning neuron underwent gradient descent with Adam for 100 updates. Out of 50 learning neurons, 44 neurons showed improvement in MAPE after 100 gradient descent steps. 6 of them had a higher MAPE error after 100 steps than at the start. This corresponds to 92.5% (exact binomial test, 1.62 *10^-8) learning neurons converging to weights of their witness neurons. This percentage is higher than the values we obtained while employing traditional gradient descent in both homogeneous poisson input case(83.67%) and inhomogeneous poisson input case(86%).

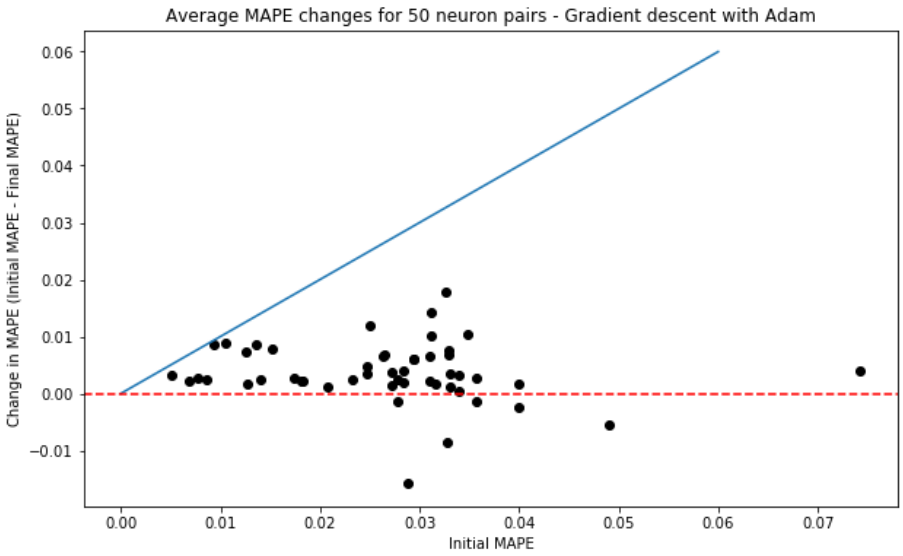

Figure 3.11 : Fifty pairs of witness-learning neurons were subjected to gradient descent with Adam. The initial MAPE versus change in MAPE after 100 gradient descent steps is plotted in the figure.

3.10 Effect of RMSProp, AdaGrad and Adam on inhibitory synapses

We created a witness learning neuron pair with both excitatory and inhibitory synapses to see how the weight update rule with optimisers influences the convergence of inhibitory weights. The neurons were driven by 8 excitatory synapses and 2 inhibitory synapses.While using Adam, the learning neuron's synaptic weights converged 38.82% in 500 gradient descent updates. The synapse wise MAPE changes are quite interesting. The synapse-wise MAPEs of the two inhibitory synapses diverged from their witness weights during gradient descent and one of the synapses was silent.

While using RMSProp, the learning neuron's synaptic weights converged 6% in 500 gradient descent updates. One of the inhibitory synapses remained silent and another one converged to its witness weight. During gradient descent with AdaGrad, we observe a convergence of 15.23% in 500 gradient descent steps. However, the synapse wise MAPE shows that both the inhibitory synapses are converging.

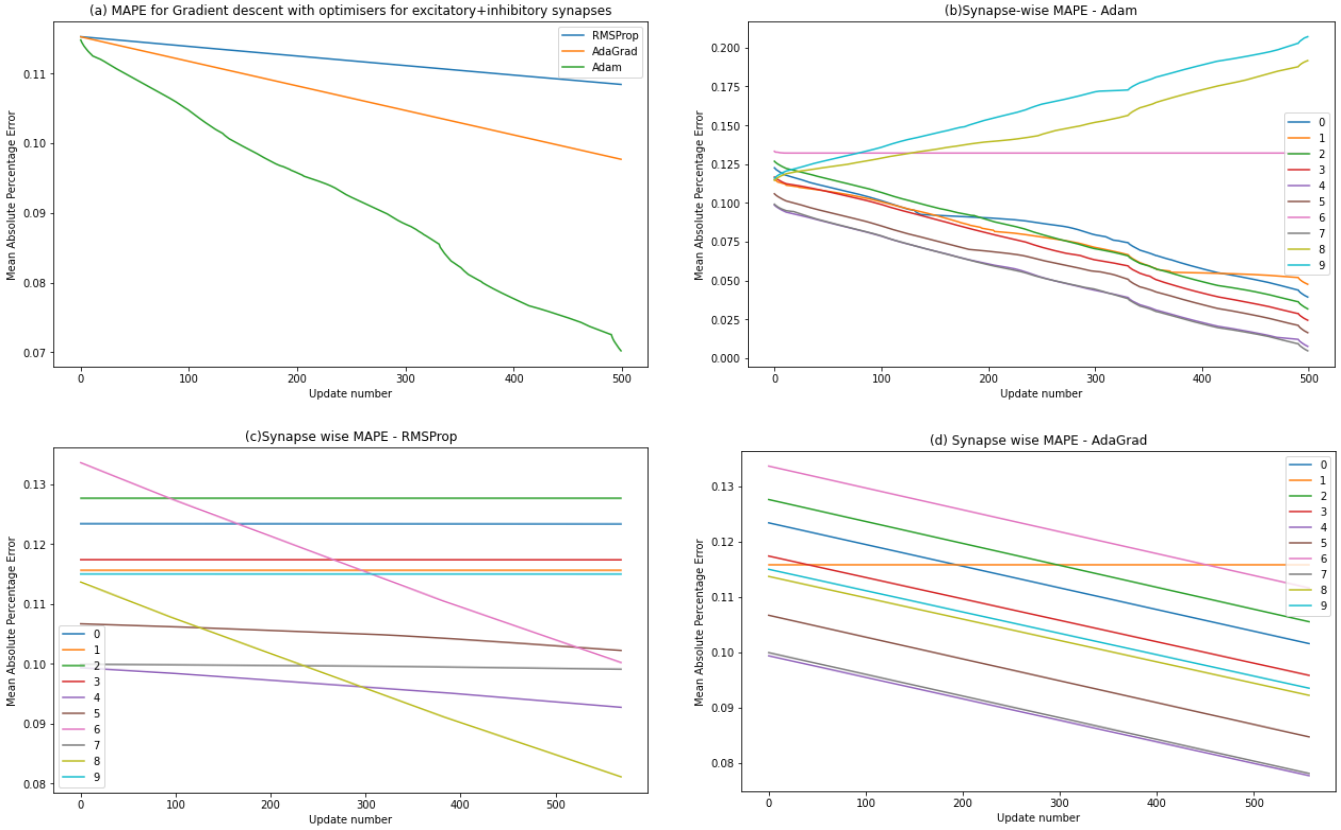

Figure 3.12 : A witness-learning neuron pair with 10 synapses in total, 8 of which are excitatory and 2 are inhibitory. (a) depicts how the MAPE changes or the learning weight converges to the witness weight for three different optimisers RMSProp, AdaGrad, Adam. (b) shows the synapse wise MAPE value for Adam. (c) shows the synapse wise MAPE value for RMSProp.(b) shows the synapse wise MAPE value for AdaGrad. The yellow (8) and cyan (9) traces are the inhibitory neurons.

The code for reproducing the results can be accessed in https://github.com/vaishnavivaliyaparambil/Supervised-Learning-in-Spiking-Neural-Networks

4. DISCUSSION

The synaptic weight learning rule proposed by Banerjee(2016) posits to tackle the spike train to spike train transformation problem. The aim of this study is to re-evaluate the results of Banerjee(2016), recognise the problems faced during implementation and come up with solutions to address the problems. We have been able to implement the synaptic weight update rule, identify key problems namely the silent synapse problem and the problem of slow learning due to a fixed learning rate and a fixed cap on the update tensor. The silent synapse problem is a very difficult problem to solve but the idea on how to try to tackle it using homeostatic plasticity is discussed in this chapter. The problem of learning slowing down due to a fixed learning rate and cap on update tensor has been solved by using momentum and adaptive gradient methods namely, AdaGrad, Adam and RMSProp.

4.1 Summary of findings

To test the effectiveness of the synaptic weight update rule that relies solely on spike time differences, we test its ability to accurately guide the synaptic weights of the learning network towards those of the witness network. When the input spike train is derived from a homogeneous poisson process, out of 50 learning neurons run through 1000 gradient descent steps, 83.67% of them showed convergence towards the synaptic weights of the witness neuron. When the same process was repeated using an inhomogeneous poisson spike train, out of 50 learning neurons, 86% showed convergence after 1000 gradient descent steps. While running individual neuron pairs to convergence we observe that the learning slows down after reaching 80-85% convergence. To identify the reason for this trend, we observed the individual synaptic

MAPE values which showed that out of all the synapses, a few of them remain silent during training because of not getting a rich enough input i.e, the input spike rate at these synapses is very low. We also ran the gradient descent on a learning neuron that had both excitatory and inhibitory synapses and observed that the inhibitory synapses learn very slowly or remain silent when compared to other synapses of the neuron. An important parameter for our synaptic weight update rule is γ. An arbitrary value of γ is chosen for running gradient descent. To see what the effect of various values of γ is on learning, we ran gradient descent for different values of γ and found that there is an optimum value of γ = 700 at which the learning is quicker and steadier. We introduced momentum to gradient descent that accelerated the learning process. Gradient descent was optimised using Adagrad, RMSprop and Adam, out of which Adam accelerated learning and did not diverge during the process. Out of 50 learning neurons run through 500 gradient descent steps with Adam, 92.25% of them showed convergence towards the synaptic weights of the witness neuron.

4.2 Number of gradient descent steps

The data collected is restricted by the number of gradient descent steps because of time constraints of running the network to convergence, learning slowing down before convergence and divergence of the network after a few gradient descent updates. Thousands of gradient descent steps might not be nearly enough to come to a conclusion about MAPE convergence. The original study by Banerjee(2016) used 50000 gradient descent steps in 10000 pairs of neurons to produce the results of percent improvement in MAPE convergence. Convergence of the learning weights to witness weights happened after 10^6 gradient descent steps. An increase in the number of gradient descent steps can increase the improvement in MAPE, shifting the points towards the y=x line and also shift the data points between the positive (initial MAPE - final MAPE >0) and negative (initial MAPE - final MAPE <0) categories in Figure 3.1 and Figure 3.3. However, the networks we ran either diverged or learning slowed down significantly after a while.

4.3 Diversity of input

When we model spike trains as a real-valued point process, the first naive model used is the homogeneous Poisson model. In this model, the process is only characterized by its sole parameter, firing rate λ, which was fixed to be 10 Hz in our study. When it comes to homogeneous Poisson processes, where the firing rate λ remains constant, the non-stationarity of the spike trains cannot be accurately simulated. The lack of diversity in the spike train inputs can be a cause of lack of improvement in the convergence of weights. To improve the results, the input spike train is modeled from an inhomogeneous poisson process with a time dependent firing rate $\lambda(t)$. The frequency of spiketrain varied sinusoidally between 5 and 10 Hz at a frequency of 2 Hz. The spike trains are phase-shifted uniformly for the 15 synapses.

The increase in the number of converging neuron pairs when the input spike train is derived from an inhomogeneous poisson process shows that diversity in the input at the synapses is important for the synaptic weight update rule to learn the transformation accurately. When every synapse receives the same Poisson spike train input, even when the network learns the transformation since the algorithm is insensitive to the order of the synapses, a scrambled order of weights might be learned, which ultimately gives a large MAPE value. To discriminate one synapse from the other, it is necessary to give diverse input to each synapse.

4.4 Quiescent neurons and silent synapses

In a witness-learning neuron pair, when the learning neuron, driven by inhomogeneous poisson input is run till convergence, we observe that it is able to reach 81.6% convergence in 1075 gradient descent steps after which learning slows down considerably and the convergence of weights stops. It has been observed that the synapses that have very huge initial MAPE dominate the learning while other synapses

remain silent or learn very slowly. Learning slows down once the synapses with very high initial MAPEs converge considerably.

When synapses do not receive rich enough input from the presynaptic neuron (in this case, the poisson input from outside) for the neuron to generate spikes, they remain silent (or their weights do not change during learning). The synaptic weight update rule depends on the spiking of the neurons to show us what the synaptic weights are. If a neuron is not spiking, we obtain no information from the neuron that can aid learning.

Generalizing to a multi-layered network, if a presynaptic neuron is firing at a low rate, the PSPs at the synapse of postsynaptic neuron does not reach the spiking threshold or do not contribute significantly to spiking. As a result, the learning at that particular synapse stops. Silent synapses occur due to the dependence of the learning rule on the pre-post synaptic firing rate.

A potential solution for the silent synapse problem is by introducing homeostatic plasticity to the learning rule. Through homeostatic plasticity, if we are able to regulate the output spike rate of a synapse irrespective of the incoming spike train or the excitability of the postsynaptic neuron, the synapses would not go silent and therefore, learning doesn't slow down.

4.5 Role of homeostatic plasticity in solving silent synapse problem

Homeostatic plasticity counterbalances the many forces that destabilize activity in the neural circuits of the brain like long term potentiation (LTP) and long term depression (LTD). LTP, for example, is widely considered responsible for learning and storage of information but it can also result in increasing the excitability of the postsynaptic neuron which ultimately hinders the synaptic specificity of correlation-based plasticity mechanisms and information storage based on synaptic strengths (Turrigiano,2012). Homeostatic plasticity mechanisms work by generating an error signal which when deviates from an optimum set value, changes the excitability of the neuron to bring the activity towards the optimum value. Empirical evidence from several studies suggest that the neurons of the central nervous system are able to maintain an optimum average firing rate by compensatory changes like synaptic scaling which involves

increase or decrease of the amplitude distribution of miniature excitatory postsynaptic currents(mEPSPs)

(Turrigiano et.al, 1998;Desai et al., 2002). Homeostatic plasticity mechanisms in the CNS operate over a wide range of spatial and temporal scales - local and global as well as presynaptic and postsynaptic.

Synaptic scaling and intrinsic plasticity mechanisms have been used in many studies in order to stabilize learning with other learning rules, for example, STDP. One of the more popular mechanisms, global multiplicative synaptic scaling, adjusts the weights of synapses based on the error between target postsynaptic activity signal and time averaged postsynaptic activity signal. van Rossum (2000) models the activity dependent scaling as follows:

Let's say a(t) is the post-synaptic activity signal given by

$$\tau \frac{da}{dt} = -a(t) + \sum_i \delta(t - t_i)$$

where t_i are the spiketimes of neurons. We can update the weights at every time step according to:

$$\frac{dw}{dt} = \beta w(t)[a_{goal} - a(t)]$$

where a_{goal} gives the target postsynaptic activity and β is the learning rate.

Incorporating this with the current error functional of our synaptic weight update rule, will help to solve the problem caused by very high or low presynaptic neuron firing in a multilayered neuronal network. If a presynaptic neuron is firing at a high rate, the synapse from that neuron to the postsynaptic neuron will contribute significantly to the firing of the postsynaptic neuron which will result in a high firing rate of the postsynaptic neuron. However, the tuple of weights that can produce the output spike train from the postsynaptic neuron is no longer unique. Therefore, with each gradient update the weights will move towards one desired weight to the other and the error remains very high and erratic (Banerjee, 2016). Conversely, if the presynaptic neuron is firing at a low rate, the synapses from that neuron will contribute less to the firing of the postsynaptic neuron and therefore $\frac{\partial E}{\partial w}$ will be small and the learning of those synapses slows down

(Banerjee, 2016). This is what we call the silent synapse problem. Both of these scenarios can be solved by introducing a homeostatic plasticity mechanism to our weight update rule which will regulate the firing rate of the postsynaptic neuron.

4.6 Steepest descent and fixed learning rate

The synaptic weight update rule implemented in this work, updates the parameters i.e, the weights of the synapses by computing the gradient of the error function w.r.t. to the weight and following the direction of the steepest descent which is given by the negative of the gradient (Meza, 2010). This can be represented by:

$$\Delta x_t = -\eta g_t$$

where g_t is the gradient of the error function w.r.t the parameters at the t^{th} iteration and η is the learning rate.

Selecting an optimal learning rate is a very significant step in the learning process because a very high learning rate can result in overshooting the minima and a very low learning rate can result in very slow learning or getting stuck in a local minima. One way in which our synaptic weight update rule has made the learning rate adaptable is by introducing a cap on the update tensor that clamps the size of the update step at a maximum value. However, choosing the learning rate and cap on update tensor is not a trivial task. Suboptimal values can result in slow learning, oscillations, or getting stuck in a local minima. When the update vector is getting capped continuously, true learning can no longer take place and it has been observed to result in oscillations of the weights of the learning network. Manually increasing the cap on the update vector seems to solve the oscillation problem.

4.7 Momentum and Adaptive gradient methods

Adaptive gradient methods use an adaptive learning rate which takes into account the characteristics of the data and the parameters and helps in overcoming the issues discussed earlier caused due to fixed rate and cap.

One way to speed up learning and prevent its dependence of the hyperparameters, learning rate and cap on update tensor, is by using momentum and adaptive gradient methods. Adding momentum to gradient descent has two advantages: the weights converge faster and weights will not get stuck in a local minima. Momentum prevents the weights from going in the direction of steepest descent and pushes it to continue going in the previous direction. This also damps oscillations as gradients of opposite signs are added. Assigning separate learning rates to each synaptic connection makes sure that even if the gradients are very different for each synapse, there is no overshoot of minima caused by simultaneously adjusting multiple incoming weights of a neuron to rectify the same error.

Out of momentum, AdaGrad, RMSprop and Adam, the latter showed better convergence, at an accelerated rate without getting stuck in local minima or diverging after a few steps. Momentum showed accelerated convergence compared to Adam, but diverged after a few gradient descent steps. While using AdaGrad and RSMProp, learning slowed down considerably after a few gradient descent steps.

Adam inherits the benefits of AdaGrad, RMSProp and Momentum which makes it computationally faster and perform better than traditional optimisers.

Momentum : Momentum enables the algorithm to use information from the previous iterations to improve the speed and accuracy of the optimization process. It helps the gradient descent algorithm converge more quickly and efficiently. Gradient descent updates the weights of the neural network by taking a step in the direction of the steepest descent of the error function. However, this process can be slow and inefficient, especially when the error function has many local minima. Momentum helps to address this issue by taking into account the average of the previous batches of gradient descent, which allows the algorithm to remember the direction of the previous steps and use this information to make more informed decisions about which direction to take in the next iteration. This approach helps the optimization algorithm to converge faster towards the minimum point of the error function and avoid getting stuck in local minima.

RMSprop: RMSprop employs an adaptive learning rate. RMSprop adapts the learning rate of each weight based on the previous average of the magnitudes of its gradients. This allows the algorithm to take larger steps when the gradients are small and smaller steps when the gradients are large, which can help converge to the optimal solution faster. It only needs to store the exponential moving average of the squared gradients for each weight, which requires less memory than storing all the gradients of each weight over time. The two benefits mentioned above ,the adaptive learning rate and efficient memory usage of RMSprop can lead to faster convergence to the optimal solution, especially for complex non-convex optimization problems.

Adagrad : Adagrad adapts the learning rate of each weight based on the historical sum of the squares of its gradients. This allows the algorithm to take smaller steps for frequently occurring features and larger steps for infrequent features, which can be useful in dealing with sparse data. Adagrad, apart from having an adaptive learning rate also performs automatic feature selection. It downweights the importance of features that have already been well-learned which can help reduce the risk of learning slowing down due to small errors in the recent past.

Adam : Like Adagrad and RMSprop, Adam adapts the learning rate of each weight based on the historical average of the magnitudes of its gradients. However, instead of using the sum of the squares of the gradients or the moving average of the squared gradients, Adam uses the moving average of the first and second moments of the gradients, which can lead to more efficient and effective adaptation of the learning rate. Like momentum, Adam uses the moving average of the gradients to reduce the variance of the updates and accelerate the optimization process. However, unlike momentum, the moving average in Adam is biased towards zero in the early iterations, which can help with faster convergence.

4.8 Implications of the work

The choice of optimiser is very important when dealing with large datasets and large spiking neural networks. Traditional gradient descent in addition to being slow to converge, is also oblivious to the characteristics of the data being learned. In order to efficiently make use of time and memory, it is important to choose an optimiser that uses a per-parameter or adaptive learning rate and can accelerate the learning process. There is no universally best optimiser that suits every network and data. Adaptive gradient methods like AdaGrad, RMSProp and Adam are particularly useful when dealing with sparse gradient tasks, where many of the gradient values are close to zero (Zhou, 2020). In such cases, traditional methods like traditional gradient descent can be slow to converge, since the same learning rate is applied to all parameters. AdaGrad, RMSProp and Adam adaptively adjust the learning rate for each parameter based on the magnitude of recent gradients. This allows the learning rate to be scaled differently for different parameters, which can improve convergence rates in sparse gradient tasks. Adam especially has become an industry standard in the past few years because of its benefits. Adam inherits the benefits of Stochastic gradient descent with Momentum , AdaGrad and RMSProp. Adam seems a perfect fit to accelerate learning using our synaptic weight update rule and its properties ensure convergence of weights even when the traditional gradient descent is diverging.

We also identify the problem of silent synapses and discuss a way to address the problem when it occurs in a multi-layered feedforward network. The problem can be solved by adding a separate homeostatic plasticity error term that brings the frequency of the output spike train closer to the target spike train as a first step during the weight update rule. Later this term switches off and the spike time disparity rule takes precedence to learn the precise timing of spikes.

4.9 Future directions

The current study involves implementation of the synaptic weight update rule on a single spiking neuron receiving multiple synaptic inputs. The next step would be implementing the weight update rule on a spiking neural network with one or multiple hidden layers. For an intermediate layer neuron, gradient descent is defined as

$$\frac{\partial t_l^O}{\partial w_{g,h}} = \sum_{j \epsilon \mathcal{F}_\rangle} \frac{\partial t_l^O}{\partial t_{i,j}^I} \frac{\partial t_j^H}{\partial w_{g,h}}$$

where,

$$\frac{\partial t_l^O}{\partial t_{i,j}} = \frac{w_{i,j} \frac{\partial \varepsilon_i}{\partial t}|_{(t_{i,j}^I - t_l^O)} + \sum_{k \epsilon F} \frac{\partial \eta}{\partial t}|_{(t_k^O - t_l^O)} \frac{\partial t_k^O}{\partial t_{i,j}}}{\sum_{i \epsilon \Gamma} \sum_{j \epsilon F_i} w_{i,j} \frac{\partial \varepsilon_i}{\partial t}|_{(t_{i,j}^I - t_l^O)} + \sum_{k \epsilon F} \frac{\partial \eta}{\partial t}|_{(t_k^O - t_l^O)}}$$

The problem of quiescent neurons can be solved by incorporating homeostatic plasticity that changes the excitability of neurons. Homeostatic plasticity mechanisms function by producing an error signal that alters the neuron's excitability to move the activity closer to the optimal value when it strays from the set value. Further details of implementation discussed in section 4.5.

REFERENCES

1. Banerjee, A. (2001). On the Phase-Space Dynamics of Systems of Spiking Neurons. I: Model and Experiments. Neural Computation, 13(1), 161–193.
2. Banerjee, A. (2016). Learning precise spike train-to-spike train transformations in multilayer feedforward neuronal networks. Neural Computation, 28(5), 826–848.
3. Bialek, W., Rieke, F., Steveninck, R. de Ruyter van, & Warland, D. (1991). Reading a neural code. Science, 252, 1854-1857.
4. Bohte, S. M., Kok, J. N., & La Poutré, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing, 48(1), 17–37.
5. Booij, O., & tat Nguyen, H. (2005). A gradient descent rule for spiking neurons emitting multiple spikes. Information Processing Letters, 95(6), 552–558.
6. Brette R (2012) Computing with Neural Synchrony. PLoS Comput Biol 8(6): e1002561.
7. Brette, R. (2015) 'Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain', Frontiers in Systems Neuroscience, 9.
8. Brunel, N., & van Rossum, M. C. W. (2007). Lapicque's 1907 paper: from frogs to integrate-and-fire. In Biological Cybernetics. Springer Science and Business Media LLC. 97(5–6) pp. 337–339.
9. Deneve, S. (2008). Bayesian spiking neurons i: Inference. Neural Computation, 20(1):91-117.
10. de Ryter van Steveninck, R., Lewen, G., Strong, S., Koberle, R., & Bialek,W. (1997). Reproducibility and variability in neural spike trains. Science, 275, 1805-1808.
11. Desai, N.S., Cudmore, R.H., Nelson, S.B., & Turrigiano, G.G. (2002). Critical periods for experience-dependent synaptic scaling in visual cortex. Nature Neuroscience, 5, 783-789.
12. Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).

13. Florian R. V. (2012). The chronotron: a neuron that learns to fire temporally precise spike patterns. PloS one, 7(8), e40233.
14. Gawne, T. J., Kjaer, T. W., & Richmond, B. J. (1996). Latency: another potential code for feature binding in striate cortex. Journal of neurophysiology, 76(2), 1356–1360.
15. Gerstner, W., Ritz, R. & van Hemmen, J.L. (1993). Why spikes? Hebbian learning and retrieval of time-resolved excitation patterns. Biol. Cybern. 69, 503–515.
16. Gerstner, W. (1995). Time structure of the activity in neural network models. Phys. Rev. E, 51, 738–758.
17. Gütig, R., & Sompolinsky, H. (2006). The tempotron: a neuron that learns spike timing–based decisions. Nature Neuroscience. Springer Science and Business Media LLC. 9(3), pp. 420–428.
18. Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. The Journal of physiology, 117(4), 500–544.
19. Izhikevich. E. M. (2003). Simple model of spiking neurons. IEEE Transactions on Neural Networks. vol. 14. no. 6. pp. 1569-1572.
20. Izhikevich E. M. (2006). Polychronization: computation with spikes. Neural computation, 18(2), 245–282.
21. Johansson, R. S., & Birznieks, I. (2004). First spikes in ensembles of human tactile afferents code complex spatial fingertip events. Nature neuroscience, 7(2), 170–177.
22. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
23. Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. In Neural Networks (Vol. 10, Issue 9, pp. 1659–1671). Elsevier BV.
24. Meza, J.C. (2010). Steepest descent. Wiley Interdisciplinary Reviews: Computational Statistics, 2.

25. Middlebrooks, J. C., Clock, A. E., Xu, L., & Green, D. M. (1994). A panoramic code for sound location by cortical neurons. Science, 264(5160), 842–844.
26. Nemenman I, Lewen GD, Bialek W, de Ruyter van Steveninck RR (2008) Neural Coding of Natural Stimuli: Information at Sub-Millisecond Resolution. PLoS Comput Biol 4(3): e1000025.
27. Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016). Training Deep Spiking Neural Networks Using Backpropagation. Frontiers in Neuroscience, 10.
28. Panzeri S, Petersen RS, Schultz SR, Lebedev M, Diamond ME (2001) The role of spike timing in the coding of stimulus location in rat somatosensory cortex. Neuron 29:769–77
29. Ponulak, F. (2005). ReSuMe—new supervised learning method for spiking neural networks (Tech. Rep.). Poznan: Institute of Control and Information Engineering, Poznan University of Technology.
30. Ramaswamy, V., & Banerjee, A. (2014). Connectomic constraints on computation in feedforward networks of spiking neurons. Journal of computational neuroscience, 37(2), 209–228.
31. Rumelhart, D., Hinton, G. & Williams, R.(1986). Learning representations by back-propagating errors. Nature 323, 533–536.
32. Thorpe, S., Delorme, A., & Van Rullen, R. (2001). Spike-based strategies for rapid processing. Neural networks : the official journal of the International Neural Network Society, 14(6-7), 715–725.
33. Turrigiano, G. G., Leslie, K. R., Desai, N. S., Rutherford, L. C., & Nelson, S. B. (1998). Activity-dependent scaling of quantal amplitude in neocortical neurons. Nature, 391(6670), 892–896.
34. Turrigiano G. (2012). Homeostatic synaptic plasticity: local and global mechanisms for stabilizing neuronal function. Cold Spring Harbor perspectives in biology, 4(1), a005736.
35. Victor, J. D., and Purpura, K. P. (1996). Nature and precision of temporal coding in visual cortex: a metric-space analysis. J. Neurophysiol. 76, 1310–1326.
36. Widrow .B and Hoff M. E. Adaptive Switching Circuits. (1960). IRE WESCON Convention Record, pp. 96-104.

37. Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.
38. Zenke, F., & Ganguli, S. (2018). SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks. Neural computation, 30(6), 1514–1541. https://doi.org/10.1162/neco_a_01086
39. Zeldenrust, F., Wadman, W. J., & Englitz, B. (2018). Neural Coding With Bursts—Current State and Future Perspectives. Frontiers in Computational Neuroscience, 12.
40. Zhou, D., Chen, J., Cao, Y., Tang, Y., Yang, Z., & Gu, Q. (2020) On the convergence of adaptive gradient methods for nonconvex optimization. NeurIPS Workshop on Optimization for Machine Learning (OPT)