Supervised Learning in Multilayer Spiking Neural Network

Sumit Bam Shrestha

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University in partial fulfillment of the requirement for the degree of Doctor of Philosophy
Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted to higher degree to any other University or Institution.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

______________________________
Date

______________________________
SUMIT BAM SHRESTHA
To my Mother and Father.
Acknowledgement

I am deeply indebted to Dr. Song Qing, my adviser and mentor during my PhD research, for his constant guidance, motivation and encouragement. The research traits that I learned from him during the period of my PhD are invaluable. Apart from research, the emotional support he provided to me during the devastating earthquake in Nepal in 2015 was really commendable. I would like to thank him from the bottom of my heart.

I am grateful to my mother and father for all their support, motivation and encouragement that helped me throughout the period of my PhD study. Not to mention my wonderful friends and colleagues who made this period enjoyable and lend support to me throughout these years.

I would like to express my gratitude to my research colleague Dr. Haijin Fan for his help and support especially during my early years of my PhD. Many thanks to Mr. Sao Peck Heng and Mr. Yuen Sien Huan, the staff at Internet of Things Lab, for providing me work-space resources for my PhD research. My thanks to the research students and my friends at Internet of Things Lab for their kind help and support.

My sincere thanks to SINGA scholarship program committee for providing me with financial assistance for my PhD and Nanyang Technological University staffs for their immense help during my study. Finally, I would like to thank everyone who lend support to me and helped me during these years.
Abstract

Spiking Neural Networks (SNNs) are an exciting prospect in the field of Artificial Neural Networks (ANNs). We try to replicate the massive interconnection of neurons, the computational units, evident in brain to perform useful task in ANNs, albeit with highly abstracted model of neurons. Mostly the artificial neurons are realized in the form of non-linear activation function which process numeric inputs and output. SNNs are less abstract than these systems with non-linear activation function in the sense that they make use of mathematical model of neurons, termed spiking neurons, which process inputs in the form of spikes and emits spike as output. This is exactly the way in which natural neurons exchange information. Since spikes are events in time, there is an extra dimension of time along with amplitude in SNNs which makes them suited to temporal processes.

There are a few supervised learning algorithms for learning in SNN. As far as learning in multilayer architecture, we have SpikeProp and its extensions and Multi-ReSuMe. The SpikeProp methods are based on adaptation of backpropagation for SNNs and mostly consider first spike of the neuron. Original SpikeProp is usually slow and face stability issues during learning. Large learning rate and even very small learning rate often makes it unstable. The instability is observable in the form of sudden jumps in training error, called surge, which change the course of learning and often cause failure of the learning process as well.

To introduce stability criterion, we present weight convergence analysis of SpikeProp. Based on the convergence condition, we introduce an adaptive learning rate rule which selects suitable learning rate to guarantee convergence of learning process and large enough learning rate so that the learning process is fast enough. Based on performance on several benchmark problems, this method with learning rate adaptation, SpikePropAd, demonstrates less surges and faster learning as well compared to SpikeProp and its faster variant RProp. The performance is evaluated broadly in terms of speed of learning, rate of successful learning.

We also consider the internal and external disturbances to the learning process and provide a thorough error analysis in addition to weight convergence analysis. We use conic sector stability theory to determine the conditions for making the learning process stable in \( L_2 \) space and extend the result for \( L_\infty \) stability. \( L_2 \) stability in theory requires the disturbance to die out after a certain period of time whereas the \( L_\infty \) stability implies that the system is stable provided the disturbance is within bounds. We explore two approaches for robust stability
of SpikeProp in presence of disturbance: individual error approach, which leads to SpikePropR learning and total error approach, which leads to SpikePropRT learning. SpikePropR has slight improvement over SpikePropAd. SpikePropRT on the other hand has significant improvement over SpikePropAd, especially for real world non synthetic datasets.

An event based weight update rule for learning spike-train rather than the time of first spike, EvSpikeProp, is also proposed. This method overcomes the limitations of other multi-spike extension of SpikeProp and is suitable for learning in an online fashion which is more suited to SNNs because spikes are continuous processes. The results derived in the convergence and stability analysis of SpikeProp are extended for multi-spike framework to show weight convergence and robust stability in $L_2$ and $L_\infty$ space. The resulting method is named EvSpikePropR. It shows better performance compared to Multi-ReSuMe based on the performance results for different learning problems.

Apart from that, we also extended the adaptive learning rule based on weight convergence for delay learning of SNN as well. It is named SpikePropAdDel. This delay learning extension is useful because it speeds the learning process, eliminates redundant synapses and minimizes surge as well.
# Table of Contents

Acknowledgement  
Abstract  
List of Figures  
List of Tables  
List of Algorithms  
Abbreviations and Symbols  

1 Introduction  
1.1 Spiking Neural Networks  
1.1.1 The Biological Inspiration  
1.1.2 Significance of SNN as a Computational Unit  
1.1.3 Neuromorphic Significance  
1.2 Main Contribution  
1.3 Other Current Research in SNN  
1.4 Overview of Thesis  

2 Spiking Neuron Basics  
2.1 Dynamics of a Biological Neuron  
2.2 Models of Spiking Neuron  
2.2.1 Hodgkin-Huxley Model  
2.2.2 Two Dimensional Neuron Model  
2.2.3 One Dimensional Neuron Model  
2.2.4 Spike Response Model  
2.2.5 Type A and Type B Spiking Neuron  
2.3 Spike Coding  
2.3.1 Population Encoding  
2.3.2 Threshold Based Encoding  

3 Supervised Learning in SNN: An Overview  
3.1 Neural Network Architecture  
3.2 Forward Simulation of SNN
## Table of Contents

### 3.3 SpikeProp
- 3.3.1 Error Backpropagation ........................................ 34
- 3.3.2 Implementation Caveats ...................................... 36

### 3.4 Extension to SpikeProp
- 3.4.1 Extended SpikeProp ........................................... 38
- 3.4.2 QuickProp .................................................. 39
- 3.4.3 RProp ..................................................... 40
- 3.4.4 SpikeProp for Multiple Spiking Neuron .................... 40

### 3.5 Remote Supervision Method ..................................... 42

### 3.6 Multi-ReSuMe .................................................. 43

### 3.7 Performance Measures .......................................... 44
- 3.7.1 Network Error ............................................... 44
- 3.7.2 Convergence Rate ........................................... 45
- 3.7.3 Mean Epoch ................................................. 45
- 3.7.4 Accuracy .................................................. 46
- 3.7.5 Average Learning Curve .................................... 46

### 4 Non-Linear Stability Theory ...................................... 47
  4.1 Mathematical Preliminaries .................................. 47
    - 4.1.1 The $L_2$ space and Extended $L_2$ space .............. 48
    - 4.1.2 The $L_{\infty}$ space ................................ 49
    - 4.1.3 Gain .................................................. 49
    - 4.1.4 Passivity ............................................ 50
    - 4.1.5 Conic Sector ........................................ 50
    - 4.1.6 Normalized Signal .................................... 51
    - 4.1.7 Exponentially Weighted Signal ....................... 52
  4.2 Lyapunov Stability ............................................ 52
  4.3 Input-Output Stability ....................................... 53
    - 4.3.1 Conic Sector Stability ................................ 54
    - 4.3.2 Extension of Conic Sector Stability to $L_{\infty}$ space 59

### 5 SpikeProp with Adaptive Learning Rate ......................... 63
  5.1 The SpikePropAd Algorithm .................................. 64
  5.2 Weight Convergence Analysis of SpikePropAd ................ 65
  5.3 Performance Comparison of SpikePropAd ..................... 69
    - 5.3.1 XOR problem .......................................... 69
    - 5.3.2 Fisher’s Iris classification .......................... 73
    - 5.3.3 Wisconsin Breast Cancer classification .............. 77
    - 5.3.4 Statlog (Landsat Satellite) data classification .... 82
  5.4 Choice of SpikePropAd Learning Parameter .................. 86

### 6 SpikeProp with Adaptive Delay Learning ......................... 89
  6.1 The SpikePropAdDel Algorithm ................................ 90
  6.2 Delay Convergence Analysis of SpikePropAdDel ............. 92
  6.3 Performance Comparison of SpikePropAdDel .................. 95
    - 6.3.1 XOR problem .......................................... 96
# Table of Contents

6.3.2 Fisher’s Iris classification ........................................ 99
6.3.3 Wisconsin Breast Cancer classification .......................... 101
6.3.4 Statlog (Landsat Satellite) data classification ............... 106
6.4 Choice of SpikePropAdDel Learning Parameter .................. 111

7 Robust Learning in SpikeProp ........................................ 113
  7.1 The SpikePropR Algorithm ........................................ 114
  7.2 Error System Formulation for SpikePropR ....................... 115
    7.2.1 Output Layer Error System ................................ 117
    7.2.2 Hidden Layer Error System ................................ 118
  7.3 Weight Convergence Analysis of SpikePropR .................... 121
  7.4 Robust Stability Analysis of SpikePropR ....................... 123
  7.5 The SpikePropRT Algorithm ..................................... 126
  7.6 Error System Formulation for SpikePropRT .................... 127
    7.6.1 Output Layer Error System ................................ 129
    7.6.2 Hidden Layer Error System ................................ 130
  7.7 Weight Convergence Analysis of SpikePropRT .................... 131
    7.7.1 Output Layer Weight Convergence Analysis ................. 132
    7.7.2 Hidden Layer Weight Convergence Analysis ............... 133
  7.8 Robust Stability Analysis of SpikePropRT ....................... 135
  7.9 Performance Comparison ........................................ 138
    7.9.1 XOR problem ............................................... 139
    7.9.2 Fisher’s Iris classification ................................ 142
    7.9.3 Wisconsin Breast Cancer classification ................... 146
    7.9.4 Statlog (Landsat Satellite) data classification ........ 150
  7.10 Choice of SpikePropR Learning Parameter ...................... 154
  7.11 Choice of SpikePropRT Learning Parameter ..................... 154

8 Event based SpikeProp for Learning Spike Train .................. 157
  8.1 The EvSpikeProp Algorithm ..................................... 158
    8.1.1 Spiking Neuron Model and Network Architecture ........... 158
    8.1.2 Event Based Weight Update ................................ 159
    8.1.3 The Recurrent Factor ...................................... 160
    8.1.4 Error Backpropagation .................................... 161
  8.2 The EvSpikePropR Algorithm .................................... 163
  8.3 Weight Convergence Analysis of EvSpikePropR ................... 164
    8.3.1 Output layer Weight Convergence Analysis ................. 164
    8.3.2 Hidden layer Weight Convergence Analysis ............... 166
  8.4 Robust Stability Analysis of EvSpikePropR ...................... 170
  8.5 Performance Comparison of EvSpikePropR ....................... 173
    8.5.1 XOR problem ............................................... 173
    8.5.2 Fisher’s Iris classification ................................ 176
    8.5.3 Wisconsin Breast Cancer classification ................... 179
    8.5.4 Poisson Spike data classification ........................ 183
  8.6 Choice of EvSpikePropR Learning Parameter ..................... 187
## 9 Conclusion and Future Work 191

9.1 Conclusion ........................................ 191
9.2 Future Work ....................................... 195
   9.2.1 Robustness Analysis of Delay Learning Extension of SpikeProp 195
   9.2.2 Delay Adaptation for Learning Spike Train .................. 195
   9.2.3 Implementation in Neuromorphic Chips ..................... 196
   9.2.4 Spike Encoding and Decoding Principle ................. 196

## A Matrix Calculus 197

A.1 Derivatives of Vector Functions .............. 197
   A.1.1 Derivative of Vector with Respect to Vector .... 198
   A.1.2 Derivative of Scalar with Respect to Vector .... 198
   A.1.3 Derivative of Vector with Respect to Scalar .... 198
   A.1.4 Chain Rule for Vector Functions .............. 198
   A.1.5 Vector Derivative Formulae ................... 199

A.2 Derivative of Matrix functions ................. 200
   A.2.1 Derivative of Scalar by a Matrix ............... 200
   A.2.2 Derivative of Matrix by a Scalar .............. 200

A.3 Derivative of Function of Matrix Product .... 200

## B Discrete Bellman-Gronwall Lemma 203

## Author’s Publications 207

## Bibliography 209
List of Figures

1.1 In vivo membrane potential of visual cortex neuron. .......................... 3
1.2 Computation in terms of spikes. The vertical lines in the input and output represent spikes. .......................................................... 4

2.1 Typical EPSP and IPSP response of a biological neuron. .................. 13
2.2 Dynamics of a biological neuron .................................................. 14
2.3 Hodgkin Huxley model based on ion channels. ............................. 15
2.4 Type A and Type B spiking neurons as defined in [1] (a) spike response function for Type A spiking neuron (b) spike response function for Type B spiking neuron (c) variable threshold for refractoriness. .......................................................... 21
2.5 Population encoding of input values using Gaussian receptive fields. 23
2.6 Threshold based encoding of a continuous signal. .......................... 24

3.1 SNN architecture with |K| delayed synaptic connections, |I| input layer neurons, |H| hidden layer neurons and |O| output layer neurons. .......................................................... 28
3.2 Weight adaptation in ReSuMe learning using correlation between input-output spike train and input-teacher spike train. ................. 42
3.3 Weight adaptation in ReSuMe learning using learning windows. (a) Changes in weight \( w_{oi} \) caused by teacher neuron spike and input neuron spike. (b) Changes in weight \( w_{oi} \) caused by output neuron spike and input neuron spike. .......................................................... 43

4.1 A conic sector in a plane .......................................................... 50
4.2 Closed-loop feedback system .................................................... 54
4.3 Closed-loop feedback system for vector signals ............................ 57

5.1 Surges during SpikeProp training for XOR data, \( \eta = 0.01 \). ............ 63
5.2 Convergence Rate plot for learning XOR problem using SpikeProp, RProp and SpikePropAd .................................................. 70
5.3 Average number of epochs required to learn XOR problem using SpikeProp, Rprop and SpikePropAd ........................................ 71
5.4 Average learning curves during XOR problem learning using SpikeProp, RProp and SpikePropAd ........................................ 71
5.5 Typical learning curves during XOR pattern learning using SpikeProp, RProp and SpikePropAd. ................................................. 72
5.6 Trace of learning rate of output layer neuron in a typical XOR learning using SpikePropAd with $\lambda = 0.01$. ................................. 72
5.7 Trace of learning rate of a hidden layer neuron in a typical XOR learning using SpikePropAd with $\lambda = 0.01$. ................................. 72
5.8 Convergence Rate plot for learning Fisher’s Iris data using SpikeProp, RProp and SpikePropAd. ....................................................... 74
5.9 Average number of epochs required to learn Fisher’s Iris data using SpikeProp, Rprop and SpikePropAd. ........................................ 74
5.10 Training accuracy in Fisher’s Iris classification using SpikeProp, RProp and SpikePropAd. ............................................................. 75
5.11 Testing accuracy in Fisher’s Iris classification using SpikeProp, RProp and SpikePropAd. ............................................................... 75
5.12 Average learning curves during Fisher’s Iris classification using SpikeProp, RProp and SpikePropAd. .................................................. 76
5.13 Typical learning curves during Fisher’s Iris classification using SpikeProp, RProp and SpikePropAd. ................................................... 76
5.14 Trace of learning rate of output layer neuron in a typical Fisher’s Iris learning using SpikePropAd with $\lambda = 0.01$ for first 1000 iterations. 77
5.15 Trace of learning rate of a hidden layer neuron in a typical Fisher’s Iris learning using SpikePropAd with $\lambda = 0.01$ for first 1000 iterations. 77
5.16 Convergence Rate plot for learning Wisconsin Breast Cancer data using SpikeProp, RProp and SpikePropAd. ...................................... 78
5.17 Average number of epochs required to learn Wisconsin Breast Cancer data using SpikeProp, Rprop and SpikePropAd. ....................... 79
5.18 Training accuracy in Wisconsin Breast Cancer classification using SpikeProp, RProp and SpikePropAd. .............................................. 79
5.19 Testing accuracy in Wisconsin Breast Cancer classification using SpikeProp, RProp and SpikePropAd. ............................................... 80
5.20 Average learning curves during Wisconsin Breast Cancer classification using SpikeProp, RProp and SpikePropAd. ................................. 80
5.21 Typical learning curves during Wisconsin Breast Cancer classification using SpikeProp, RProp and SpikePropAd. ............................... 81
5.22 Trace of learning rate of output layer neuron in a typical Wisconsin Breast Cancer classification using SpikePropAd with $\lambda = 0.01$ for first 1000 iterations. ......................................................... 81
5.23 Trace of learning rate of a hidden layer neuron in a typical Wisconsin Breast Cancer classification using SpikePropAd with $\lambda = 0.01$ for first 1000 iterations. ......................................................... 81
5.24 Convergence Rate plot for learning Statlog (Landsat Satellite) data using SpikeProp, RProp and SpikePropAd. ...................................... 83
5.25 Average number of epochs required to learn Statlog (Landsat Satellite) data using SpikeProp, RProp and SpikePropAd. .......................... 83
5.26 Training accuracy in Statlog (LandSat Satellite) classification using SpikeProp, RProp and SpikePropAd. .................................................. 84
5.27 Testing accuracy in Statlog (LandSat Satellite) classification using SpikeProp, RProp and SpikePropAd. .................................................. 84
5.28 Average learning curves during Statlog (Landsat Satellite) classification using SpikeProp, RProp and SpikePropAd. .................................................. 85
5.29 Typical learning curves during Statlog (Landsat Satellite) data classification SpikeProp, RProp and SpikePropAd. .................................................. 85
5.30 Trace of learning rate of output layer neuron in a typical Statlog (Landsat Satellite) data classification using SpikePropAd with $\lambda = 0.001$ for first 1000 iterations. .................................................. 86
5.31 Trace of learning rate of a hidden layer neuron in a typical Statlog (Landsat Satellite) data classification using SpikePropAd with $\lambda = 0.001$ for first 1000 iterations. .................................................. 86

6.1 Convergence Rate plot for learning XOR pattern using SpikePropDel and SpikePropAdDel. .................................................. 96
6.2 Average number of epochs required to learn XOR pattern using SpikePropDel and SpikePropAdDel. .................................................. 97
6.3 Average learning curves for XOR problem using SpikePropDel and SpikePropAdDel. .................................................. 97
6.4 Typical learning curves for XOR problem using SpikePropDel and SpikePropAdDel. .................................................. 99
6.5 Convergence Rate plot for learning Fisher’s Iris data using SpikePropDel and SpikePropAdDel. .................................................. 100
6.6 Average number of epochs required to learn Fisher’s Iris data using SpikePropDel and SpikePropAdDel. .................................................. 100
6.7 Training accuracy in Fisher’s Iris classification using SpikePropDel and SpikePropAdDel. .................................................. 101
6.8 Testing accuracy in Fisher’s Iris classification using SpikePropDel and SpikePropAdDel. .................................................. 101
6.9 Average learning curves during Fisher’s Iris classification using SpikePropDel and SpikePropAdDel. .................................................. 103
6.10 Typical learning curves during Fisher’s Iris classification using SpikePropDel and SpikePropAdDel. .................................................. 103
6.11 Convergence Rate plot for learning Wisconsin Breast Cancer data using SpikePropDel and SpikePropAdDel. .................................................. 104
6.12 Average number of epochs required to learn Wisconsin Breast Cancer data using SpikePropDel and SpikePropAdDel. .................................................. 104
6.13 Training accuracy in Wisconsin Breast Cancer classification using SpikePropDel and SpikePropAdDel. .................................................. 105
6.14 Testing accuracy in Wisconsin Breast Cancer classification using SpikePropDel and SpikePropAdDel. .................................................. 105
6.15 Average learning curves during Wisconsin Breast Cancer classification using SpikePropDel and SpikePropAdDel. .................................................. 106
6.16 Typical learning curves during Wisconsin Breast Cancer classification using SpikePropDel and SpikePropAdDel. ................. 106
6.17 Convergence Rate plot for learning Statlog (Landsat Satellite) data using SpikePropDel and SpikePropAdDel. ................. 108
6.18 Average number of epochs required to learn Statlog (Landsat Satellite) data using SpikePropDel and SpikePropAdDel. ................. 109
6.19 Training accuracy in Statlog (Landsat Satellite) classification using SpikePropDel and SpikePropAdDel. ......................... 109
6.20 Testing accuracy in Statlog (Landsat Satellite) classification using SpikePropDel and SpikePropAdDel. ......................... 111
6.21 Average learning curves during Statlog (Landsat Satellite) classification using SpikePropDel and SpikePropAdDel. ................. 111
6.22 Typical learning curves during Statlog (Landsat Satellite) classification using SpikePropDel and SpikePropAdDel. ................. 112

7.1 Closed-loop feedback system of SpikePropR weight update. . . . . 123
7.2 Closed-loop feedback system for SpikePropRT weight update. . . . 135
7.3 Convergence Rate plot for learning XOR pattern using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ................. 140
7.4 Average number of epochs required to learn XOR pattern using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. .... 140
7.5 Average learning curves for XOR problem using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................... 141
7.6 Typical learning curves for XOR problem using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................... 141
7.7 Trace of output and hidden layer learning rate in a typical XOR learning using SpikePropR ($\epsilon_m = 0.001$). ......................... 141
7.8 Trace of output and hidden layer learning rate in a typical XOR learning using SpikePropRT ($\eta = 1$, $\bar{\rho} = 1$, $\epsilon_m = 0.01$, $\mu = 0.5$). 142
7.9 Convergence Rate plot for learning Fisher’s Iris data using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ................. 143
7.10 Average number of epochs required to learn Fisher’s Iris data using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. .... 143
7.11 Training accuracy in Fisher’s Iris classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................... 144
7.12 Testing accuracy in Fisher’s Iris classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................... 144
7.13 Average learning curves during Fisher’s Iris classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ................. 145
7.14 Typical learning curves during Fisher’s Iris classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ................. 145
7.15 Trace of output and hidden layer learning rate in a typical Fisher’s Iris learning using SpikePropRT ($\epsilon_m = 0.001$) for first 1000 iterations. 146
7.16 Trace of output and hidden layer learning rate in a typical Fisher’s Iris learning using SpikePropRT ($\eta = 1$, $\bar{\rho} = 0.1$, $\epsilon_m = 0.001$, $\mu = 0.01$) for first 2000 iterations. ......................... 146
7.17 Convergence Rate plot for learning Wisconsin Breast Cancer data using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 147
7.18 Average number of epochs required to learn Wisconsin Breast Cancer data using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 147
7.19 Training accuracy in Wisconsin Breast Cancer classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 148
7.20 Testing accuracy in Wisconsin Breast Cancer classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 148
7.21 Average learning curves during Wisconsin Breast Cancer classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 149
7.22 Typical learning curves during Wisconsin Breast Cancer classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 149
7.23 Trace of output and hidden layer learning rate in a typical Wisconsin Breast Cancer classification using SpikePropR ($\epsilon_m = 0.001$) for first 5000 iterations. ......................................................... 149
7.24 Trace of output and hidden layer learning rate in a typical Wisconsin Breast Cancer classification using SpikePropRT ($\eta = 1$, $\bar{\rho} = 0.1$, $\epsilon_m = 0.01$, $\mu = 0.05$) for first 10000 iterations. ......................................................... 150
7.25 Convergence Rate plot for learning Statlog (Landsat Satellite) data using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 151
7.26 Average number of epochs required to learn Statlog (Landsat Satellite) data using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 151
7.27 Training accuracy in Statlog (LandSat Satellite) classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 152
7.28 Training accuracy in Statlog (LandSat Satellite) classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 152
7.29 Average learning curves during Statlog (Landsat Satellite) classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 153
7.30 Typical learning curves during Statlog (Landsat Satellite) classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT. ......................................................... 153
7.31 Trace of learning rate of a neuron in a typical Statlog (Landsat Satellite) data classification using SpikePropR ($\epsilon_m = 0.0001$) for first 5,000 iterations. ......................................................... 153
7.32 Trace of learning rate of a neuron in a typical Statlog (Landsat Satellite) data classification using SpikePropRT ($\eta = 1$, $\bar{\rho} = 0.1$, $\epsilon_m = 0.01$, $\mu = 0.5$) for first 500,000 iterations. 154

8.1 Event based weight update strategy. 160
8.2 Closed-loop feedback system of EvSpikePropR weight update. 170
8.3 Convergence Rate plot for learning XOR pattern using MultiReSuMe, EvSpikePropR and EvSpikePropR. 174
8.4 Average number of epochs required to learn XOR pattern using MultiReSuMe, EvSpikePropR and EvSpikePropR. 175
8.5 Average learning curves for XOR problem using MultiReSuMe, EvSpikePropR and EvSpikePropR. 175
8.6 Spike Raster plot after each epoch during XOR learning using MultiReSuMe, EvSpikePropR and EvSpikePropR. 176
8.7 Trace of output and hidden layer learning rate in a typical XOR learning using EvSpikePropR ($\eta = 0.1$, $\bar{\rho} = 0.5$, $\epsilon_m = 0.001$, $\mu = 0.1$). 176
8.8 Convergence Rate plot for learning Fisher’s Iris data using MultiReSuMe, EvSpikePropR and EvSpikePropR. 177
8.9 Average number of epochs required to learn Fisher’s Iris data using MultiReSuMe, EvSpikePropR and EvSpikePropR. 178
8.10 Average learning curves during Fisher’s Iris classification using MultiReSuMe, EvSpikePropR and EvSpikePropR. 178
8.11 Spike Raster plot after each epoch during Fisher’s Iris Learning using MultiReSuMe, EvSpikePropR and EvSpikePropR. 179
8.12 Training accuracy in Fisher’s Iris classification using MultiReSuMe, EvSpikePropR and EvSpikePropR. 179
8.14 Trace of output and hidden layer learning rate in a typical Fisher’s Iris learning using EvSpikePropR ($\eta = 0.1$, $\bar{\rho} = 0.1$, $\epsilon_m = 0.1$, $\mu = 0.5$) for first 1000 iterations. 180
8.16 Average number of epochs required to learn Wisconsin Breast Cancer data using MultiReSuMe, EvSpikePropR and EvSpikePropR. 181
8.18 Spike Raster plot after each epoch during Wisconsin Breast Cancer Learning using MultiReSuMe, EvSpikePropR and EvSpikePropR. 182
8.21 Trace of output and hidden layer learning rate in a typical Wisconsin Breast Cancer classification using EvSpikePropR ($\eta = 0.032$, $\bar{\rho} = 0.1$, $\epsilon_m = 0.001$, $\mu = 0.05$) for first 1000 iterations. 184
8.22 Convergence Rate plot for learning Poisson spike data classification using MultiReSuMe, EvSpikePropR and EvSpikePropR. 185
8.23 Average number of epochs required to learn Poisson spike data classification using MultiReSuMe, EvSpikePropR and EvSpikePropR. 185
8.24 Average learning curves during Poisson spike data classification using MultiReSuMe, EvSpikePropR and EvSpikePropR. 186
8.25 Spike Raster plot after each epoch during Poisson spike data classification Learning using MultiReSuMe, EvSpikePropR and EvSpikePropR. 186
8.28 Trace of output and hidden layer learning rate in a typical Poisson spike data classification using EvSpikePropR ($\eta = 0.032$, $\bar{\rho} = 1$, $\epsilon_m = 0.1$, $\mu = 0.5$) for first 1000 iterations. 189
List of Tables

1.1 Computational capacity of SNN with Type A neuron on Boolean functions ........................................... 5

5.1 Comparison of performance of SNN with standard neural network methods [2] ........................................... 87

5.2 Variation of \( \lambda \) in SpikePropAd. ........................................... 88

6.1 SpikePropAdDel Simulation Results: XOR problem ........................................... 98
6.2 SpikePropAdDel Simulation Results: Fisher’s Iris classification ........................................... 102
6.3 SpikePropAdDel Simulation Results: Wisconsin Breast cancer classification ........................................... 107
6.4 SpikePropAdDel Simulation Results: Landsat data classification ........................................... 110
6.5 Variation of \( \lambda_d \) in SpikePropAdDel. ........................................... 112

7.1 Variation of \( \epsilon_m \) in SpikePropR. ........................................... 155
7.2 Variation of learning parameters in SpikePropRT. ........................................... 156

8.1 Variation of learning parameters in EvSpikePropR. ........................................... 188
## List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>First firing time of Neuron $N_j$</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>Spike train firing times of Neuron $N_j$</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>SpikeProp Learning</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>SpikePropAd Learning</td>
<td>66</td>
</tr>
<tr>
<td>5</td>
<td>SpikePropAdDel Learning</td>
<td>91</td>
</tr>
<tr>
<td>6</td>
<td>SpikePropR Learning</td>
<td>116</td>
</tr>
<tr>
<td>7</td>
<td>SpikePropRT Learning</td>
<td>128</td>
</tr>
</tbody>
</table>
# Abbreviations and Symbols

## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGI</td>
<td>Artificial General Intelligence</td>
</tr>
<tr>
<td>AHP</td>
<td>After-Hyperpolarization Potential</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>AP</td>
<td>Action Potential</td>
</tr>
<tr>
<td>BCESNN</td>
<td>Basis Coupled Evolving Spiking Neural Network</td>
</tr>
<tr>
<td>BIBO</td>
<td>Bounded Input Bounded Output</td>
</tr>
<tr>
<td>CD</td>
<td>Coincidence Detector</td>
</tr>
<tr>
<td>deSNN</td>
<td>dynamic evolving Spiking Neural Network</td>
</tr>
<tr>
<td>ED</td>
<td>Element Distinctness</td>
</tr>
<tr>
<td>EIF</td>
<td>Exponential Integrate and Fire</td>
</tr>
<tr>
<td>EPSP</td>
<td>Excitatory Post-Synaptic Potential</td>
</tr>
<tr>
<td>eSNN</td>
<td>evolving Spiking Neural Networks</td>
</tr>
<tr>
<td>EvSpikeProp</td>
<td>Event Based SpikeProp</td>
</tr>
<tr>
<td>EvSpikePropR</td>
<td>Event Based SpikeProp with Robustness to disturbance</td>
</tr>
<tr>
<td>GLM</td>
<td>Generalized Linear Model</td>
</tr>
<tr>
<td>IPSP</td>
<td>Inhibitory Post-Synaptic Potential</td>
</tr>
<tr>
<td>LIF</td>
<td>Leaky Integrate and Fire</td>
</tr>
<tr>
<td>LMS</td>
<td>Least Mean Square</td>
</tr>
<tr>
<td>LNP</td>
<td>Linear Non-Linear Poisson</td>
</tr>
<tr>
<td>MLP</td>
<td>Multilayer Perceptron</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Square Error</td>
</tr>
<tr>
<td>PSP</td>
<td>Post-Synaptic Potential</td>
</tr>
<tr>
<td>QIF</td>
<td>Quadratic Integrate and Fire</td>
</tr>
</tbody>
</table>
Abbreviations and Symbols

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBF</td>
<td>Radial Basis Functions</td>
</tr>
<tr>
<td>ReSuMe</td>
<td>Remote Supervision Method</td>
</tr>
<tr>
<td>RGC</td>
<td>Retinal Ganglion Cell</td>
</tr>
<tr>
<td>RO</td>
<td>Rank Order</td>
</tr>
<tr>
<td>RProp</td>
<td>Resilient Propagation</td>
</tr>
<tr>
<td>SDSP</td>
<td>Spike Driven Synaptic Plasticity</td>
</tr>
<tr>
<td>SNN</td>
<td>Spiking Neural Network</td>
</tr>
<tr>
<td>SOM</td>
<td>Self Organizing Maps</td>
</tr>
<tr>
<td>SpikeProp</td>
<td>Spike Propagation</td>
</tr>
<tr>
<td>SpikePropAd</td>
<td>SpikeProp with adaptive learning rate</td>
</tr>
<tr>
<td>SpikePropAdDel</td>
<td>SpikeProp with adaptive delay learning rate</td>
</tr>
<tr>
<td>SpikePropR</td>
<td>SpikeProp with robustness to disturbances</td>
</tr>
<tr>
<td>SpikePropRT</td>
<td>SpikeProp with robustness to total disturbances</td>
</tr>
<tr>
<td>SRM</td>
<td>Spike Response Model</td>
</tr>
<tr>
<td>STDP</td>
<td>Spike Timing Dependent Plasticity</td>
</tr>
<tr>
<td>TBE</td>
<td>Threshold-based encoding</td>
</tr>
</tbody>
</table>

Symbols

\[ \alpha \] Learning rate for delay adaptation
\[ d_{ji}^{(k)} \] Delay of \( k^{th} \) synaptic connection from \( N_i \) to \( N_j \)
\[ \varepsilon(s) \] Spike response kernel
\[ \eta \] Learning rate for weight adaptation
\[ F_j \] Set of firing times of neuron \( N_j \)
\[ \Gamma_j \] Set of neurons postsynaptic to \( N_j \)
\[ \Gamma_j \] Set of neurons presynaptic to \( N_j \)
\[ I(t) \] Injection current to neuron
\[ \kappa(s) \] Injection current kernel
\[ N_h \] Neuron in hidden layer
\[ N_i \] Neuron in input layer
\[ N_j \] \( j^{th} \) neuron in the set

Nanyang Technological University Singapore
Abbreviations and Symbols

$N_o$ Neuron in output layer

$\nu(s)$ Refractory response kernel

$\mathcal{H}$ Set of hidden layer neurons

$\mathcal{I}$ Set of input layer neurons

$\mathcal{K}$ Set of delayed synaptic connection from $N_i$ to $N_j$

$\mathcal{O}$ Set of output layer neurons

$\bar{t}_o$ Ideal output neuron firing time

$t_j^{(f)}$ Spike firing time of neuron $N_j$

$t_j^{(f-1)}$ Penultimate spike firing time of neuron $N_j$

$t_j^{(f)}$ Spike firing time of a neuron in hidden layer

$t_i^{(f)}$ Spike firing time of a neuron in input layer

$t_o^{(f)}$ Spike firing time of a neuron in output layer

$t_h$ First spike firing time of a neuron in hidden layer

$t_i$ First spike firing time of a neuron in input layer

$t_o$ First spike firing time of a neuron in output layer

$t_f$ Spike firing times of neuron $N_j$ in vector form

$\hat{t}_o$ Target output neuron firing time in training sample

$\Theta(\cdot)$ Heaviside step function

$\vartheta$ Neuron threshold value

$u_{\text{rest}}$ Resting potential of neuron

$u(t)$ Membrane potential of neuron

$w_{ji}^{(k)}$ Weight of $k$th synaptic connection from $N_i$ to $N_j$

$\Box$ Ideal value of parameter $\Box$

$\tilde{\Box}$ $\Box - \Box$
Notation & Convention

Matrices upper case boldface letters (e.g. $Z$)
Vectors lower case boldface letters (e.g. $z$)
Scalars lower case letters (e.g. $z$)
Sets calligraphic letters (e.g. $\mathcal{G}$)
Standard sets upper case blackboard bold characters (e.g. $\mathbb{R}$)

Given a matrix $Z$ its

$i^{th}$ row $z_i$
$i^{th}$ column $z_i$
$(i, j)^{th}$ element $z_{ij}$

Given a vector $z$ its

$i^{th}$ element $z_i$

Given a set $\mathcal{G}$ its

Cardinal number $|\mathcal{G}|$

Matrix-Vector calculus Denominator layout convention (cf. Appendix A)
Chapter 1

Introduction

As engineers, we would be foolish to ignore the lessons of a billion years of evolution.

—Carver Mead, 1993

Natural brain is the benchmark for modern computational systems. In all our endeavors to achieve intelligent learning systems that perform tasks of Artificial General Intelligence (AGI), we are continually trying to match the ability of brain in its simplest forms such as tasks of vision, comprehension, speech etc. Even a young brain can perform such tasks quite accurately and effortlessly. The beauty of brain is that a single system is able to perform all these feats of AGI rather than any specific specialized task. Needless to say, nature has perfected its design of the brain through millions of years of refinement via evolution. The brain consists of massively parallel interconnection of simple computational units. The computational units are called neurons and they are interconnected by information transmission channels called synapse. A single neuron by itself cannot perform sophisticated tasks of intelligence, however, such tremendous intelligence capability of natural brain is due to the highly parallel information exchange between a population of neurons via synapses.

There are different approaches towards achieving AGI. One of the popular computational system in use is Artificial Neural Network (ANN). The basic idea behind ANNs is to mimic the functionality of natural brain using highly parallel and interconnected system of simple abstract model of neurons as computational units. The synapse is represented by weights of the network connection and sometimes delay of the connection as well. Based on the degree of abstraction on the model
of neurons, we can classify ANNs into different generations.

In 1943, McCulloh and Pits formulated the first artificial neuron, famously termed McCulloh-Pits neuron. In it, the neuron is modeled using thresholding function as the *activation function*. The thresholding is performed on the weighted sum of inputs as non-linear operation. This constitutes the *first generation of ANNs* [1].

The examples of first generation ANNs that use McCulloh-Pits neurons are Rosenblatt’s Perceptron, Hopfield nets, Bidirectional Associative Memory, Boltzman machines etc. and are capable of outputting binary outputs only. Theoretically, these neural networks with hidden layer are capable of approximating any Boolean function.

In the *second generation of ANNs* [1], the neuron is modeled using monotonically-increasing continuous piecewise differentiable activation function which is able to process inputs and respond with real-valued outputs. Sigmoid function, hyperbolic tangent function, piecewise linear function, Gaussian function, Chebyshev polynomial etc. are some examples of activation functions used in ANNs of the second generation. The second generation of ANNs are very powerful, in the sense that they can approximate any real valued function to arbitrary precision within a given interval as stated by universal approximation theorem [3,4]. They have also proven their computational prowess through various practical applications and are the current state of the art in machine learning. The examples include Multilayer Perceptron (MLP), Radial Basis Functions (RBF), Self Organizing Maps (SOM), Elman Networks [5,6], convolutional neural networks [7], deep neural networks [8,9] etc.

In the *third generation of ANNs* [1], the neuron is modeled using spiking neurons. Spiking neurons are mathematical model of biological neurons and similar to biological neurons, they process inputs in the form of a sequence of events in time, called *spikes* and produce output response in the form of spikes as well. This is drastically different from the real/binary input output values of the first and second generation of ANNs, however, it is exactly the way in which biological neurons exchange information between them. A Spiking Neural Network (SNN) is an ANN that uses spiking neurons as its computational units. It is less abstract than the ANNs of first and second generation and has increased similarity to the biological network.

We will introduce details of SNN in the following section, followed by motivation behind SNNs. We will then clearly list the objectives of this thesis and out major
contributions. Finally we will end this chapter with an overview of the contents of this thesis.

1.1 Spiking Neural Networks

Spiking Neural Networks are ANNs that have spiking neurons as their computational units. Spiking neurons differ from non-linear activation function used in earlier generation of ANNs in the sense that they exchange information in the form of spikes rather than numeric input-output. The question arises, what are spikes exactly? We have plotted the activity in a visual cortex neuron in figure 1.1.

![Figure 1.1: In vivo membrane potential of visual cortex neuron.](http://medicine.yale.edu/lab/mccormick/data/index.aspx)

The plot shows the readings of the electric potential of the neuron, termed *membrane potential*, over time. We can see that at different instances, the membrane potential suddenly jump by around 100 mV. These jumps are typically of 1-2 ms duration and the pulse event is called *spike*. The nature of spike potential is not important, rather the number of spikes and the time of spikes is important [10]. In SNNs, the inputs to the ANN is spikes and the output is also spikes i.e. the currency of information exchange in SNN is in terms of spikes. This is illustrated in figure 1.2.

Computation in terms of spikes means the information is encoded with respect to time. This brings an additional dimension of time as means of information exchange. Further, the inherent delay in the synapse and the time constant of a spiking neuron response (cf. Chapter 2.2.4) provides a memory feature in built in SNN and the recurrence due to refractory response of a spiking neuron bestows SNN with recurrent structure without really going through the complexity of recurrent neural networks. This makes SNNs potentially very useful, especially

---

1 Courtesy [http://medicine.yale.edu/lab/mccormick/data/index.aspx](http://medicine.yale.edu/lab/mccormick/data/index.aspx)
for the processes that are temporal, such as audio, video etc. The computation in terms of spike, however, adds complexity of encoding the real world numeric inputs into spikes to encode the information in time axis. For this, we need to employ special measures to encode the real world numeric inputs into spikes (cf. Chapter 2.3).

We will now discuss the significance and usefulness of SNNs.

1.1.1 The Biological Inspiration

It has been known to researchers for a long time that the currency of information exchange in biological neurons is spikes. Since SNNs also process and transmit information in the form of spikes, they resemble the biological phenomenon closely, especially compared to the first and second generation of ANNs. The increased similarity to biological phenomenon makes SNNs very useful tool in modelling and analysis of biological neural system [11–15]. This, although not the sole motivation, is an inspiration towards computing in terms of spikes.

We know for a fact that biological neurons exchange information in the form of spikes. The question “How exactly is the information encapsulated by a spike train?” still does not have clear answer. There are two school of thoughts on how the information is exchanged using spike train: rate encoding and pulse encoding. Classically, it was argued that the totality of the information being transmitted is contained by the firing rate of a neuron and the exact timing of spike does not matter. This principle is called rate encoding [1,16–25]. The real numeric inputs and outputs of the first and second generation of neural networks are loosely interpreted as the instantaneous firing rate of the neurons in rate coding principle. However, numerous observations have been made in neuroscience, especially in the visual cortex and auditory neurons, where the neuron response over successive
layers is so swift that the firing rate interpretation is not plausible. The response is so quick that there is not enough spikes along subsequent layers to make an estimate of the firing rate. These findings suggest that the precise timing of spike carries significant amount of information being transmitted [1, 19–26]. The idea that precise time of spikes carry information is termed pulse encoding. Since few spikes can transmit useful information, SNNs can potentially process the information in efficient and rapid manner which makes SNNs an appealing prospect as efficient computational unit.

1.1.2 Significance of SNN as a Computational Unit

During the early years of research on SNN as a computational unit, Wolfgang Maass conducted several theoretical analysis on the complexity of computing with spiking neuron. For his analysis, he mostly considered simple abstract spiking neurons, namely Type A and Type B neurons [1, 20, 21, 27] (cf. Chapter 2.2.5).

Table 1.1: Computational capacity of SNN with Type A neuron on Boolean functions

<table>
<thead>
<tr>
<th>Boolean function</th>
<th>No of hidden layer neurons required</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CD_n(x, y)$</td>
<td>SNN†</td>
</tr>
<tr>
<td>$CD_n(x, y)$</td>
<td>$1$</td>
</tr>
<tr>
<td>$ED_n(x)$</td>
<td>$1$</td>
</tr>
</tbody>
</table>

† No hidden layer in SNN.

Table 1.1 summarizes the theoretical results on computational capacity of SNNs on Boolean function deduced by Wolfgang Maass [1]. We can see that a single SNN can achieve the Boolean results on $CD_n$ and $ED_n$ Boolean functions whereas for threshold based neural network and MLP, we require an ANN with multiple hidden neurons. The functions $CD_n$ and $ED_n$ serve as example Boolean functions to illustrate the usefulness of SNN. In addition, we have the following theoretical result as well:

Any threshold circuit of $s$ gates having real valued inputs from $[0, 1]^n$ can be simulated by a network of $O(s)$ spiking neurons of Type B.

[1](pp. 1668)
Since, a threshold circuit with hidden layer can approximate any Boolean function, it follows that SNNs can also approximate any Boolean function. Michael Schmitt [28] extended the theoretical results of Wolfgang Maass further showing the usefulness of SNNs in computing Boolean functions.

In [27], Wolfgang Maass theoretically studied noisy spiking neuron (similar to Spike Response Model. cf. Chapter 2.2.4) with stochastic spiking. He proved the following result:

For any given $\epsilon, \delta > 0$ one can simulate any given feedforward sigmoidal neural net $N$ with activation function $\pi$ by a network $\mathcal{N}_{N,\epsilon,\delta}$ of noisy spiking neurons in temporal coding. [27](pp. 213)

This in conjuction with universal approximation theorem [3,4] means that SNNs are universal approximators. These theoretical results suggest open whole world of possibilities for SNN as a computational unit. The realization of these theoretical potential is, however, in its infancy [17]. Nevertheless, aforementioned theoretical potential renders SNNs a compelling research topic.

### 1.1.3 Neuromorphic Significance

Recently, the research in the field of hardware based neural networks has grained much traction, with the focus on implementing brain-like spike based computation in dedicated hardware [29–33]. These chips are called neuromorphic circuits. Some of the currently popular advanced neuromorphic chips include IBM True North [29], SpiNNaker [31,34] etc.

Apart from studying neuron behaviour and simulating brain functions, these chips are extremely low power devices. Neuromorphic are extremely efficient in terms of chip area savings and energy savings compared to conventional CPUs and GPUs based machine learning networks [35–37]. These developments in neuromorphic chips makes learning using SNN a very exciting field.

### 1.2 Main Contribution

Although SNNs differ from first and second generation of ANNs in the way in which neuron is modeled, the inherent non-linearity and dynamic nature of a spiking neuron means that we cannot directly tap into the broad range of algorithms
that have been developed for first and especially second generation of ANNs. A handful of methods, both supervised and unsupervised, have been developed for training an SNN. We will discuss in detail about these methods in detail in Chapter 3. As far as learning in a multilayer framework using SNN, we are left with SpikeProp [2] and its derivatives and Multilayer extension of ReSuMe [38]. SpikeProp and its derivatives mostly react to the first spike of the neuron only and ignore the subsequent spikes. Therefore, the full potential of efficient information processing using spike train is mostly untapped.

The complex non-linear dynamic behavior of spiking neuron means that stability and convergence issues are of significant importance, more so compared to MLP networks. It is highly desirable to have learning methods that exhibit stable and convergent nature. Since simulating an SNN on a traditional sequential machine is a time consuming process, the assurance of stability and convergence is an important trait to have. In addition faster learning is also immensely valuable. Stability, convergence and faster learning is even more significant asset for leaning methods dedicated to multiple spiking SNN.

The aim of this thesis is to design and develop fast, stable and convergent methods to learn in an SNN with multilayer architecture, first for single spiking variant and extend the concept to achieve similar results for multiple spiking SNN as well. The main contributions of this thesis are summarizes as follows.

1. An adaptive learning rate rule for SpikeProp based on weight convergence criterion (SpikePropAd) is proposed [39,40]. The algorithm reinforces SpikeProp learning with weight convergence based on Lyapunov criterion and the adaptive learning rate is determined based on the weight convergence criterion. The weight convergence results ensures more successful learning instances and the adaptive learning rate means that the learning is substantially faster than the original SpikeProp method.

2. An adaptive learning rate rule for delay learning extension of SpikeProp based on delay convergence criterion (SpikePropAdDel) is proposed [41]. The delay convergence is based on Lyapunov criterion and the adaptive delay learning rate is deduced from the delay convergence condition. The adaptive learning rate for weight from SpikePropAd and the adaptive learning rate for delay constitute SpikePropAdDel. The weight and delay convergence means that there is more success in learning. Since we can tune delay as well, the learning is faster compared to learning weights only using SpikePropAd and
1.3. Other Current Research in SNN

Apart from computational unit, the research in SNN are mostly in neuroscience in trying to understand and interface biological neurons. They are of great importance in neuroscience as a useful tool to model brain functionality. There have been high profile efforts to use SNN to simulate brain network, viz. The Blue Brain project [11]. SNNs have also been used to understand the working of small scale natural system such as a cricket neural system [14]. Other research area in SNN include memory models [12, 13], neuroprosthetics and brain machine interface [46, 47] etc.

Usually, simulating an SNN requires tremendous resources and time. Lately, there have been numerous effort for efficient simulation of spiking neural networks using CPU & GPU [11, 48–52] and special neuromorphic circuits as well [29, 30, 53].
1.4 Overview of Thesis

This thesis consists of nine chapters. The contents of this is organized as follows.

In **this Chapter**, we have introduced SNN, the motivation behind research in SNN, our main contribution and current state of research in SNN.

In **Chapter 2**, we will go through the basic definitions and fundamentals of spiking neuron. We will introduce various mathematical model of spiking neuron and discuss their advantages and disadvantages. We will also discuss about how the information is encoded in spikes and how the output spikes are interpreted.

In **Chapter 3**, we will elaborate on existing learning methods for supervised learning in SNN, list their issues and advantages. Our focus will be on learning methods that can train an SNN with hidden layer and we will describe those learning methods in detail.

In **Chapter 4**, we will explain the fundamentals of non-linear stability theory and some key concepts that we will make use of later in the thesis. We will first discuss key mathematical ideas before we explain the stability methods. Our focus in Chapter 4 will be on Lyapunov stability and conic sector stability.

In **Chapter 5**, we will introduce adaptive learning rate algorithm, SpikePropAd, as an extension of SpikeProp. The adaptive learning rate is based on weight convergence analysis of SpikeProp and guarantees convergence in terms of weight parameter.

In **Chapter 6** we continue on the same concept for adaptive learning rate for delay learning, SpikePropAdDel, this time using delay convergence analysis of delay learning extension of SpikeProp.

In **Chapter 7**, we will derive robust stability results along with convergence condition based on total error and individual error: SpikePropRT and SpikePropR learning algorithms. The robust stability means that the learning process is stable even in presence of external disturbance and weight convergence means that learning converges in terms of weight parameter.

In **Chapter 8**, we will formalize event based weight update rule, EvSpikeProp, for learning spike train. We will also perform convergence and stability analysis of EvSpikeProp to develop enhanced event based learning rule– EvSpikePropR – in Chapter 8.
Finally in Chapter 9, we will summarize this thesis and recommend future research directions.
Chapter 2

Spiking Neuron Basics

Your brain is built of cells called neurons and glia – hundred and billions of them. Each one of these cells is as complicated as a city.

—David Eagleman, 2012

The main component that differentiates a Spiking Neural Network from the previous generation of Artificial Neural Networks is the computational unit, which is a spiking neuron rather than some functional unit governed by activation function. Formally, a Spiking Neural Network is defined as:

**Spiking Neural Network** — A Spiking Neural Network is a massively parallel distributed system made up of individual computational units comprising of spiking neurons which processes its inputs in the form of spike event and responds with spike event. The knowledge is stored in the synaptic weights and synaptic delays of the interneuron connection and is acquired by the network from its environment through a learning process.

and a spiking neuron is defined as:

**Spiking Neuron** — A spiking neuron is a mathematical model that describes the dynamics of a biological neuron, its response to incoming spike events through different synapses and is able to describe spike events, either directly or indirectly.

In this chapter, we will discuss different mathematical models of a biological neu-
rons that can be used as the computational unit in the form of spiking neuron in an SNN. The mathematical models of neuron are the product of researches in Computational Neuroscience pertaining to electrophysiological behaviour of a neuron. In this chapter, we will discuss the dynamics of a biological neuron. Next, we will discuss about different spiking neuron models with focus on their biological plausibility as well as their ease of implementation in large scale. A spiking neuron responds to a spike and reacts with a spike event. However, the real word signals are numeric valued. We will discuss different school of thoughts about the numeric interpretation of spikes and how real world data can be represented in terms of spikes.

2.1 Dynamics of a Biological Neuron

We can see the plot of electrophysiological reading of a visual cortex neuron in figure 1.1. The voltage value of the neuron is called membrane potential, usually denoted by \( u(t) \), and the distinct short surges in membrane potential occurring from time to time are the spikes. Before we proceed to describe the dynamics of a biological neuron, we will define some formal terms:

**Spike or Action Potential** — A spike or an action potential is a brief electric pulse in membrane potential, typically 1 to 2 ms in duration and an amplitude of about 100 mv which roughly look alike and is usually followed by a dip in membrane potential, that occurs in regular or irregular intervals. The action potential is the elementary unit of signal transmission [10].

**Spike Train** — A chain of action potentials emitted by a single neuron is called a spike train — a sequence of stereotyped events which occur at regular or irregular interval [10].

**Synapse** — The site where the axon of a presynaptic neuron makes contact with the dendrite (or soma) of a postsynaptic neuron cell is the synapse [10].

In the event of spike at a presynaptic neuron, synapse release neurotransmitters from the presynaptic terminal into the synaptic cleft which creates an imbalance in ionic concentration. The imbalance in ionic concentration effects the membrane potential of postsynaptic neuron such that the postsynaptic neuron is more likely
to emit a spike or less likely to emit a spike depending upon whether the synapse is excitatory synapse or inhibitory synapse [10,25]. The resulting effect of an input spike on postsynaptic neuron is called Post-Synaptic Potential (PSP). The PSP via excitatory synapse is called Excitatory Post-Synaptic Potential (EPSP) which typically increases the membrane potential above its resting potential, usually denoted by $u_{rest}$. The resting potential in biological neurons is typically around -70 mV. We can alternatively measure the membrane potential with reference to resting potential as well and consider 0 mV as resting potential. Similarly, the PSP via inhibitory synapse is called Inhibitory Post-Synaptic Potential (IPSP) which decreases the membrane potential below resting potential. A typical EPSP and IPSP response of a biological neuron is shown in figure 2.1. The magnitude of PSP depends on the synaptic weight or synaptic efficacy and the immediacy of PSP after spike is determined by synaptic delay which is dependent on the length of the synapse that the spike needs to travel. The cumulative effect of all the PSPs from presynaptic neuron forms the pre-spike membrane potential.

When the pre-spike membrane potential increases from resting potential high enough, usually called threshold value, the linear accumulation of PSP breaks down and it exhibits a pulse like excursion, called spike or Action Potential (AP) [10]. After spike, the neuron membrane potential does not return to resting potential. Instead, the membrane potential passes through a phase of hyperpolarization below the resting value. This hyperpolarization is called After-Hyperpolarization Potential (AHP). The AHP is due to what is called refractory response of the neuron. There is a certain amount of time immediately after the action potential when the neuron is unable to fire at any cost. This period is called absolute refractory period. After absolute refractory period, the neuron still hesitates to fire, but is able to fire if the cumulative PSP is high enough. This phase is known
as relative refractory period. This behavior of a biological neuron is depicted in figure 2.2.

Figure 2.2: Dynamics of a biological neuron.

2.2 Models of Spiking Neuron

A spiking neuron tries to describe the dynamic behavior of a biological neuron mathematically with varying degree of detail. We will discuss different mathematical models suitable for this purpose next.

2.2.1 Hodgkin-Huxley Model

Alan L Hodgkin and Andrew F Huxley made a detailed study of action potential in Squid Giant axon and based on their ionic conduction model, they came up with what is now famously known as Hodgkin-Huxley model of neuron. They described the ionic conduction channel with various effective conductance model for Sodium and Potassium ion channel with each of the variable conductance separately modeled via their own set of non-linear differential relation. The electrical
model that Hodgkin-Huxley proposed for modeling the Squid Giant axon is illustrated in figure 2.3.

![Hodgkin Huxley model based on ion channels.](image)

The membrane potential, $u(t)$, is described using following set of equations [54]:

$$C \frac{du}{dt} = -g_{Na} m^3 h (u - E_{Na}) - g_K n^4 (u - E_K) - g_{leak} (u - E_{leak}) + I(t)$$  \hspace{1cm} (2.1a)

$$\frac{dm}{dt} = -\frac{m - m_0(u)}{\tau_m(u)}$$  \hspace{1cm} (2.1b)

$$\frac{dh}{dt} = -\frac{h - h_0(u)}{\tau_h(u)}$$  \hspace{1cm} (2.1c)

$$\frac{dn}{dt} = -\frac{n - n_0(u)}{\tau_n(u)}$$  \hspace{1cm} (2.1d)

where

- $C$ is capacitance of the membrane
- $g_{Na}$, $g_K$ and $g_{leak}$ are the conductance parameters for different ion channels
- $E_{Na}$, $E_K$ and $E_{leak}$ are the equilibrium potentials of ion channels
- $m$, $n$ and $h$ describe the opening and closing of voltage dependent channels
- $I(t)$ is the injection current

For more complex neurons with additional ion channels, one can simply include additional conductance term for it in (2.1). With suitable model for conductance parameters and channel gate functions, Hodgkin-Huxley model provides very realistic results. It is for their remarkable modeling of physiological behavior of
neuron, Hodgkin and Huxley were awarded Nobel Prize in Physiology or Medicine in 1963.

Despite being very accurate model of a neuron, the different non-linear dynamic interrelationship in the coupled system of differential equation makes it very difficult to efficiently solve the system of differential equation, especially when it comes to solving it for a large network of neurons interacting with each other. Therefore, Hodgkin-Huxley model is not suitable in modeling a population of spiking neurons.

2.2.2 Two Dimensional Neuron Model

The complexity of Hodgkin-Huxley model can be simplified for practical purposes exploiting the similarities between ionic conductance parameters and using the idea of separation of time scale. These simplification reduce the complex formulation of Hodgkin-Huxley model into a system of two differential equations which can be generalized into following form [10].

\[ C \frac{du}{dt} = f(u, w) + I(t) \]  
\[ \frac{dw}{dt} = g(u, w) \]

Where, \( w(t) \) is the membrane recovery variable.

There are a handful two dimensional model of neuron, each with diverse origin. However, they follow the same system of equation described in (2.2). Some of the two dimensional models are listed below:
2.2. Models of Spiking Neuron

FitzHugh-Nagumo model [55]

\[ f(u, w) = u - \frac{u^3}{3} - w, \quad C = 1 \]
\[ g(u, w) = 0.08(u + 0.7 - 0.8w) \]

Morris Lecar model [56]

\[ f(u, w) = -g_{Ca} M_{ss}(u)(u - E_{Ca}) - g_K w(u - E_K) - g_L (u - E_L) \]
\[ g(u, w) = \frac{W_{ss}(u) - w}{T_w(u)} \]

Izhikevich Model [57]

\[ f(u, w) = 0.04 u^2 + 5 u + 140 - w, \quad C = 1 \]
\[ g(u, w) = a(bu - w) \]

spike if \( u \geq 30 \) mV, then \( \begin{cases} u \leftarrow c \\ w \leftarrow w + d \end{cases} \)

AdEx

Adaptive exponential integrate-and-fire model [58]

\[ f(u, w) = -g_L (u - E_L) + g_L \Delta T \exp \left( \frac{u - \vartheta}{\Delta_T} \right) - w \]
\[ g(u, w) = \frac{a(u - E_L) - w}{\tau_w} \]

spike if \( u \geq \vartheta \), then \( \begin{cases} u \leftarrow u_{\text{rest}} \\ w \leftarrow w + d \end{cases} \)

These two dimensional model of neuron are simple enough to be practically implement for a population of spiking neurons and are able to exhibit complex dynamic behavior seen in biological neurons.

2.2.3 One Dimensional Neuron Model

For even more scalable simulation, the two dimensional model of neuron can be further simplified into single differential equation. This is called one dimensional neuron model which takes the following general form:

\[ \tau \frac{du}{dt} = f(u) + RI(t) \]  \hspace{1cm} (2.3)

spike if \( u = \vartheta \), then \( u \leftarrow u_{\text{rest}} \)

The function \( f \) can take different form resulting in \textit{Leaky Integrate and Fire (LIF)} as linear formulation and \textit{Nonlinear Integrate Fire} model. They are listed below:
Leaky Integrate and Fire (LIF) \([4, 10, 25]\)  
\[ f(u) = -(u - u_{\text{rest}}) \]

Exponential Integrate and Fire (EIF) \([59]\)  
\[ f(u) = -(u - u_{\text{rest}}) + \Delta \exp\left(\frac{u-\vartheta}{\Delta}\right) \]

Quadratic Integrate and Fire (QIF) \([10]\)  
\[ f(u) = c_2(u - c_1)^2 + c_0 \]

These one-dimensional model of neuron are very simple and easy to implement, however, their major drawback is that these models do not demonstrate the refractory behavior of neuron which is a major neuronal behavior. Nevertheless, they are reliable in predicting the response spike to input spikes barring very nuanced behavior \([60]\).

### 2.2.4 Spike Response Model

The previous models of neuron were based on modeling the membrane dynamics of biological neuron. Spike Response Model (SRM) on the other hand is based on characterization of behavior observed in biological neuron. It was introduced by Gerstner \([61]\). This model of neuron is based on linear filters corresponding to PSP, refractory response and injection current.

Consider a neuron \(N_j\) which is receiving input spikes from a set of presynaptic neurons \(\Gamma_j = \{i : N_i \text{ is presynaptic to } N_j\} = \mathcal{I}\). Denote the set of delayed synaptic connection by \(\mathcal{K}\) the synaptic weight of \(k\)th synapse, where \(k \in \mathcal{K}\), from \(N_i\) to \(N_j\) by \(w_{ji}^{(k)}\), the synaptic delay of \(k\)th synapse, where \(k \in \mathcal{K}\), from \(N_i\) to \(N_j\) by \(d_{ji}^{(k)}\), the set of previous firing times of neuron \(N_j\) by \(\mathcal{F}_j = \{t_{j(1)}, t_{j(2)}, \ldots, t_{j(f-1)}\}\) and the injection current input to neuron \(N_j\) by \(I(t)\). Then the pre-spike membrane potential is modeled as:

\[
u_j(t) = \sum_{t_{j(f-1)} \in \mathcal{F}_j} \nu(t - t_{j(f-1)}) + \sum_{i \in \Gamma_j} \sum_{t_{i}^{(f)} \in \mathcal{F}_i} \sum_{k} w_{ji}^{(k)} \varepsilon(t - t_{i}^{(f)} - d_{ji}^{(k)}) + \int_{0}^{\infty} \kappa(r) I(t - r) \, dr \tag{2.4}\]

where \(\nu(s)\) is the refractory response kernel which models potential reset after spike, \(\varepsilon(s)\) is the spike response kernel which models normalized PSP and \(\kappa(s)\) is the injection current kernel which models the response to injection input current.
The commonly used SRM kernels are (see \cite{2,25,61,63,64}):

\[
\nu(s) = \begin{cases} 
-\vartheta e^{-\frac{s}{\tau}} \Theta(s) & \text{OR} \\
-v_0 e^{-\frac{s-\delta_{\text{abs}}}{\tau}} \Theta(s - \delta_{\text{abs}}) - K \Theta(s) \Theta(\delta_{\text{abs}} - s)
\end{cases}
\]

\[
\varepsilon(s) = \begin{cases} 
\frac{s}{\tau_s} e^{-\frac{s}{\tau}} \Theta(s) & \text{OR} \\
\frac{s}{\tau_s} e^{1-\frac{s}{\tau}} \Theta(s) & \text{OR} \\
(e^{-\frac{s}{\tau_m}} - e^{-\frac{s}{\tau}}) \Theta(s), & 0 \leq \tau_s \leq \tau_m
\end{cases}
\]

\[
\kappa(s) = e^{-\frac{s}{\tau}} \Theta(s)
\]

where \(\Theta(\cdot)\) is the Heaviside step function and \(\tau\) is the time constant corresponding to the kernels. Different form of spike response kernel, \(\varepsilon(s)\), is possible for excitatory and inhibitory synapse. It is common to represent inhibitory synapse by negative weight and use the same spike response kernel.

When the membrane potential rises up to a threshold value \(\vartheta\), \(N_j\) emits a spike at \(t = t_j^{(f)}\) i.e.

\[
t_j^{(f)} : u_j(t_j^{(f)}) = \vartheta, \quad u_j'(t_j^{(f)}) > 0, \quad t_j^{(f)} > t_j^{(f-1)}
\]  \hspace{1cm} (2.5)

SRM model of neuron, although seems simple, is quite general. With proper response kernels, it can simulate very complex neuronal behavior observed in biological neurons \cite{10,25}. In addition, this neuron model is rather straightforward to implement as well.

Denote the input spikes from neuron \(N_i\) in vector form as \(t_i\).

The sum of normalized PSP response from neuron \(N_i\) is usually denoted as

\[
y_{ji}^{(k)} (t_i, t_j) = \sum_{t_{ji}^{(f)} \in F_i} \varepsilon(t - t_{ji}^{(f)} - d_{ji}^{(k)})
\]  \hspace{1cm} (2.6)

Denote the set of all the weights of \(N_j\) by

\[
w_{ji} = [\cdots, w_{ji}^{(k)}, \cdots]^T \in \mathbb{R}^{[\ell][\kappa]}
\]  \hspace{1cm} (2.7)
and the corresponding spike response vector by

\[ y_{ji}(t, t_i) = [\cdots, y_{ji}^{(k)}(t, t_i), \cdots]^T \in \mathbb{R}^{2|K|} \]  

(2.8)

Then the Spike Response Model can be succinctly written as

\[ u_j(t) = \sum_{t_j^{(f-l)} \in \mathcal{F}_j} \nu(t - t_j^{(f-l)}) + w_j^T y_{ji}(t, t_i) + \int_0^\infty \kappa(r) I(t - r) \, dr \]  

(2.9)

\[ t_j^{(f)} : u_j(t_j^{(f)}) = \vartheta, \quad u_j'(t_j^{(f)}) > 0, \quad t_j^{(f)} > t_j^{(f-l)} \]

Usually, the refractory response kernel, \( \nu(s) \), and the spike response kernel, \( \varepsilon(s) \), die out with time. The refractory response of the most recent neuron spike, denoted by \( t_j^{(f-l)} \), is the dominant refractory kernel. Further, the effect of any input spike before the last neuron spike is also insignificant. Therefore, we can reduce the set of input spikes from presynaptic neuron by including additional restriction as follows.

\[ \mathcal{F}_i = \{ t_i^{(f)} : i \in \Gamma_j \text{ & } t_i^{(f)} + d_i^{(k)} > t_j^{(f-l)} \} \]  

(2.10)

This results in short term memory variant of SRM, called Short-term Memory neuron [25].

\[ u_j(t) = \nu(t - t_j^{(f-l)}) + w_j^T y_{ji}(t, t_i) \]  

(2.11)

\[ t_j^{(f)} : u_j(t_j^{(f)}) = \vartheta, \quad u_j'(t_j^{(f)}) > 0, \quad t_j^{(f)} > t_j^{(f-l)} \]

Further, if we consider only the first spike of the neuron, the refractory response is vestigial. This is called SRM\(_0\) model [2, 25].

\[ u_j(t) = w_j^T y_{ji}(t) \]  

(2.12)

\[ t_j^{(f)} : u_j(t_j^{(f)}) = \vartheta, \quad u_j'(t_j^{(f)}) > 0 \]

### 2.2.5 Type A and Type B Spiking Neuron

For the study of computational capacity of spiking neurons, Wolfgang Maass [1] used spiking neuron model with simplified spike response function and variable
threshold to represent refractoriness. Mathematically, they are defined as follows.

\[ u_j(t) = \sum_{i \in \Gamma_j} w_{ji} \varepsilon(t - t_i^{(f)} - d_{ji}) \quad (2.13) \]

\[ t_j^{(f)} : u_j(t_j^{(f)}) = \theta(t - t_j^{(f-1)}) \]

Type A and Type B spiking neuron differ in their spike response function. The former has simple pulse function as spike response function and the latter has triangular function as spike response function. This is depicted in figure 2.4.

![Type A and Type B spiking neurons](image)

Figure 2.4: Type A and Type B spiking neurons as defined in [1] (a) spike response function for Type A spiking neuron (b) spike response function for Type B spiking neuron (c) variable threshold for refractoriness.

These simple models of spiking neuron have been used to show various computational prowess of spiking neural network (cf. Chapter 1.1.2).

### 2.3 Spike Coding

The currency of information exchange in an SNN is spike. On the other hand, the real world signals and data are numeric values. To use an SNN as a computational unit, it is of paramount importance to convert the real world signals into spikes and back.

It is clear that spike train carry information between neurons. However, what aspect of spike train is exactly responsible for information transfer is not very clear. There are two schools of thought: rate encoding and pulse encoding (cf. Chapter 1.1.1). Rate encoding assumes that the information is borne by the firing rate of the spike train, whereas pulse encoding advocates that precise timing of each individual spike carries information.

For interpretation of output spikes, winner-takes-all scheme for classification i.e. whichever neuron fires first is the associated class or classification based on more
active neuron i.e. whichever neuron shows more spike activity is the associated class are plausible methods of interpreting output spikes.

As far as encoding numeric signals into spikes, traditional method in neuroscience is to consider spikes as a Poisson process with instantaneous firing rate modulated by input signal \[16\]. A Linear Non-Linear Poisson (LNP) model based on analysis of LIF and EIF neuron model produces better spike encoding compared to Poisson encoding \[65\]. Even more accurate spike generation has been proposed by Pillow et al. using spiking neuron model: generalized integrate and fire model \[66\] and Generalized Linear Model (GLM) \[67\].

For computational purposes, in case of binary inputs the signal can be encoded using earlier or later spike. The real inputs on the other hand need special treatment. A simple method is to use interspike interval to represent the numeric data. The methods described above that are used in neuroscience are also suitable for encoding numeric data. Other methods include Threshold-based encoding (TBE) \[68,69\], adaptive response method \[70\] which encode numeric time series into spike trains. However, these methods produce a spike train. The issue is that most of the methods for learning in multilayer spiking neural network cannot handle multiple spikes. Population encoding \[2\] is an effective method for encoding numeric signal values into single spike time for a group of neurons. Population encoding and Threshold based encoding are described below.

### 2.3.1 Population Encoding

In population encoding, an array of Gaussian receptive fields that are uniformly separated with overlapping profile along the input data range is used to represent input neurons. Different input value have different strength of response in the receptive fields. If the strength of response is high for a neuron’s receptive field, earlier spike is encoded for it and if the strength of response is low for a neuron’s receptive field, later spike is encoded for it. If the strength of response is very low, no spike is encoded for the corresponding neuron \[2\]. The spike times are rounded to nearest time unit.

In figure 2.5, the process of population encoding is demonstrated using four Gaussian receptive fields to encode the input numeric into spikes from time \( t = 0 \) to 4 units.
Bohte et al. [2] suggest distributing the center of Gaussian receptive fields at

$$\mu_i = I_{\text{min}} + \frac{2i - 3}{2} \frac{I_{\text{max}} - I_{\text{min}}}{m - 2}, \quad i = 1, 2, \cdots, m$$  \hspace{1cm} (2.14)

and their spread defined by

$$\sigma = \frac{1}{\beta} \frac{I_{\text{max}} - I_{\text{min}}}{m - 2}$$  \hspace{1cm} (2.15)

where $\beta$ controls the amount of spread, the input numeric is bounded by the range $[I_{\text{min}}, I_{\text{max}}]$.

The population encoding scheme work similar to the Retinal Ganglion Cell (RGC) in eye and produces sparse encoding [2].

### 2.3.2 Threshold Based Encoding

Threshold based encoding (TBE), in contrast to population encoding, encodes the information in a continuous signal into on and off spikes rather than encoding the information over a population of neurons. In its basic form, a threshold is set in log-scale i.e. the threshold value changes by an order of magnitude. Whenever the change in input data exceeds the threshold, a spike is generated. If the change is increasing, an on spike is generated and if the change is decreasing, an off spike is generated. This process is depicted in figure 2.6. The green signal follows the input signal until it crosses threshold. Whenever it crosses the threshold, the signal reference is reset and an on or off spike is generated.
This simple mechanism is able to track the changes in input signal quite well. Although it cannot track the finer nuances in the input, the overall changes is represented quite well. Therefore, it suits quite well to convert temporal signals into spike using TBE. In fact same principle is used to encode real signal into spikes in Silicon Retina [71] and Silicon Cochlea [72,73] which produce the sensor readings directly in terms of spikes.
Chapter 3

Supervised Learning in SNN: An Overview

Networks of noisy spiking neurons with temporal coding have have a strictly larger computational power than a sigmoidal neuron with same number of units —Wolfgang Maass, 2000

There are abundant learning algorithms to train an MLP. One might think that we can port the learning algorithms for MLP in the realm of SNN and use them for SNN training with little or no modification. However, since spiking neurons have far more complex dynamics than continuously differentiable monotonically increasing activation function, we face a major hurdle while trying to take derivative of input-output functional of a spiking neuron. Almost all the learning algorithms for learning an MLPs requires the use of derivative, therefore, formulating the derivative of input-output functional of a spiking neuron is necessary to be able to use backpropagation based learning rules in context of SNNs. The inherent discontinuity in membrane potential and an added aspect of time into the mix pose additional challenges in learning using SNNs. There have been workarounds that circumvent the issues posed by SNNs and put forth a couple of learning algorithms specially tailored for SNNs. In spite of that, learning in SNNs is still a developing area [74]. Some learning methods have been extended from the domain of MLPs while some have been specifically developed for SNNs which utilize the salient features of spiking neurons and arrive at novel learning methods in SNNs [25].
The unsupervised learning methods for SNNs are mostly variants of Hebbian Learning [25]. In neuroscience experiments, it has been observed that change in synaptic strength is a function of relative differences in input and output spike times [75,76]. If the presynaptic spike precedes postsynaptic spike, in general the synaptic strength increases – potentiation and if the postsynaptic spike precedes presynaptic spike, in general the synaptic strength decreases – depression. This phenomenon is called Spike Timing Dependent Plasticity (STDP) in neuroscience. STDP is the basis of Hebbian learning in SNN. Sometimes, the reverse behaviour to STDP takes place. This is called anti-STDP or anti-Hebbian plasticity [77]. Hebbian learning based unsupervised learning methods for SNN include Multi-layer RBF for SNNs by Bohte et al. [63] and SOM for SNN by Ruf et al. [22]. Gerstner et al. [12] and Zamani et al. [13] use associative memory consisting of spiking neurons which act as unsupervised learning unit for SNNs.

The supervised learning for SNNs can be categorized into various groups. We will describe them along with their pros and cons below.

The first group of supervised learning method are the methods that train a single spiking neuron to learn a spike train. The methods include Remote Supervised Method (ReSuMe) [78,79], Chronotron [80], Spike Pattern Association Neuron (SPAN) [81,82], precise spike driven synaptic plasticity rule [83], Linear Algebraic Method [84] etc. These set of learning methods learn the weights parameter of a spiking neuron, given the set of input spike trains and output spike train. Some of these methods may also learn other parameters of a spiking neuron such as synaptic delay and firing threshold. These methods are usually independent of the spiking neuron model, which makes them versatile. These methods learn the parameters in batch mode, meaning complete set of input spike trains and desired output spike train need to be presented for learning. This makes them not well suited for learning in sequential fashion for indefinite time. Nevertheless, they can be used in mini-batch mode in such a scenario. Although, these learning methods learn weights for a single neuron, extension of ReSuMe called Multi-ReSuMe [38] has been forked to learn in a multilayer architecture as well.

The next group of supervised learning method for SNNs fall into the category of reservoir computing methods. The methods include echo state network and liquid state machine [85–87]. Here, a pool of neurons with random interconnections between them is used. These neurons can even have self recurrence as well. The pool of neurons are referred to as reservoirs. The neurons in the reservoir receive spikes from input neurons. For output, a neuron randomly receives input from
the reservoir neurons and its weight parameter is learned so that it emits output spike of desired characteristics. The output neuron is usually trained using one of the methods mentioned above i.e. SNN method for training a single neuron. As far as the reservoir is concerned, the neurons in this pool can also tune their weights using unsupervised STDP rule. The advantage of this method is that same reservoir network can be used to learn other patterns by simply adding an output neuron and connecting it to the reservoir neuron and continuous stream of spikes are handled naturally. Their major drawback, however, is that there is little control over how the reservoir is created as it is based on random connections and there is no general rule for designing a good reservoir. The recipe for creating a good reservoir is an acquired skill via experience and how they work is unclear.

Next group of supervised learning methods for SNNs are the class of learning methods known as evolving Spiking Neural Networks (eSNN) [88]. The basic eSNN adds a neuron in the output layer on the fly for each new input pattern. The weights are determined by Rank Order (RO) coding, which uses the first input spike time to determine weight. The new neuron is merged to a pre-existing neuron if its acquired weight is similar to a pre-existing neuron, based on some similarity metric. When neurons are merged, their weights are averaged. This makes it vary fast learning method, requiring one pass only. However, the weight of a neuron does not change after that as a result it ignores the subsequent spikes at the input. The extensions to eSNN allow for adaptation of weights as a response to subsequent input spikes [89–92]. They differ on the manner in which weights are tuned for subsequent input spikes, for e.g., dynamic evolving Spiking Neural Network (deSNN) [89] uses Spike Driven Synaptic Plasticity (SDSP) rule whereas Basis Coupled Evolving Spiking Neural Network (BCESNN) uses basis coupled rank order learning to fine tune the weights.

A special structured network of spiking neurons with spatially influenced connection between them has also been developed. It is called NeuCube [93]. The spatially arranged structure makes it suitable for spatial data. The NeuCube structure is similar to reservoir network in reservoir computing, with additional structural arrangement and no recurrence. In NeuCube, the output neuron is added using deSNN rule. It makes NeuCube suitable for spatio-temporal data.

Another group of learning method for SNN are SpikeProp [2] and its derivatives viz. Extended SpikeProp [94], SpikeProp for multiple spiking neuron [17,64,95], Resilient Propagation (RProp) [96] etc. These methods allow learning in a multi-layer architecture, similar to MLPs. Although, the presence of hidden layer gives
them computational edge, their major drawback is that they are mostly limited to a single spike. In addition they also have issue of stability issues as well. Nevertheless, they are the among the very few SNN learning methods that can learn in a multilayer neural network architecture, making use of hidden layer.

Among the supervised learning methods listed above, SpikeProp, its derivatives and Multilayer extension of ReSuMe [38] can deal with learning in an SNN with hidden layer. Reservoir computing methods do have hidden layer neurons, however, the learning takes place in output layer only. We will discuss these methods in detail in the following sections. But before that, we will formally describe the neuron architecture and speed-up techniques during forward simulation of SNN.

### 3.1 Neural Network Architecture

![SNN Architecture](image)

*Figure 3.1: SNN architecture with $|K|$ delayed synaptic connections, $|I|$ input layer neurons, $|H|$ hidden layer neurons and $|O|$ output layer neurons.*
In this thesis we will deal with three-layer-fully-connected feedforward neural network architecture. Each connection between two neurons consists of multiple delayed synaptic connections. Such a network is illustrated in figure 3.1. Here $\mathcal{I}$, $\mathcal{H}$ and $\mathcal{O}$ denotes the set of input layer neurons, hidden layer neurons and output layer neurons. Furthermore, $\mathcal{K}$ denotes the set of delayed synaptic connection. Therefore, there are $|\mathcal{I}|$ input layer neurons, $|\mathcal{H}|$ hidden layer neurons, $|\mathcal{O}|$ output layer neurons and $|\mathcal{K}|$ delayed synaptic connections.

$N_i^{(\mathcal{I})} : i \in \mathcal{I}$ denotes a neuron in input layer, $N_h^{(\mathcal{H})} : h \in \mathcal{H}$ denotes a neuron in hidden layer and $N_o^{(\mathcal{O})} : o \in \mathcal{O}$ denotes a neuron in output layer. For the sake of brevity, we will denote a neuron in input layer, hidden layer and output layer by $N_i$, $N_h$ and $N_o$ respectively. We will refer to the detailed notation only when there is ambiguity in meaning. Similarly, the firing time of $N_i$, $N_h$ and $N_o$ is denoted by $t_i^{(f)}$, $t_h^{(f)}$ and $t_o^{(f)}$ respectively. If we are only considering the first spike of the $N_i$, $N_h$ and $N_o$, which is common in SpikeProp as we will see in Section 3.3, the first spike time is denoted by $t_i$, $t_h$ and $t_o$ respectively.

For $k^{th} : k \in \mathcal{K}$ synaptic connection between $N_i$ and $N_h$, synaptic weight is denoted by $w_{hi}^{(k)}$ and synaptic delay is denoted by $d_{hi}^{(k)}$. Similarly, for $k^{th}$ synaptic connection between $N_h$ and $N_o$, synaptic weight is denoted by $w_{oh}^{(k)}$ and synaptic delay is denoted by $d_{oh}^{(k)}$. All the weights associated with neuron $N_h$ and $N_o$ represented in vector form is defined as

\[
\mathbf{w}_{hl} = [\cdots, w_{hi}^{(k)}, \cdots]^T \in \mathbb{R}^{|\mathcal{I}||\mathcal{K}|}
\]

\[
\mathbf{w}_{oh} = [\cdots, w_{oh}^{(k)}, \cdots]^T \in \mathbb{R}^{|\mathcal{H}||\mathcal{K}|}
\]

respectively and all the delays associated with neuron $N_h$ and $N_o$ represented in vector form is defined as

\[
\mathbf{d}_{hl} = [\cdots, d_{hi}^{(k)}, \cdots]^T \in \mathbb{R}^{|\mathcal{I}||\mathcal{K}|}
\]

\[
\mathbf{d}_{oh} = [\cdots, d_{oh}^{(k)}, \cdots]^T \in \mathbb{R}^{|\mathcal{H}||\mathcal{K}|}
\]

respectively. Next, the weight matrix associated with hidden layer neurons and output layer neurons represented in matrix form is defined as

\[
\mathbf{W}_{HI} = [\cdots, \mathbf{w}_{hl}, \cdots]^T \in \mathbb{R}^{|\mathcal{H}| \times |\mathcal{I}| |\mathcal{K}|}
\]

\[
\mathbf{W}_{OH} = [\cdots, \mathbf{w}_{oh}, \cdots]^T \in \mathbb{R}^{|\mathcal{O}| \times |\mathcal{H}| |\mathcal{K}|}
\]
respectively and the delay matrix associated with hidden layer neurons and output layer neurons represented in matrix form is defined as

\[
D_{HI} = [\cdots, d_{hi}, \cdots]^T \in \mathbb{R}^{|H| \times |Z| |K|} \\
D_{OH} = [\cdots, d_{oh}, \cdots]^T \in \mathbb{R}^{|O| \times |H| |K|}
\]

(3.7)

(3.8)

respectively. These weights and delays hold the knowledge acquired by the SNN described in this thesis.

Throughout this thesis, we use \((\bar{\cdot})\) notation to denote the ideal parameters. Therefore, ideal spike value of output neuron is denoted by \(\bar{t}_o^{(f)}\) (\(\hat{t}_o\) if first spike is considered). Further, \(\epsilon_{\text{ext}}\) refers to the additive noise which acts as external disturbance to the system. This external disturbance is not additional disturbance that is added, but rather it is mixed with the desired spike firing time in learning sample, denoted by \(\hat{t}_o^{(f)}\) (\(\bar{t}_o\) if first spike is considered), which is always mixed with noise, i.e.

\[
\bar{t}_o^{(f)} = \bar{t}_o^{(f)} + \epsilon_{\text{ext}}^{o} \\
\hat{t}_o = \bar{t}_o + \epsilon_{\text{ext}}^{o}
\]

(3.9)

(3.10)

We will use this additive model to perform detailed error analysis to prove stability of SNN learning algorithm later in Chapter 7 and Chapter 8. For the time being, one can ignore this disturbance term and continue.

### 3.2 Forward Simulation of SNN

Forward simulation is the first step during training a neural network. It is more important step for SNNs because it is the most taxing step during training process. This is because unlike traditional MLPs spiking neurons do not have closed form solution for their output spike times. As a result one must iteratively find the solution for output spike times of a spiking neuron which means that it requires multiple pass at different time to determine the solution. The general strategy is to perform a linear scan for say \(t = t_{\text{start}} : t_{\text{res}} : t_{\text{stop}}\) and keep on checking for the firing condition \(u_j(t) > \vartheta\). There have been numerous efforts in efficiently simulating large scale spiking neural networks using CPUs and GPUs \([11, 48–52]\). This topic is out of the scope of this thesis. For our purposes, we use step-wise refinement technique on a progressively smaller interval to efficiently evaluate neuron spike
time. This approach is described in Algorithm 1 for evaluating first spike time of a spiking neuron and in Algorithm 2 for evaluating spike train firing times of a spiking neuron.

**Algorithm 1** First firing time of Neuron \( N_j \)

```plaintext
1: function \textsc{CalcTfire}(N_j) 
2: \hspace{1em} t_{\text{step}} \leftarrow \text{INIT\_STEP} \quad \triangleright \text{INIT\_STEP} = 1
3: \hspace{1em} t_j \leftarrow \text{NO\_FIRE}
4: \hspace{1em} repeat
5: \hspace{2em} for \( t \leftarrow t_{\text{start}} : t_{\text{step}} : t_{\text{stop}} \) do
6: \hspace{3em} calculate \( u_j(t) \)
7: \hspace{3em} if \( u_j(t) \geq \vartheta \) then
8: \hspace{4em} \hspace{1em} t_j \leftarrow t
9: \hspace{4em} \hspace{1em} t_{\text{start}} \leftarrow t - t_{\text{step}}
10: \hspace{4em} \hspace{1em} t_{\text{stop}} \leftarrow t
11: \hspace{3em} \hspace{1em} \text{break}
12: \hspace{2em} end if
13: \hspace{1em} end for
14: \hspace{1em} t_{\text{step}} \leftarrow t_{\text{step}} / \text{STEP\_FACTOR} \quad \triangleright \text{STEP\_FACTOR} = 10
15: \hspace{1em} until \( t_{\text{step}} \geq t_{\text{res}} \)
16: \hspace{1em} return \( t_j \)
17: end function
```

**Algorithm 2** Spike train firing times of Neuron \( N_j \)

```plaintext
1: function \textsc{CalcTfireMulti}(N_j, \{t_{j}^{(f)}\}, t_{\text{start}}, t_{\text{stop}}, t_{\text{step}}) 
2: \hspace{1em} if \( t_{\text{step}} < t_{\text{res}} \) then
3: \hspace{2em} \{t_{j}^{(f)}\} \leftarrow \{t_{j}^{(f)}\} \cup \{t_{\text{start}}\}
4: \hspace{1em} else
5: \hspace{2em} for \( t \leftarrow t_{\text{start}} : t_{\text{step}} : t_{\text{stop}} \) do
6: \hspace{3em} calculate \( u_j(t) \)
7: \hspace{3em} if \( u_j(t) \geq \vartheta \) then
8: \hspace{4em} \{t_{j}^{(f)}\} \leftarrow \textsc{CalcTfireMulti}(N_j, \{t_{j}^{(f)}\}, t - t_{\text{step}}, t, t_{\text{step}} / \text{STEP\_FACTOR})
9: \hspace{4em} end if
10: \hspace{3em} end if
11: \hspace{2em} end for
12: \hspace{1em} end if
13: \hspace{1em} return \{t_{j}^{(f)}\}
14: end function
```

This step-wise refinement method significantly speeds up forward simulation of SNN (appx. \( 50 \times \)), as it evaluates spike times in logarithmic order compared to linear order of plain brute force method.
3.3 SpikeProp

The SpikeProp algorithm for SNNs is the counterpart of the backpropagation algorithm for MLPs. It follows the principle of error sharing in hidden layer through gradient evaluation. As we mentioned before, the non-linear dynamic behavior of spiking neuron makes it difficult to evaluate derivative of spike event with respect to continuous time values such as membrane potential. Bohte et al. circumvent this issue in SpikeProp [2] by using linearization approximation of membrane potential in the neighborhood of neuron firing time. As it turns out, this so called approximation is mathematically correct which we will see shortly.

**Assumptions:**

1. Each neuron in the network fires exactly once during the period under consideration. In other words, the neurons have long enough refractory period so that they are able to spike only once during the simulation interval.

2. The spiking neuron is modeled using Spike Response Model (SRM) (cf. Chapter 2.2.4)

The assumption that a neuron fires exactly once allows us to greatly simplify the SRM model allowing us to get rid of refractory response and multiple input spike characterization. This results in what is known as SRM\(_0\) model. We will reiterate the SRM\(_0\) model again next.

\[
\begin{align*}
\dot{u}_j(t) &= w^T_{ji}y_{ji}(t) \\
t_j & : u_j(t_j) = \vartheta, \quad u_j'(t_j) > 0
\end{align*}
\]

Here,

\[
y_{ji}^{(k)}(t) = \varepsilon(t - t_i - \Delta_{ji}^{(k)})
\]

is the delayed spike response function

\[
\begin{align*}
\mathbf{y}_{hi}(t_I) &= [\cdots, y_{hi}^{(k)}(t_i), \cdots]^T \in \mathbb{R}^{[I]\times[K]} \\
\mathbf{y}_{oh}(t_H) &= [\cdots, y_{oh}^{(k)}(t_h), \cdots]^T \in \mathbb{R}^{[H]\times[K]}
\end{align*}
\]

are spike response vector for a neuron in hidden layer and output layer respectively.
and

\( Y_{HI}(t_I) = [\cdots, y_{hI}(t_I), \cdots]^T \in \mathbb{R}^{|H|\times|I|} \tag{3.15} \)
\( Y_{OH}(t_H) = [\cdots, y_{oH}(t_H), \cdots]^T \in \mathbb{R}^{|O|\times|HI|} \tag{3.16} \)

are spike response matrices of hidden layer and output layer respectively.

From (3.11), the membrane potential vectors are

\( u_H = [\cdots, w^T_{hI}y_{hI}(t_I), \cdots]^T = \text{diag}(W_{HI}Y_{HI}(t_I)^T) \in \mathbb{R}^{|H|} \tag{3.17} \)
\( u_O = [\cdots, w^T_{oH}y_{oH}(t_H), \cdots]^T = \text{diag}(W_{OH}Y_{OH}(t_H)^T) \in \mathbb{R}^{|O|} \tag{3.18} \)

Even though, the mapping from membrane potential to spike time is an event mapping as defined in (3.11), for backpropagation, we need to transfer the error in spike time into error in membrane potential. For completeness, denote the functional that maps membrane potential in subthreshold regime into spike time as

\( f(u_j) = t_j \tag{3.19} \)

and its corresponding vector function as

\( f(u) = [f(u_1), f(u_2), \cdots, f(u_n)]^T : \mathbb{R}^n \to \mathbb{R}^n \tag{3.20} \)

Then, one can write the operation in hidden layer framework as

\( t_H = f_d(W_{HI}Y_{HI}(t_I)^T) = f_d(W_{HI}Y_{HI}(t_I)^T) \tag{3.21} \)

where

\( f_d(\cdot) = f(\text{diag}(\cdot)) : \mathbb{R}^{n \times n} \to \mathbb{R}^n \tag{3.22} \)

Then, the input-output mapping of a single spiking SNN with one hidden layer can be written as

\( t_O = f_d(W_{OH}Y_{OH}(f_d(W_{HI}Y_{HI}(t_I)^T))^T) \tag{3.23} \)

Now, for SpikeProp rule, we proceed as follows. Since it is necessary to have \( u_j(t_f) = \vartheta \) during spike event, taking differential with respect to weight on both
sides, we get

\[
\frac{du_j(t_j)}{dw_{ji}^{(k)}} = 0 \\
\implies \frac{\partial u_j(t_j)}{\partial w_{ji}^{(k)}} + \frac{\partial u_j(t_j)}{\partial t_j} \frac{\partial t_j}{\partial w_{ji}^{(k)}} = 0 \\
\implies \frac{\partial u_j(t_j)}{\partial w_{ji}^{(k)}} + u_j'(t_j) \frac{\partial t_j}{\partial u_j(t_j)} \frac{\partial u_j(t_j)}{\partial w_{ji}^{(k)}} = 0 \\
\implies \frac{\partial t_j}{\partial u_j(t_j)} = \frac{\partial f(u_j)}{\partial u_j(t_j)} = -\frac{1}{u_j'(t_j)}
\] (3.24)

This is the main contribution by Bohte et al. in [2], although the result is derived using linearization approximation. With this SpikeProp rule, the remaining backpropagation is straightforward.

### 3.3.1 Error Backpropagation

In this section, we will derive the error backpropagation for SpikeProp. The formulation presented here is different from the original derivation in [2]. We use linear algebraic representation to derive the result which will facilitate in the convergence and error analysis in upcoming chapters. Nevertheless, the result is same as in [2].

Consider the target firing time in training sample, \( \hat{t}_O \) Then the system error can be written as

\[
e_O = \hat{t}_O - t_O \] (3.25)

The learning cost function is defined as

\[
E = \frac{1}{2} e_O^T e_O
\] (3.26)

and the gradient descent weight update is given by

\[
\Delta W_{ff} = -\eta \frac{\partial E}{\partial W_{ff}}, \ W_{ff} = W_{OH} \text{ or } W_{HI}
\] (3.27)

where \( \eta \) is learning rate. The learning rate \( \eta \) can be either positive scalar or positive-definite matrix [97]. A positive number is used as learning rate in SpikeProp. In Chapter 5, Chapter 6 and Chapter 7, we will use a positive-definite
matrix of the form $\eta = \text{diag}(\eta_1, \cdots, \eta_J)^T$, $\dot{J} = \mathcal{O}$ or $\mathcal{H}$ so that each spiking neuron has its individual learning rate.

Using § A.1, the gradient for output layer weights is as follows.

$$\frac{\partial E}{\partial W_{OH}} = \text{diag} \left( \frac{\partial t_O}{\partial u_O} \frac{\partial E}{\partial t_O} \right) Y_{OH}$$

$$= - \text{diag} \left( \frac{\partial f(u_O)}{\partial u_O} e_O \right) Y_{OH}$$

$$= - \text{diag} \left( \begin{bmatrix} -1/u'_1(t_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & -1/u'_|\mathcal{O}|(t_{|\mathcal{O}|}) \end{bmatrix} e_O \right) Y_{OH}$$

$$= \text{diag} \left( \begin{bmatrix} e_1(u'_1(t_1)) \\ \vdots \\ e_{|\mathcal{O}|}(u'_{|\mathcal{O}|}(t_{|\mathcal{O}|})) \end{bmatrix}^T \right) Y_{OH}$$

$$\frac{\partial E}{\partial W_{OH}} = \text{diag}(\delta_O) Y_{OH} \quad (3.28)$$

The off-diagonal elements of $\frac{\partial f(u_O)}{\partial u_O}$ are 0 because firing time of a spiking neuron only depends upon its membrane potential only. For diagonal elements we have used (3.24). We have also defined intermediate variable

$$\delta_O \triangleq \frac{\partial t_O}{\partial u_O} \frac{\partial E}{\partial t_O} \quad \delta_o \triangleq \frac{e_o}{u'_o(t_o)} \quad (3.29)$$

for the sake of brevity.

Similarly for hidden layer weights, we can obtain

$$\frac{\partial E}{\partial W_{HI}} = \text{diag}(\delta_H) Y_{HI} \quad (3.30)$$

where

$$\delta_H \triangleq \frac{\partial t_H}{\partial u_H} \frac{\partial E}{\partial t_H} = \frac{\partial t_H}{\partial u_H} \frac{\partial u_O}{\partial t_H} \delta_O$$

$$= \text{diag} \left( \begin{bmatrix} -1/u'_1(t_1) \\ \vdots \\ -1/u'_{|\mathcal{H}|}(t_{|\mathcal{H}|}) \end{bmatrix}^T \right) \left[ - \sum_k w^{(k)}_{oh} \frac{\partial y^{(k)}_{oh}}{\partial u_o} \right] \delta_O \quad (3.31a)$$

$$\delta_h \triangleq \frac{\sum_o \delta_o \sum_k w^{(k)}_{oh} \frac{\partial y^{(k)}_{oh}}{\partial u_o}}{u'_h(t_h)} = -\frac{1}{u'_h(t_h)} \frac{\partial u_O}{\partial t_h} \delta_O = \frac{1}{u'_h(t_h)} \frac{\partial t_O}{\partial t_h} e_O \quad (3.31b)$$
Therefore, the SpikeProp weight update rule at iteration \( p \) is

\[
W_{jj}(p + 1) = W_{jj}(p) - \eta \text{diag}(\delta_j) Y_{jj}
\]  

(3.32)

and the SpikeProp weight update rule for a neuron \( N_j \) is

\[
w_{ji}(p + 1) = w_{ji}(p) - \eta \delta_j y_{ji}.
\]  

(3.33)

The SpikeProp learning algorithm is presented in Algorithm 3 for batch update.

**Algorithm 3 SpikeProp Learning**

1: set up network connections
2: initialize weights
3: repeat
4: \( MSE \leftarrow 0 \)
5: for all training sample do \( \triangleright \) assume \( m \) samples
6: \( \Delta w_{ji}^{(k)} \leftarrow 0 \quad \forall j, i, k \)
7: for all \( N_i \in I \) do
8: \( t_i \leftarrow \text{input}_i \)
9: end for
10: for all \( N_j \in H, O \) do \( \triangleright \) Forward Simulation
11: \( t_j \leftarrow \text{CALCTFIRE}(N_j) \) \( \triangleright \) algorithm 1
12: end for
13: for all \( N_j \in O, H \) do
14: calculate \( \delta_j \) \( \triangleright \) equations (3.29) & (3.31)
15: for \( k \leftarrow 1 : |K| \) do
16: \( \Delta w_{ji}^{(k)} \leftarrow \Delta w_{ji}^{(k)} - \eta \delta_j y_{ji}^{(k)}(t_j) \) \( \triangleright \) \( i \in \Gamma_j \)
17: end for
18: if \( N_j \in O \) then
19: \( MSE \leftarrow MSE + E/m \) \( \triangleright \) equation (3.26)
20: end if
21: end for
22: end for
23: \( w_{ji}^{(k)} \leftarrow w_{ji}^{(k)} + \frac{1}{m} \Delta w_{ji}^{(k)} \)
24: until \( MSE < \text{TOL} \) OR \( \text{MAX\_EPOCH} \) is exceeded

### 3.3.2 Implementation Caveats

The weight update rule of SpikeProp backpropagation is very similar to the MLP backpropagation weight update rule. However, there are a couple of practical details that need to be considered during implementation of SpikeProp.
Weight initialization

SpikeProp training is influenced by initial weights of the network. Simon C. Moore [98] conducted an in-depth study on how different weight initialization affect the the result of SpikeProp learning. Furthermore, Bohte et al. [2] suggest using explicitly positive and negative weights for excitatory and inhibitory synapse. Although, a mix of positive and negative weights is needed to be learnt to arrive at solution, we find that it is not necessary to have explicit demarcation as excitatory and inhibitory synapse. We allow the weights to change sign and determine by itself whether a particular synapse needs to be excitatory or inhibitory and it works well. Same observation has been made in [96,99] as well.

For initial weight initialization, we use the method suggested by McKennoch et al. [96]. This method is intuitive and works better in practice than simple random initialization. It is described below. Define

$$w_{\text{max}} = \frac{\vartheta}{|K||\Gamma_j| \varepsilon(t_{\text{min}})} \quad \& \quad w_{\text{min}} = \frac{\vartheta}{|K||\Gamma_j| \varepsilon(t_{\text{max}})}.$$  

(3.34)

Here, $|\Gamma_j|$ is the number of neurons in preceding layer, $t_{\text{max}}$ and $t_{\text{min}}$ are the maximum and minimum delay in firing that is expected once the neuron receives input spike. The weights are drawn form the normal distribution $W_{ji} \sim \mathcal{N}(\mu, \sigma^2)$ such that $\mu = \frac{2}{3}(w_{\text{max}} - w_{\text{min}}) + w_{\text{min}}$ and $\sigma = \frac{1}{6}(w_{\text{max}} - w_{\text{min}})$.

Dead Neuron Problem

Frequently, the weight associated to a neuron gets updated to such a small value that its membrane potential is unable to reach threshold value. This scenario is called dead neuron or silent neuron situation. Since we need the neuron to fire to proceed backpropagation using SpikeProp, SpikeProp and methods based on SpikeProp gradient are stuck in this scenario as no weight update can be made pertaining to this neuron. Fortunately, there are heuristic solutions that work well in practice. One such a solution is to lower the threshold of the specific neuron [96,98]. Alternatively, we can selectively increase the weights associated with the dead neuron [64]. This is called boosting. We use boosting whenever a dead neuron is encountered in our simulation.
Hair Trigger Situation

Another problem that is common in SpikeProp learning is when \( u'(t_j) \approx 0 \) i.e. the neuron membrane potential barely reaches the threshold. This scenario is called hair trigger situation. Since the term \( u'(t_j) \) is in the denominator of (3.29) and (3.31), from (3.33) the change in weight is very large. This is a major problem as it disturbs the learning trajectory. A practical solution to this problem is to set a lower bound to \( u'(t_j) \) at a reasonably small value, say 0.01. This is similar to Laplacian smoothing of statistical estimator. We will see in Chapter 5, Chapter 6, Chapter 7 and Chapter 8, the adaptive learning rules curb the hair trigger problem.

3.4 Extension to SpikeProp

With the problem of mapping change in spike times into change in membrane potential tackled by Bohte et al., different extensions based on gradient evaluation technique of SpikeProp have been made. In [94], Schrauwen et al. extended SpikeProp for leaning delay parameters (cf. Section 3.4.1). In [96], McKennoch et al. use the sign of gradient to port QuickProp and RProp algorithms for use in SNN learning (cf. Section 3.4.2 and Section 3.4.3). Booji et al. [64], Ghosh et al. [17] and Xu et al. [95] extend the idea of SpikeProp for multiple firing neurons (cf. Section 3.4.4). We will discuss these extensions next.

3.4.1 Extended SpikeProp

Apart from synaptic weights, there are other free parameters in a SRM\(_0\) network like synaptic delay, neuron threshold and spike response time constant. In [94], Schrauwen et al. worked on learning all these free parameters. It was observed that delay learning is the most effective among others which is described below.

For delay adaptation, consider

\[
\frac{\partial E}{\partial d_{ji}^{(k)}} = \frac{\partial E}{\partial t_j} \frac{\partial t_j}{\partial d_{ji}^{(k)}} \frac{\partial d_{ji}^{(k)}}{\partial t_j} = \delta_j \left( -w_{ji}^{(k)} \varepsilon'(t_j - t_i - d_{ji}^{(k)}) \right)
\]

\[
\Rightarrow \frac{\partial E}{\partial d_{ji}^{(k)}} = -\delta_j w_{ji}^{(k)} \frac{\partial y_{ji}^{(k)}(t_j)}{\partial t}
\]

(3.35)
Define
\[
\dot{\mathbf{j}} = \left[ \cdots, \frac{\partial y_k(t_j)}{\partial t}, \cdots \right] \in \mathbb{R}^{|J||K|} \quad (3.36)
\]

Then the delay update rule according to gradient descent rule is
\[
d_j(p + 1) = d_j(p) + \alpha \delta \text{diag}(w_j) \dot{y}_j \quad (3.37)
\]

where \(\alpha\) is the learning rate for delay adaptation.

Similar results exist for adaptation of \(\tau_s\) and \(\vartheta\) as well. However, experimental results in [94] show that these parameters do not change their values significantly from their initial point. The SpikeProp with delay learning referred to as SpikePropDel in the rest of this thesis.

### 3.4.2 QuickProp

QuickProp for SNN follows the same process as the original QuickProp for MLP in [100] with the gradient term evaluated using SpikeProp. The weight adaptation rule at \(p\) iteration is given by
\[
\Delta w^{(k)}_{ji}(p) = \frac{\partial E}{\partial w^{(k)}_{ji}(p)} \Delta w^{(k)}_{ji}(p - 1) = \beta \Delta w^{(k)}_{ji}(p - 1) \quad (3.38)
\]

where \(\frac{\partial E}{\partial w^{(k)}_{ji}(p)}\) refers to the error gradient at \(p\) iteration and \(\Delta w^{(k)}_{ji}(p - 1)\) is the weight adaptation in previous step (iteration \(p - 1\)). It is based upon two basic assumptions [100]:

1. The error vs. weight curve for each weight can be approximated by a parabola whose arms open upward.
2. The change in the slope of the error curve, as seen by each weight, is not affected by all the other weights that are changing at the same time.

When the learning step reaches minimum point, the gradient values become very small and nearly equal. From (3.38), we can see that this will cause large change in weight and is a problematic situation. One must use an additional limiting mechanism to settle this problem [100]. Further, in the sensitivity analysis of QuickProp in [96] shows that the change in firing time cause by QuickProp weight
update rule is unbounded and dependent on current weight. This is not a general behaviour of QuickProp and is unique to SNNs [96]. The highly non-linear error surface of SNNs test the limits of assumptions made by QuickProp and aggravates its unstable nature. Nevertheless it is faster than SpikeProp if it does not diverge.

### 3.4.3 RProp

RProp for SNN is also same as the original RProp for MLP in [101]. The weight is updated according to the rules in (3.39) and (3.40).

\[
\Delta^{(k)}_{ji}(p) = \begin{cases} 
\eta^+ \Delta^{(k)}_{ji}(p-1), & \text{if } \frac{\partial E}{\partial w^{(k)}_{ji}}(p-1) \frac{\partial E}{\partial w^{(k)}_{ji}}(p) > 0 \\
\eta^- \Delta^{(k)}_{ji}(p-1), & \text{if } \frac{\partial E}{\partial w^{(k)}_{ji}}(p-1) \frac{\partial E}{\partial w^{(k)}_{ji}}(p) < 0 \\
\Delta^{(k)}_{ji}(p-1), & \text{otherwise}
\end{cases}
\] (3.39)

\[
\Delta w^{(k)}_{ji}(p) = \begin{cases} 
-\Delta^{(k)}_{ji}(p), & \text{if } \frac{\partial E}{\partial w^{(k)}_{ji}}(p) > 0 \\
\Delta^{(k)}_{ji}(p), & \text{if } \frac{\partial E}{\partial w^{(k)}_{ji}}(p) < 0 \\
0, & \text{otherwise}
\end{cases}
\] (3.40)

Here, \(0 < \eta^- < 1 < \eta^+\) are the parameters that change the weight adaptation amount. In RProp, \(\eta^+ \approx 1.2\) and \(\eta^- \approx 0.5\) usually works well [101]. This is evident in the results of [96] for RProp.

RProp method is ill suited for sequential update because the change in gradient from learning sample to sample results in oscillating behaviour in weight update rather than learning. Such behaviour is not observed in batch update. Nevertheless, such oscillating behaviour is common when the learning error becomes small (cf. Chapter 5.3). In terms of speed, RProp is much faster than standard SpikeProp and faster than QuickProp as well.

### 3.4.4 SpikeProp for Multiple Spiking Neuron

Originally, SpikeProp was developed for multilayer SNNs with assumption that neurons fire exactly once. In spite of that, there have been couple of attempts to extend it for multiple spiking neurons as well.
Booji et al. [64] considered multiple spiking neurons in hidden layer and a recursive learning rule was developed. Ghosh et al. [17] simplify this idea using short term memory neuron model (cf. Chapter 2.2.4) and derived learning rule. This learning rule is called Multi-SpikeProp. This rule is more tractable than the one in [64] because it gets rid of refractory response of older firing times, which are usually zero anyway. The learning rule is described below.

Network error is defined as

$$E = \frac{1}{2} \sum_o \left( \hat{t}_o - t_o \right)^2$$ \hspace{1cm} (3.41)

Weight update rule is

$$\delta w_{ji}^{(k)} = -\eta \frac{\partial E}{\partial w_{ji}^{(k)}}$$ \hspace{1cm} (3.42)

where for output layer neuron,

$$\frac{\partial E}{\partial w_{oh}^{(k)}} = (t_o - \hat{t}_o) \frac{\partial t_o}{\partial w_{oh}^{(k)}}$$ \hspace{1cm} (3.43)

for hidden layer neuron,

$$\frac{\partial E}{\partial w_{hi}^{(k)}} = \sum_o (t_o - \hat{t}_o) \frac{1}{u'_o(t_o)} \sum_h \sum_k w_{oh}^{(k)} \epsilon'(t_o - t_h^{(f)} - d_{oh}^{(k)}) \frac{\partial t_h^{(f)}}{\partial w_{hi}^{(k)}}$$ \hspace{1cm} (3.44)

and the term \( \frac{\partial t_j^{(f)}}{\partial w_{ji}^{(k)}} \) is evaluated recursively as follows

$$\frac{\partial t_j^{(f)}}{\partial w_{ji}^{(k)}} = -\frac{1}{u'_j(t_j)} \left[ y_{ji}^{(k)} (t_j^{(f)}) - \nu'(t_j^{(f)} - t_j^{(f-1)}) \frac{\partial t_j^{(f-1)}}{\partial w_{ji}^{(k)}} \right]$$ \hspace{1cm} (3.45)

Xu et al. [95] further developed on this idea to postulate SpikeProp-Multi rule for learning a fixed number of spike times with the assumption that the number of desired spike times is always equal to the number of actual firing times in the network. This is a rather rigid assumption because it is not always possible to satisfy the condition, more so when the learning is in its initial stage.
3.5 Remote Supervision Method

ReSuMe is short for Remote Supervision Method. It is an extension to supervised Hebbian learning. It overcomes the inability of supervised Hebbian learning to weaken synaptic weights and instability problem [74]. Further, it can be used to train a single spiking neuron to learn a spike train pattern [15,78,79]. Its major advantage is that it is independent of the spiking neuron model.

Denote the input spike train, output spike train and the desired spike train as $S_i(t)$, $S_o(t)$ and $S_d(t)$ respectively. The neuron with desired spike train is called the teacher neuron and is used to adapt the weight $w_{oi}$ between the input neuron, $N_i$, and the output neuron, $N_o$. This mechanism is illustrated in figure 3.2.

\[ \frac{d}{dt} w_{oi}(t) = S_d(t) \left[ a_d + \int_0^\infty W_d(s) S_i(t-s) \, ds \right] + S_o(t) \left[ a_o + \int_0^\infty W_o(s) S_i(t-s) \, ds \right] \quad (3.46) \]

For excitatory synapses, the constants $a_d > 0$ & $a_o < 0$ and for the case of inhibitory synapses, $a_d < 0$ & $a_o > 0$ [15,78,79]. The learning windows in (3.46) have been proposed as

\[ W_d(s) = A_d e^{-\frac{s}{\tau_d}} \Theta(s) \quad (3.47) \]
\[ W_o(s) = -A_o e^{-\frac{s}{\tau_o}} \Theta(s) \quad (3.48) \]

where $A_d, A_o > 0$ for excitatory synapses and $A_d, A_o < 0$ for inhibitory synapses. The learning window time constants $\tau_d$ and $\tau_o$ are always positive. ReSuMe learning with learning window is illustrated in figure 3.3.
Among the methods that teach spike train pattern to a spiking neuron, ReSuMe is very popular, partly because of its model independence and partly because of its effectiveness in learning.

### 3.6 Multi-ReSuMe

Sporea et al. [38] have recently extended ReSuMe learning for multilayer SNN, called Multi-ReSuMe. The authors used firing rate backpropagation to transfer the error from output layer to hidden layer neurons and then use ReSuMe like transformation into spike signals. The learning rule for Multi-ReSuMe is described below.

Similar to ReSuMe learning, the learning window is defined as

\[
W(s) \begin{cases} 
    a^{\text{pre}}(s) = -A e^{\frac{s}{\tau}} \left( 1 - \Theta(s) \right) \\
    a^{\text{post}}(s) = A e^{-\frac{s}{\tau}} \Theta(s)
\end{cases} \tag{3.49}
\]
Now, the learning rule for output layer neuron is

\[
\frac{dw_{oh}(t)}{dt} = \frac{1}{|O|} S_h(t) \left[ \int_0^\infty a^{\text{pre}}(s) \left[ S_o^d(t - s) - S_o(t - s) \right] \, ds \right] \\
+ \frac{1}{|O|} \left[ S_o^d(t) - S_o(t) \right] \left[ a + \int_0^\infty a^{\text{post}}(s) S_h(t - s) \, ds \right]
\] (3.50)

and the learning rule for hidden layer neuron is

\[
\frac{dw_{hi}(t)}{dt} = \frac{1}{|O||H|} S_i(t) \sum_o \left[ \int_0^\infty a^{\text{pre}}(s) \left[ S_o^d(t - s) - S_o(t - s) \right] \, ds \right] w_{oh} \\
+ \frac{1}{|O||H|} \sum_o \left[ S_o^d(t) - S_o(t) \right] \left[ a + \int_0^\infty a^{\text{post}}(s) S_i(t - s) \, ds \right] w_{oh}
\] (3.51)

The complexity of this method increases in quadratic order when the size of spike train goes on increasing. Apart from that, it assumes rate coding to distribute error in hidden layer which does not accord with the precise time argument for SNN.

### 3.7 Performance Measures

In this section, we will discuss about different measures that will be used to compare the learning performance of two SNN algorithms. We will analyze the performance from different vantage points using various measures to obtain fair overall performance comparison. These measures are specifically chosen because they are independent of the coding implementation, therefore, they evaluate the effectiveness of the algorithm rather than the effectiveness of coding.

#### 3.7.1 Network Error

Network is the measure of how close the achieved output is with respect to the target output. For single spiking neurons, Mean Square Error (MSE) is sufficient measure of network error which is the cost function defined in (3.26). However for spike train learning, MSE is not sufficient. We need to use similarity measures for spike trains to characterize network error. There are different similarity measures to compare spike trains \textit{viz.} Victor Purpura distance, van Rossum distance,
Schriber et al. induced divergence, Hunter Milton similarity measure etc. (for comparison, refer to [102]). We will use popular van Rossum distance which is of similar form as MSE to measure network error. For two spike trains \( S(t) \) and \( \hat{S}(t) \), it is defined as follows [102].

\[
D_R(S(t), \hat{S}(t)) = \frac{1}{\tau_R} \int_t \left[ (S(t) - \hat{S}(t)) * \left( e^{-\frac{t}{\tau_R}} \Theta(t) \right) \right]^2 dt
\]

(3.52)

The distance is zero if and only if the two spikes are identical.

Network error serves as the target of minimum error value of learning for an algorithm. We will refer to it as network error tolerance or simply tolerance value.

### 3.7.2 Convergence Rate

We will use convergence rate as a measure of success during learning instances. It is defined as follows.

**Convergence Rate** — *Convergence Rate of a learning algorithm is defined as the probability of success in learning during a trial. The success in learning is measured in terms of the network error reaching a certain tolerance value during learning.*

For a learning algorithm, a convergence rate close is one is a desirable property. Sometimes, we will refer to convergence rate as conv. in shorthand form to save space in illustrations.

### 3.7.3 Mean Epoch

As a measure of speed of a learning algorithm, we will use the number of epochs to reach a specified network error to measure it. We chose epoch for this purpose because it is not dependent on the implementation of the code and reflects the speed of learning rather than the efficiency in coding. Further, it also allows us make comparison between algorithms that are implemented in batch or sequential mode as in an epoch the network gets to learn from all the available training samples and change its parameters.

As a statistical measure of speed, we will use the average of number of epochs required by a certain algorithm to reach a tolerance network error value. We will
refer to it as mean epoch throughout this thesis. A learning algorithm with smaller mean epoch to reach sufficiently low tolerance value is a desirable property of an algorithm.

3.7.4 Accuracy

Accuracy of learning is the main objective of classification learning. We will measure accuracy on training dataset as well as testing dataset whenever relevant to measure the performance of a learning algorithm. As always higher training accuracy along with higher testing accuracy is desired for an algorithm.

3.7.5 Average Learning Curve

In this thesis, by learning curve we will mean the plot of network error values versus the epochs elapsed. The average of learning curve over different trials depicts the efficacy of an learning algorithm showing how consistent the learning method is for different trials. An average learning curve that quickly settles into a stable and lower network error is desired for an algorithm.
Chapter 4

Non-Linear Stability Theory

It is relatively easy to estimate conic bounds for simple interconnections, where it might be more difficult, say, to find Lyapunov functions.

—George Zames, 1966

Stability is an important and desirable property of a system, be it continuous time or discrete time. Broadly, when we say a system is stable, we mean that the system is well behaved i.e. the internal states of the system or outputs of the system do not diverge to infinity along with time, but rather are bounded, perhaps converge over time. In practical systems, stability of the system is paramount. There are two broad techniques for stability analysis of non-linear system: Lyapunov’s internal state method and Input-Output method.

The convergence and stability issues is crucial aspect of machine learning systems. We will discuss these stability concepts with focus on machine learning uses in this chapter. Before we discuss about aforementioned stability techniques for non-linear systems, we will present some basic definitions.

4.1 Mathematical Preliminaries

Throughout this thesis, for a variable $x$ we will use the notation $x(t)$ to imply it is a continuous time signal and use $x(p)$ to imply it is a discrete time signal. We will not use the more common notation for discrete time signal, $x(k)$ or $x_k$ to avoid
confusion with the delayed synaptic connection which is denoted by \( k \) as well.

### 4.1.1 The \( L_2 \) space and Extended \( L_2 \) space

**Inner Product** — The Inner product of two signals \( x \) and \( y \) is denoted by \( \langle x | y \rangle \). For continuous time signals \( x(t) \) and \( y(t) \), it is defined as

\[
\langle x | y \rangle \triangleq \int_0^\infty x(t) y(t) \, dt
\]

and for discrete time signals \( x(p) \) and \( y(p) \), it is defined as

\[
\langle x | y \rangle \triangleq \sum_{p=1}^\infty x(p) y(p)
\]

**\( L_2 \) norm** — The \( L_2 \) norm of signal \( x \) is denoted by \( \| x \|_2 \) and defined as

\[
\| x \|_2 \triangleq \sqrt{\langle x | x \rangle} = \begin{cases} 
\int_0^\infty x(t)^2 \, dt & \text{for continuous time signal} \\
\sum_{p=1}^\infty x(p)^2 & \text{for discrete time signal}
\end{cases}
\]

By default, a norm \( \| x \| \) is referred to as \( L_2 \) norm.

**\( L_2 \) space** — The \( L_2 \) space is the space of all square integral functions defined by

\[
L_2 \triangleq \{ x : \| x \|_2 < \infty \}
\]

The extended \( L_2 \) space, denoted by \( L_{2e} \) is the space consisting of those elements of \( x \) whose truncation lies in \( L_2 \) \[103\]. The inner product and norm are redefined in extended space as

**Truncated Inner Product** — The truncated inner product of two signals \( x \) and \( y \) is denoted by \( \langle x | y \rangle_T \). For continuous time signals \( x(t) \) and \( y(t) \), it is defined as

\[
\langle x | y \rangle_T \triangleq \int_0^T x(t) y(t) \, dt
\]
and for discrete time signals $x(p)$ and $y(p)$, it is defined as

$$\langle x | y \rangle_T \triangleq \sum_{p=1}^{T} x(p) y(p)$$

$L_2e$ norm — The $L_2e$ norm of signal $x$ is denoted by $\|x\|_{2e}$ and defined as

$$\|x\|_{2e} \triangleq \sqrt{\langle x | x \rangle_T} = \begin{cases} \int_0^T x(t)^2 \, dt & \text{for continuous time signal} \\ \sum_{p=1}^{T} x(p)^2 & \text{for discrete time signal} \end{cases}$$

$L_2e$ space — The $L_2e$ space is the space functions defined by

$$L_2e \triangleq \{ x : \|x\|_{2e} < \infty \}$$

4.1.2 The $L_\infty$ space

$L_\infty$ norm — The $L_\infty$ norm of signal $x$ is denoted by $\|x\|_\infty$ and defined as

$$\|x\|_\infty \triangleq \sup |x|$$

$L_\infty$ space — The $L_\infty$ space is the space of all bounded functions defined by

$$L_\infty \triangleq \{ x : \|x\|_\infty < \infty \}$$

4.1.3 Gain

Gain — [104] The gain of an operator $H : x \rightarrow y$ is denoted by $\gamma(H)$ and defined by

$$\gamma(H) = \sup \frac{\|Hx\|}{\|x\|}$$

where the supremum is taken over all $x$ in $\text{Do}(H)$, all $y$ in $\text{Ra}(H)$ for which $x \neq 0$. 
4.1. Mathematical Preliminaries

Depending upon the norm, gain in corresponding space is defined. The inequality

$$\|Hx\| \leq \gamma(H) \|x\|$$

follows immediately.

4.1.4 Passivity

A passive system is the system that always absorbs energy [105].

**Passivity** — [105] An operator $H : x \rightarrow y$ where $x, y \in L_{2e}$ is passive only if there exists some constant $\beta$ such that

$$\langle x|y \rangle_T = \langle Hx|x \rangle_T \geq \beta$$

A passive operator is also called non-dissipative. When $\beta = 0$, the relation $H$ is also positive (short for positive semidefinite). A positive relation is degenerately conic (cf. Section 4.1.5), with a sector from 0 to $\infty$ [104].

4.1.5 Conic Sector

**Conic Sector** — [103] An operator $H : x \rightarrow y$ where $x, y \in L_{2e}$ is
(a) inside the Cone\((C, R)\) if
\[ \langle y - (C - R)x|y - (C + R)x \rangle_T \leq 0 \quad \forall T \in \mathbb{Z}^+ \]

(b) outside the Cone\((C, R)\) if
\[ \langle y - (C - R)x|y - (C + R)x \rangle_T \geq 0 \quad \forall T \in \mathbb{Z}^+ \]

(c) strictly inside the Cone\((C, R)\) if for some \(\kappa > 0\)
\[ \langle y - (C - R)x|y - (C + R)x \rangle_T \leq -\kappa \|x, y\|_T^2 \quad \forall T \in \mathbb{Z}^+ \]

(d) strictly outside the Cone\((C, R)\) if for some \(\kappa > 0\)
\[ \langle y - (C - R)x|y - (C + R)x \rangle_T \geq -\kappa \|x, y\|_T^2 \quad \forall T \in \mathbb{Z}^+ \]

Here \(\|x, y\|_T^2 = \|x\|_T^2 + \|y\|_T^2\). An illustration of conic sector is depicted in figure 4.1.

4.1.6 Normalized Signal

Normalized signals and operators are indicated by \((\cdot^n)\) in this thesis. For a discrete time signal \(x(p)\), it is defined as
\[ x^n(p) \triangleq \rho(p)^{-1/2} x(p) \]

and for operator \(H : x(p) \rightarrow y(p)\), it is defined as
\[ H^n[\cdot] \triangleq \rho(p)^{-1/2} H[\rho(p)^{1/2} \cdot] \]

where \(\rho(p)\) is called normalization factor [105]. Similarly for a continuous time signal \(x(t)\) and operator \(H : x(t) \rightarrow y(t)\), it is defined as
\[ x(t)^n \triangleq \rho(t)^{-1/2} x(t) \]
\[ H^n[\cdot] \triangleq \rho(t)^{-1/2} H[\rho(t)^{1/2} \cdot] \]

where \(\rho(t)\) is normalization factor.
4.1.7 Exponentially Weighted Signal

The exponentially weighted counterpart of a discrete time signal \( x(p) \) is denoted by \( x(p)^\beta \) and defined as

\[
x(p)^\beta = \beta^p x(p), \quad \beta > 0
\]

Similarly for continuous time signal, it is denoted by \( x(t)^\beta \) and defined as

\[
x(t)^\beta = e^{\beta t} x(t)
\]

4.2 Lyapunov Stability

Lyapunov stability is extremely popular and very powerful method for stability and convergence analysis. It describes stability of a system based in its internal state parameters. Simply stating, if the internal state parameters of the system move towards equilibrium point or remain close to the equilibrium point, then the system is Lyapunov stable. We will now formally state Lyapunov stability criterion.

**Lemma 4.1.** [106] (Second Method of Lyapunov: For Continuous-Time Autonomous System) Consider an autonomous nonlinear dynamical system

\[
\dot{x}(t) = f(x), \quad x(t_0) = x_0 \in \mathbb{R}^n
\]

where \( x(t) \in \mathcal{D} \subseteq \mathbb{R}^n \) is the system state vector, \( \mathcal{D} \) is an open set containing the origin, and \( f : \mathcal{D} \to \mathbb{R}^n \) is a continuous function on \( \mathcal{D} \). Let \( x = 0 \) be an equilibrium point of the system. The autonomous system is stable about its zero equilibrium, if there exists a scalar-valued function \( V(x) \), defined on \( \mathcal{D} \), such that

(a) \( V(0) = 0 \).

(b) \( V(x) > 0 \) for all \( x \neq 0 \) in \( \mathcal{D} \).

(c) \( \dot{V}(x) \leq 0 \) for all \( x \neq 0 \) in \( \mathcal{D} \).

Furthermore, the system is asymptotically stable about its zero equilibrium when \( \dot{V}(x) < 0 \).

**Lemma 4.2.** [106] (Second Method of Lyapunov: For Discrete-Time Autonomous System)
Consider a discrete autonomous nonlinear dynamical system

\[ x(p + 1) = f(x(p)), \quad x(0) = x_0 \in \mathbb{R}^n \]

where \( x(p) \in \mathcal{D} \subseteq \mathbb{R}^n \) is the system state vector, \( \mathcal{D} \) is an open set containing the origin, and \( f : \mathcal{D} \to \mathbb{R}^n \) is continuously differentiable in a neighbourhood of the origin. Let \( \bar{x} = 0 \) be an equilibrium point for the system. Then the system is globally (over the entire domain \( \mathcal{D} \)) and asymptotically stable about this zero equilibrium if there exists a scalar-valued function \( V(x(p)) \), defined on \( \mathcal{D} \) and continuous in \( x(p) \), such that

(a) \( V(0) = 0 \).

(b) \( V(x(p)) > 0 \) for all \( x(p) \neq 0 \) in \( \mathcal{D} \).

(c) \( \Delta V(x(p)) = V(x(p)) - V(x(p - 1)) \leq 0 \) for all \( x(p) \neq 0 \) in \( \mathcal{D} \).

(d) \( V(x) \to \infty \) as \( \|x\| \to \infty \).

In this thesis, we will apply Lyapunov stability to derive conditions that guarantee the convergence and stability of learning algorithms for SNNs.

### 4.3 Input-Output Stability

When assessing stability via Lyapunov’s direct method, it is assumed that there are no external disturbances or noise in the system. However, in practical systems, we cannot avoid disturbances. In learning systems, the disturbance can be in form of external disturbances due to error in learning samples or internal disturbance due to incorrect internal modelling parameters as well. The functional analysis based input-output theory makes minimal assumption about the process under consideration and lends well to investigate robust stability of parameter adaptation based learning algorithms.

The Conic Sector Stability theorem is generalization of the concept of Bounded Input Bounded Output (BIBO) stability for more general processes. During early 1970’s, George Zames [104] outlined three ways of accessing closed loop stability in terms of input-output relationship: Small Gain Theorem, Passivity Theorem and Conic Sector Stability Theorem. The first two are actually special cases of Conic Sector Stability Theorem. These Conic Sector Stability results were later extended by various individuals, most notably by Safonov [107]. We will now
discuss Conic Sector Stability for nonlinear discrete systems and the extension of the stability result in $L_\infty$ space for relaxation of the constraints. Before we delve into the details, we will formalize the feedback system as follows.

\[
\begin{align*}
\epsilon(p) &= \epsilon(p) - r(p) = -H_2 \dot{e}(p) + \epsilon(p) \\
\dot{e}(p) &= H_1 e(p) \\
r(p) &= H_2 \dot{e}(p)
\end{align*}
\]

\[\text{Figure 4.2: Closed-loop feedback system.}\]

Consider a feedback system described by following equations

\[
\begin{align*}
\epsilon(p) &= \epsilon(p) - r(p) = -H_2 \dot{e}(p) + \epsilon(p) \\
\dot{e}(p) &= H_1 e(p) \\
r(p) &= H_2 \dot{e}(p)
\end{align*}
\]

with $H_1, H_2 : L_{2e} \rightarrow L_{2e}, r(p), \dot{e}(p), e(p) \in L_{2e}$ and $\epsilon(p) \in L_{2e}$.

The feedback system is shown in figure 4.2. This system will be the subject of conic sector stability in the upcoming sections and represent the update system of parameter adaptation algorithms for SNN learning in this thesis.

\subsection*{4.3.1 Conic Sector Stability}

The basic formulation of conic sector stability is presented in [104] (Theorem 2a). We will present a specific extension to conic sector stability by Michael G. Safanov [107] for discrete time adaptive control systems.

\textbf{Theorem 4.3.} Consider the feedback system of (4.1). If

(a) $H_1 : e(p) \rightarrow \dot{e}(p)$ satisfies

\[
\sum_{p=0}^{N} \left[ \frac{\sigma}{2} \epsilon(p)^2 + e(p) \dot{e}(p) \right] \geq -\gamma
\]

\[\text{(4.2a)}\]
(b) $H_2 : e(p) \to r(p)$ satisfies

$$
\sum_{p=0}^{N} \left[ \frac{\sigma}{2} r(p)^2 - r(p) \dot{e}(p) \right] \leq -\omega \| r(p), \dot{e}(p) \|_N^2
$$

(4.2b)

for some $\sigma, \gamma, \omega > 0$, then the closed loop signals $e(p), \dot{e}(p) \in L_2$.

Proof. The proof follows similar approach in [103]. From (4.2a) and (4.1a), we get

$$
\sum_{p=0}^{N} \left[ \frac{\sigma}{2} (\epsilon(p) - r(p))^2 + (\epsilon(p) - r(p)) \dot{e}(p) \right] \geq -\gamma
$$

$$
\implies \sum_{p=0}^{N} \left[ \frac{\sigma}{2} \epsilon(p)^2 + \frac{\sigma}{2} r(p)^2 - \sigma \epsilon(p) r(p) + \epsilon(p) \dot{e}(p) - r(p) \dot{e}(p) \right] \geq -\gamma
$$

(4.3)

Combining (4.2b) and (4.3)

$$
-\omega \| r(p), \dot{e}(p) \|_N^2 + \sum_{p=0}^{N} \left[ \frac{\sigma}{2} \epsilon(p)^2 - \sigma \epsilon(p) r(p) + \epsilon(p) \dot{e}(p) \right] \geq -\gamma
$$

Using the Schwartz inequality

$$
-\omega \| r(p), \dot{e}(p) \|_N^2 + \frac{\sigma}{2} \| \epsilon(p) \|_N^2
$$

$$
+ \sigma \| \epsilon(p) \|_N \| r(p) \|_N \| \epsilon(p) \|_N \geq -\gamma
$$

$$
\implies \omega \| r(p), \dot{e}(p) \|_N^2 - \sigma \| \epsilon(p) \|_N \| r(p) \|_N
$$

$$
- \| \epsilon(p) \|_N \| \dot{e}(p) \|_N \leq \gamma + \frac{\sigma}{2} \| \epsilon(p) \|_N^2
$$

(4.5)

As $N \to \infty$, $\| r(p), \dot{e}(p) \|_N^2 \to \infty$. Therefore from (4.5), we must have $\omega < 0$ which contradicts with the original assumption. Therefore $\| r(p), \dot{e}(p) \|_N^2 \to \infty$ is bounded, which implies $r(p), \dot{e}(p) \in L_2$. From (4.1a), since $r(p), \dot{e}(p) \in L_2$, we also have $e(p) \in L_2$. $\square$

Corollary 4.3.1. Consider the feedback system of (4.1). The equivalent conditions in Theorem 4.3 for the closed loop signals $e(p), \dot{e}(p) \in L_2$ are

(a) $H_1 + \sigma/2$ is passive.

(b) $H_2$ is strictly inside $\text{Cone}(\sigma^{-1}, \sigma^{-1})$. 

Nanyang Technological University Singapore
Proof. From (4.2a), we get
\[
\sum_{p=0}^{N} \left( \frac{\sigma}{2} e(p) + H_1[e(p)] \right) e(p) \geq -\gamma
\]
\[
\iff \left\langle \left( H_1 + \frac{\sigma}{2} \right) e(p) \right| e(p) \right \rangle \geq -\gamma \tag{4.6}
\]
and the result (a) follows.

For (b), consider (4.2b)
\[
\sum_{p=0}^{N} r(p) \left( \frac{\sigma}{2} r(p) - \tilde{e}(p) \right) \leq -\omega \, \|r(p), \tilde{e}(p)\|_N^2
\]
\[
\iff \sum_{p=0}^{N} r(p) \left( r(p) - \frac{2}{\sigma} \tilde{e}(p) \right) \leq -\frac{2\omega}{\sigma} \|r(p), \tilde{e}(p)\|_N^2
\]
\[
\iff \left\langle r(p) \left| r(p) - \frac{2}{\sigma} \tilde{e}(p) \right. \right \rangle \leq -\frac{2\omega}{\sigma} \|r(p), \tilde{e}(p)\|_N^2 \tag{4.7}
\]

From definition of conic sector (cf. Section 4.1.5), $H_2$ is strictly inside a CONE with
\[
\begin{align*}
C - R &= 0 \\
\left\{ C + R = \frac{2}{\sigma} \right\} \implies C &= R = \frac{1}{\sigma}
\end{align*}
\]
This completes the proof. \qed

Theorem 4.3 is a specific version of conic sector stability extension by Safanov [107] which suits well to the stability analysis of SNN learning algorithms in this thesis. A more general version of conic sector stability extension for discrete time adaptive control systems is as follows.

Theorem 4.4. Consider the feedback system of (4.1). If

(a) $H_1 : e(p) \to \tilde{e}(p)$ satisfies

\[
\sum_{p=0}^{N} \left[ \tilde{e}(p)^2 + \alpha e(p) \tilde{e}(p) + \beta e(p)^2 \right] \geq -\gamma \tag{4.8a}
\]
(b) $H_2 : \hat{e}(p) \rightarrow r(p)$ satisfies

$$\sum_{p=0}^{N} \left[ \beta r(p)^2 - \alpha r(p) \hat{e}(p) + \hat{e}(p)^2 \right] \leq -\omega \| r(p), \hat{e}(p) \|_N^2$$ (4.8b)

for some $\alpha, \beta, \gamma, \omega \in \mathbb{R}$ and $\gamma, \omega > 0$, then the closed loop signals $e(p), \hat{e}(p) \in L_2$.

Proof. The proof follows the same approach taken for Theorem 4.3. $\square$

The $L_2$ stability result of Theorem 4.3 can be extended a system of vector feedback system as well, formulated as follows. Consider a vector feedback system described by following equations

$$e(p) = \epsilon(p) - r(p) = -H_2 \hat{e}(p) + e(p)$$ (4.9a)

$$\hat{e}(p) = H_1 e(p)$$ (4.9b)

$$r(p) = H_2 \hat{e}(p)$$ (4.9c)

with $H_1, H_2 : L_{2e}^M \rightarrow L_{2e}^M$, $r(p), \hat{e}(p), e(p) \in L_{2e}^M$ and $e(p) \in L_2^M$.

![Figure 4.3: Closed-loop feedback system for vector signals.](image)

The feedback system of vector signals is shown in figure 4.3. Following is the extension of Theorem 4.3 for system of vector signals.

**Theorem 4.5.** Consider the feedback system of (4.1). If

(a) $H_1 : e(p) \rightarrow \hat{e}(p)$ satisfies

$$\sum_{p=0}^{N} \left[ \sigma e(p)^T e(p) + e(p)^T \hat{e}(p) \right] \geq -\gamma$$ (4.10a)
(b) \( H_2 : \hat{e}(p) \rightarrow r(p) \) satisfies

\[
\sum_{p=0}^{N} \left[ \frac{\sigma}{2} r(p)^T r(p) - r(p)^T \hat{e}(p) \right] \leq -\omega \sum_{m=1}^{M} ||r_m(p), \hat{e}_m(p)||^2_N
\]

(4.10b)

for some \( \sigma, \gamma, \omega > 0 \), then the closed loop signals \( e(p), \hat{e}(p) \in L_2^M \).

**Proof.** From (4.10a) and (4.9a), we get

\[
\sum_{p=0}^{N} \left[ \frac{\sigma}{2} (e(p) - r(p))^T (e(p) - r(p)) + (e(p) - r(p))^T \hat{e}(p) \right] \geq -\gamma
\]

\[
\implies \sum_{p=0}^{N} \left[ \frac{\sigma}{2} e(p)^T e(p) + \frac{\sigma}{2} r(p)^T r(p) - \sigma e(p)^T r(p) + e(p)^T \hat{e}(p) - r(p)^T \hat{e}(p) \right] \geq -\gamma
\]

(4.11)

Combining (4.10b) and (4.11)

\[
-\omega \sum_{m=1}^{M} ||r(p), \hat{e}(p)||^2_N
\]

\[
+ \sum_{p=0}^{N} \sum_{m=1}^{M} \left[ \frac{\sigma}{2} \epsilon_m(p)^2 - \sigma \epsilon_m(p) r_m(p) + \epsilon_m(p) \hat{e}_m(p) \right] \geq -\gamma
\]

(4.12)

Using the Schwartz inequality

\[
\sum_{m=1}^{M} \left[ -\omega \ ||r_m(p), \hat{e}_m(p)||^2_N + \frac{\sigma}{2} \ ||\epsilon_m(p)||^2 \right]
\]

\[
+ \sigma \ ||\epsilon_m(p)|| \ ||r_m(p)|| + \ ||\epsilon_m(p)|| \ ||\hat{e}_m(p)|| \right] \geq -\gamma
\]

\[
\implies \sum_{m=1}^{M} \left[ \omega \ ||r_m(p), \hat{e}_m(p)||^2_N - \sigma \ ||\epsilon_m(p)||_N \ ||r_m(p)||_N \right.
\]

\[
- \ ||\epsilon_m(p)||_N \ ||\hat{e}_m(p)||_N \left. \right] \leq \gamma + \frac{\sigma}{2} \sum_{m=1}^{M} ||\epsilon_m(p)||^2_N
\]

(4.13)

Following the same argument of contradiction for Theorem 4.3, we have \( r_m(p), \hat{e}_m(p), \epsilon_m \in L_2 \ \forall m \). Therefore, \( r(p), \hat{e}(p), e \in L_2^M \).

\[ \square \]
4.3.2 Extension of Conic Sector Stability to $L_\infty$ space

The necessary condition for $L_2$ stability of feedback system (4.1) using Conic Sector Stability (cf. Section 4.3.1) is $\epsilon(p) \in L_2$. Usually in adaptive discrete control system, $\epsilon(p)$ represents the disturbance to the system. The condition that $\epsilon(p) \in L_2$ means that the disturbance to the system must die out to zero eventually. In practical system, this is never the case and not a practical condition for stability. However, it is reasonable in practical system to assume that the disturbance is bounded i.e. $\epsilon(p) \in L_{\infty}$. Therefore, $L_{\infty}$ relaxation of Conic Sector Stability is rather useful for practical system. The relaxation of Conic Sector Stability result in $L_{\infty}$ uses the idea of exponentially weighted signals (cf. Section 4.1.7) as is done in [103,105].

We will now present the $L_{\infty}$ extension of Conic Sector Stability results to $L_{\infty}$. The proof follows similar flow as in [103,105],

**Theorem 4.6.** If the exponentially weighted signals for the feedback system (4.1) $r(p)^{\beta}, \hat{e}(p)^{\beta}, e(p)^{\beta} \in L_2$ and the disturbance $\epsilon(p) \in L_{\infty}$, then $\epsilon(p) \in L_{\infty}$.

**Proof.** With $r(p)^{\beta}, \hat{e}(p)^{\beta}, e(p)^{\beta} \in L_2$, the map $H_I : \epsilon(p)^{\beta} \rightarrow \hat{e}(p)^{\beta}$ is $L_2$ stable. Therefore, there exists $K$, such that

$$\|\hat{e}(p)^{\beta}\|_N^2 \leq K \|\epsilon(p)^{\beta}\|_N^2 \tag{4.14}$$

By definition

$$\|\hat{e}(p)^{\beta}\|_N^2 = \sum_{p=0}^{N} (\beta^p \hat{e}(p))^2 \geq (\beta^N \hat{e}(N))^2 \tag{4.15}$$

and

$$\|\epsilon(p)^{\beta}\|_N^2 \leq \|\epsilon(p)\|_\infty^2 \sum_{p=0}^{N} \beta^{2p} \leq \|\epsilon(p)\|_\infty^2 \frac{\beta^{2N}}{1 - \beta^{-2}}, \quad \beta > 1 \tag{4.16}$$

Combining (4.14), (4.15) and (4.16), we can conclude that

$$\hat{e}(N)^2 \leq \|\epsilon(p)\|_\infty^2 \frac{K}{1 - \beta^{-2}}, \quad \beta > 1 \tag{4.17}$$

which completes the proof.  \hfill \square
It is not always possible to show stability of denormalized signals. In that case, we can prove $L_2$ stability of normalized signal and extend it to $L_\infty$ space. The extension of $L_2$ stability result of normalized signals in $L_\infty$ space requires specific form of normalization factor (cf. Section 4.1.6) defined as

$$
\rho(p) = \mu \rho(p - 1) + \max(|\phi(p)|^2, \bar{\rho}) \quad \bar{\rho} > 0, \mu \in (0, 1)
$$

(4.18)

where $\phi(p) = W[\tilde{e}(p)]$ such that $W : \tilde{e}(p) \rightarrow \phi(p)$ is $L_2$ stable. This form of normalization factor has unique property that for the map $H : x(p) \rightarrow y(p)$, $H[\mu^{1/2} \cdot ]$ and $H^n = \rho(p)^{1/2} H[\rho(p)^{1/2} \cdot ]$ share the same CONE $[103, 105]$.

We will now present the $L_\infty$ extension of Conic Sector Stability results of normalized signals to $L_\infty$ stability of denormalized signals. This stability result for denormalized signals requires the use of normalization factor of the form used in [105]. The following theorem is slightly modified form than [105] for extension of stability result in $L_\infty$ space.

**Theorem 4.7.** Consider the feedback system (4.1) and normalization factor (4.18). If $r^n(p)^\beta, \tilde{e}^n(p)^\beta, e^n(p)^\beta \in L_2$, and the disturbance $\epsilon(p) \in L_\infty$, then $\tilde{e}(p) \in L_\infty$.

Proof. With $r^n(p)^\beta, \tilde{e}^n(p)^\beta, e^n(p)^\beta \in L_2$, proceeding similarly as in Theorem 4.6, we obtain

$$
\tilde{e}^n(N)^2 \leq \|e^n(p)\|_\infty^2 \frac{K}{1 - \beta^{-2}}, \quad \beta > 1
$$

$$
\leq \|\epsilon(p)\|_\infty^2 \frac{K}{\bar{\rho}(1 - \beta^{-2})} = \psi
$$

(4.19)

Now, from (4.18)

$$
\rho(1) \leq \mu \rho(0) + |\phi(1)|^2 + \bar{\rho}
$$

$$
\implies \mu^{-1} \rho(1) \leq \rho(0) + \mu^{-1} |\phi(1)|^2 + \mu^{-1} \bar{\rho}
$$

and

$$
\rho(2) \leq \mu \rho(1) + |\phi(2)|^2 + \bar{\rho}
$$

$$
= \mu^2 \rho(0) + \mu |\phi(1)|^2 + \mu \bar{\rho} + |\phi(2)|^2 + \bar{\rho}
$$

$$
\implies \mu^{-2} \rho(2) \leq \rho(0) + (\mu^{-1} |\phi(1)|^2 + \mu^{-2} |\phi(2)|^2) + \bar{\rho}(\mu^{-1} + \mu^{-2})
$$
Continuing further with $\beta^2 = \mu^{-1}$, we get

$$\mu^{-N} \rho(N) \leq \rho(0) + \sum_{p=1}^{N} \mu^{-p} |\phi(p)|^2 + \tilde{\rho} \sum_{i=p}^{N} \mu^{-p}$$

$$= \rho(0) + \|\phi(p)\|_N^2 + \tilde{\rho} \frac{\mu^{-N} - 1}{1 - \mu}$$

$$\implies \mu^{-N} \rho(N) \leq \rho(0) + \|\phi(p)\|_N^2 + \mu^{-N} \frac{\tilde{\rho}}{1 - \mu} \tag{4.20}$$

From the definition of $L_2$ gain of the map $W^\beta : \tilde{e}(p) \rightarrow \phi(p)$, we have

$$\|\phi(p)\|_N^2 \leq \gamma_2 \|\tilde{e}(p)\|_N \quad \text{where} \quad \gamma_2 = \gamma_2(W^\beta) \tag{4.21}$$

Therefore

$$\mu^{-N} \rho(N) \leq \rho(0) + \{\gamma_2 \|\tilde{e}(p)\|_N\}^2 + \mu^{-N} \frac{\tilde{\rho}}{1 - \mu} \tag{4.22}$$

and by using (4.19), we get

$$\mu^{-N} \rho(N) \tilde{e}^a(N)^2 \leq \psi \left[ \rho(0) + \{\gamma_2 \|\tilde{e}(p)\|_N\}^2 + \mu^{-N} \frac{\tilde{\rho}}{1 - \mu} \right]$$

$$\leq \psi \left[ \rho(0) + \mu^{-N} \frac{\tilde{\rho}}{1 - \mu} + \gamma_2^2 \left( \mu^{-N} \tilde{e}(N)^2 + \|\tilde{e}(p)\|_{N-1}^2 \right) \right]$$

$$\implies \mu^{-N} (1 - \psi \gamma_2^2) \tilde{e}(N)^2 \leq \psi \left[ \rho(0) + \mu^{-N} \frac{\tilde{\rho}}{1 - \mu} + \gamma_2^2 \|\tilde{e}(p)\|_{N-1}^2 \right] \tag{4.23}$$

If $\psi$ satisfies $1 - \psi \gamma_2^2 > \mu$, we get

$$\mu^{-N} \tilde{e}(N)^2 \leq \frac{\psi}{1 - \psi \gamma_2^2} \left[ \rho(0) + \mu^{-N} \frac{\tilde{\rho}}{1 - \mu} \right] + \frac{\psi \gamma_2^2}{1 - \psi \gamma_2^2} \sum_{p=0}^{N-1} \mu^{-p} \tilde{e}(p)^2$$

$$= K_0 + K_1 \mu^{-N} + \frac{\psi \gamma_2^2}{1 - \psi \gamma_2^2} \sum_{p=0}^{N-1} \mu^{-p} \tilde{e}(p)^2 \tag{4.24}$$

where $K_0 = \frac{\psi \rho(0)}{1 - \psi \gamma_2^2} > 0$, $K_1 = \frac{\psi \gamma_2^2}{1 - \psi \gamma_2^2} \frac{\tilde{\rho}}{1 - \mu} > 0$
Now, applying Bellman-Gronwall lemma (cf.§ B.1), we get

\[ \mu^{-N} \dot{e}(N)^2 \leq K_0 + K_1 \mu^{-N} + \frac{\psi \gamma_2^2}{1 - \psi \gamma_2^2} \sum_{p=0}^{N-1} K_1 \mu^{-p} \prod_{j=p+1}^{N-1} \left[ 1 - \frac{\psi \gamma_2^2}{1 - \psi \gamma_2^2} \right] \]

\[ = K_0 + K_1 \mu^{-N} + \frac{\psi \gamma_2^2}{1 - \psi \gamma_2^2} \sum_{p=0}^{N-1} K_1 \mu^{-p} \left[ 1 - \frac{\psi \gamma_2^2}{1 - \psi \gamma_2^2} \right]^{N-p-1} \]

\[ = K_0 + K_1 \mu^{-N} \left\{ 1 + \psi \gamma_2^2 \sum_{p=0}^{N-1} \left[ \frac{\mu}{1 - \psi \gamma_2^2} \right]^{N-p-1} \right\} \tag{4.25} \]

If \( 1 - \psi \gamma_2^2 > \mu \) is true, the term inside the brackets is less than 1 and the series is convergent, therefore, we can conclude that \( \dot{e}(p) \in L_\infty \). Now by using (4.19) in \( 1 - \psi \gamma_2^2 > \mu \), we get

\[ 1 - \|e(p)\|_\infty^2 \frac{K}{\rho(1 - \beta^{-2})} \gamma_2^2 > \mu \]

\[ \Rightarrow (1 - \mu)^2 > K \gamma_2^2 \frac{1}{\rho} \|e(p)\|_\infty^2 \tag{4.26} \]

Since \( \mu \in (0,1) \), there exists a \( \check{\rho} \) which will make (4.26) true. This completes the proof. \( \square \)
Chapter 5

SpikeProp with Adaptive Learning Rate

Stability of learning is a major concern during SpikeProp learning. An observable of unstable nature in SpikeProp learning is the sudden jump in learning cost function, $E$. These sudden jump in learning cost is known as surge. A typical example of surge during SpikeProp learning is illustrated in figure 5.1. In a typical learning process using SpikeProp, surges occur quite often. Often times they are big enough that they drastically change the learning trajectory and cause failure in learning. Even if they are not big enough, they constantly change the learning ravine and consequently slow the learning process.

\[0 100 200 300 400 500 600 700] \quad E\quad \eta = 0.01.

The main source of instability during SpikeProp training is due to the complex, non-linear relationship between membrane potential of neuron and spike firing
time of the neuron. Furthermore, it is necessary to have a mix of excitatory and inhibitory synaptic connections, i.e. positive and negative weights, to learn useful results [2, 99]. The issue is that the presence of negative weights means that the membrane potential is not smooth, but has undulating nature. This is referred to as twisted dynamics (see [99] for illustration of twisted dynamics). The hair trigger issue (cf. Chapter 3.3.2) also lends to the unstable nature of SpiekProp learning, causing large change in weights as a result changing the learning trajectory substantially.

It is plausible to think that use of small learning rate, $\eta$, in SpikeProp learning would avoid the stability issues in SpikeProp with due trade-off in speed of learning. Contrary to this, in the in-depth study on the occurrence of surges during SpikeProp learning by Takase et al. [99] shows that along with large learning rates, small learning rates also means more surge and unstable learning. This is a surprising result. It means that simply using a small enough learning rate does not curb the stability problems of SpikeProp. A learning rate that is neither too large, nor too small is warranted for stable learning in SpikeProp.

In this Chapter, we will present SpikeProp with adaptive learning rate (SpikePropAd) which will vary the learning rate depending on the current state of SNN. The formulation of adaptive learning rate cancels the effect of dead neuron problem. We will prove the stability of learning using weight convergence analysis. We will show the efficacy of SpikePropAd based on simulation results on different standard datasets and compare it with other relevant methods as well. Finally, we will present our thoughts on SpikePropAd learning parameter. A preliminary version of this learning rule was proposed in [39] and extended in [40].

## 5.1 The SpikePropAd Algorithm

In this section, we will present the SpikePropAd algorithm for learning weights in multilayer feedforward SNN as an extension to SpikeProp algorithm. The adaptive learning rate is defined for each individual neuron and varies depending on the current state of the neuron. It is defined as follows:

$$
\eta_j^{opt}(\mathbf{p}) = \begin{cases} 
\tanh \left( \frac{u_j'(t_j)(u_j'(t_j) - \lambda)}{\|y_{ji}\|^2} \right) & \text{if } u_j'(t_j) > \lambda \\
0 & \text{otherwise}
\end{cases}
$$

\( (5.1) \)
where $\lambda > 0$ is a positive bound on minimum slope of membrane potential to allow for weight update. The role of tanh function here is to keep the learning rate $\eta^\text{opt}_j(p)$ bounded between 0 and 1 so that the learning rate does not grow into an impractical value. The SpikePropAd weight update rule at iteration $p$ follows the same form as (3.32) and (3.33). Therefore

$$W_{jf}(p + 1) = W_{jf}(p) - \eta^\text{opt}_j(p) \text{diag}(\delta_j) Y_{jf}$$

where $\eta^\text{opt}_j(p) = \text{diag}([\cdots, \eta^\text{opt}_j(p), \cdots]^T)$ and the SpikePropAd weight update rule for a neuron $N_j$ is

$$w_{ji}(p + 1) = w_{ji}(p) - \eta^\text{opt}_j(p) \delta_j y_{ji}.$$ (5.3)

Note that the quadratic term of $u'_j(t_j)$ in numerator of $\eta^\text{opt}_j(p)$ suppresses the $u'_j(t_j)$ term in the denominator of $\delta_j$. Further, when $u'_j(t_j) \leq \lambda$, the learning rate is 0. As a result, the hair trigger problem is not an issue in SpikePropAd.

The algorithm for SpikePropAd learning is presented in Algorithm 4.

From the point of view of implementation, there is not much additional computational overhead in using SpikePropAd. The additional step compared to SpikeProp learning is the computation of adaptive learning rate term, whose parameters $u'_j(t_j)$ and $y^{(k)}_ji(t_j)$ are precomputed for SpikeProp as well. Further, compared to forward propagation, the weight update process requires negligible computational time (cf. Chapter 3.2).

Next we will show that learning using SpikePropAd is convergent in the sense of weight parameters.

## 5.2 Weight Convergence Analysis of SpikePropAd

Consider the ideal weight matrices for hidden layer and output layer: $\overline{W}_{HI}$ and $\overline{W}_{OH}$. Then the weight vector for a hidden layer neuron and an output layer neuron is $\overline{w}_{hI}$ and $\overline{w}_{oH}$ respectively. The ideal membrane potential for hidden layer neuron and output layer neuron is

$$\overline{u}_j(t) = \overline{w}_{ji}^T \overline{y}_{ji}(t)$$

(5.4)
Algorithm 4 SpikePropAd Learning

1: set up network connections
2: initialize weights
3: repeat
4: $MSE \leftarrow 0$
5: for all training sample do $\triangleright$ assume $m$ samples
6: for all $N_i \in I$ do
7: $t_i \leftarrow \text{input}_i$
8: end for
9: for all $N_j \in \mathcal{H}, \mathcal{O}$ do
10: $t_j \leftarrow \text{CALCTFIRE}(N_j)$ $\triangleright$ algorithm 1
11: end for
12: for all $N_j \in \mathcal{O}, \mathcal{H}$ do
13: calculate $\delta_j$ $\triangleright$ equations (3.29) & (3.31)
14: calculate $\eta^\text{opt}_j$ $\triangleright$ equation (5.1)
15: for $k \leftarrow 1 : |\mathcal{K}|$ do
16: $w_{ji}^{(k)} \leftarrow w_{ji}^{(k)} - \eta^\text{opt}_j \delta_j y_j^{(k)}(t_j)$ $\triangleright i \in \Gamma_j$
17: end for
18: if $N_j \in \mathcal{O}$ then
19: $MSE \leftarrow MSE + E/m$ $\triangleright$ equation (3.26)
20: end if
21: end for
22: end for
23: until $MSE < \text{TOL}$ OR $\text{MAX\_EPOCH}$ is exceeded

and the desired output firing time vector is

$$\hat{t}_O = \bar{t}_O = f_d\left(\bar{W}_{OH}Y_{OH} \left(f_d(\bar{W}_{HI}Y_{HI}(t_I))^T\right)\right)^T \tag{5.5}$$

The approach for weight convergence analysis of SpikePropAd uses the Lyapunov function of the same form in [108, 109]. The Lyapunov function is defined as the difference between norm of actual and ideal weight vectors in two adjacent iterations. If we show Lyapunov stability of the weight based Lyapunov function, the current weight matrices or vectors always move towards ideal weight, or at least maintain the same distance from ideal weight, at each iteration. We will consider individual learning rate for each neuron $\eta_j(p)$ and relate it to the optimal learning rate $\eta^\text{opt}_j(p)$ later.

Denote the difference between actual and ideal weight matrix as $\tilde{w}_{ji}(p) = w_{ji}(p) - \bar{w}_{ji}$. 

NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE
Now, define the Lyapunov function

\[ V_j = \left\| \tilde{w}_{j}(p) \right\|^2 \]  
\[ \Delta V_j = \left\| \tilde{w}_{j}(p+1) \right\|^2 - \left\| \tilde{w}_{j}(p) \right\|^2 \]  

**Theorem 5.1.** The weight vector \( w_{j} \) is convergent in the sense of Lyapunov function \( V_j \) such that \( \Delta V_j \leq 0 \) if for a fixed parameter \( \lambda \geq 0 \), the SpikeProp learning rate of neuron \( N_j \) at iteration \( p \) satisfies

\[ 0 \leq \eta_j(p) \leq \begin{cases} 2\frac{u_j'(t_j)(u_j'(t_j) - \lambda)}{\|y_j\|^2} & \text{if } u_j'(t_j) > \lambda \\ 0 & \text{otherwise} \end{cases} \]  

**Proof.** Consider

\[ \Delta V_j = \left\| \tilde{w}_{j}(p+1) \right\|^2 - \left\| \tilde{w}_{j}(p) \right\|^2 \\
= \left\| w_{j}(p+1) - \tilde{w}_{j} \right\|^2 - \left\| w_{j}(p) - \tilde{w}_{j} \right\|^2 \\
= (w_{j}(p+1) - \tilde{w}_{j})^T (w_{j}(p+1) - \tilde{w}_{j}) \\
- (w_{j}(p) - \tilde{w}_{j})^T (w_{j}(p) - \tilde{w}_{j}) \\
= w_{j}(p+1)^T w_{j}(p+1) + \tilde{w}_{j}^T \tilde{w}_{j} - 2 \tilde{w}_{j}^T w_{j}(p+1) \\
- w_{j}(p)^T w_{j}(p) - \tilde{w}_{j}^T \tilde{w}_{j} + 2 \tilde{w}_{j}^T w_{j}(p) \\
= w_{j}(p+1)^T w_{j}(p+1) - 2 \tilde{w}_{j}^T w_{j}(p+1) \\
- w_{j}(p)^T w_{j}(p) + 2 \tilde{w}_{j}^T w_{j}(p) \\
\]

by using (5.3), we get

\[ \Delta V_j = w_{j}(p)^T w_{j}(p) + \eta_j(p)^2 \delta_j^2 y_{ji}^T y_{ji} \\
- 2\eta_j(p) \delta_j w_{j}(p)^T y_{ji} + 2\eta_j(p) \delta_j \tilde{w}_{j}^T y_{ji} - w_{j}(p)^T w_{j}(p) \]

by using (3.11) and (5.4), we get

\[ \Delta V_j = \eta_j(p)^2 \delta_j^2 \left\| y_{ji} \right\|^2 - 2\eta_j(p) \delta_j (u_j(t_j) - \tilde{u}_j(t_j)) \]  

(5.9)
By definition, we have

\[ e_o = \tilde{t}_o - t_o \]
\[ = \tilde{t}_o - t_o \]  

for output layer neuron. Similarly, the error for hidden layer neuron can be written as

\[ e_h = \tilde{t}_h - t_o \]
\[ = -\frac{\partial E_h}{\partial t_h} \approx -\frac{\partial E}{\partial t_h} = \frac{\partial t_O}{\partial t_h} e_O \]  

where \( E_h = \frac{1}{2} e_h^2 \) and the backpropagation estimate of the local error of hidden layer neuron is \( e_h \approx -\frac{\partial E}{\partial t_h} \). Note that, the membrane potential must be equal to the threshold value when the neuron fires i.e. \( u_j(t_j) = \bar{u}_j(\tilde{t}_j) = \vartheta \). Then

\[ u_j(t_j) - \bar{u}_j(t_j) = \bar{u}_j(\tilde{t}_j) - \bar{u}_j(t_j) = \bar{u}_j(\xi) e_j \]  

where \( \min(t_j, \tilde{t}_j) < \xi < \max(t_j, \tilde{t}_j) \). From (3.29), (3.31) and (5.12), (5.9) becomes

\[ \Delta V_j = \eta_j(p)^2 \frac{e_j^2}{u_j'(t_j)^2} \left\| y_{ji} \right\|^2 - 2 \eta_j(p) e_j^2 \bar{u}_j(\xi) u_j'(t_j) \]
\[ = \frac{\eta_j(p)^2 e_j^2}{u_j'(t_j)^2} \left( \eta_j(p) \left\| y_{ji} \right\|^2 - 2 \bar{u}_j(\xi) u_j'(t_j) \right) \]  

The kernel function \( \varepsilon(\cdot) \) and its derivative is bounded. Further, if the difference between \( u_j'(t_j) \) and \( \bar{u}_j'(\xi) \) is bounded by a positive parameter \( \lambda \) i.e.

\[ |u_j'(t_j) - \bar{u}_j'(\xi)| \leq \lambda \]  

\[ \Rightarrow u_j'(t_j) - \lambda \leq \bar{u}_j'(\xi) \leq u_j'(t_j) + \lambda \]  

Then from (5.13)

\[ \Delta V_j \leq \frac{\eta_j(p)^2 e_j^2}{u_j'(t_j)^2} \left( \eta_j(p) \left\| y_{ji} \right\|^2 - 2 u_j'(t_j)^2 + 2 \lambda u_j'(t_j) \right) \leq 0 \]
\[ \Rightarrow 0 \leq \eta_j(p) \leq \begin{cases} \frac{2 u_j'(t_j)(u_j'(t_j) - \lambda)}{\left\| y_{ji} \right\|^2} & \text{if } u_j'(t_j) > \lambda \\ 0 & \text{otherwise} \end{cases} \]
Corollary 5.1.1. The SpikePropAd adaptive learning rate (5.1) satisfies Theorem 5.1.

Proof. Note that $0 \leq \tanh x \leq x, \forall x \geq 0$ and the result follows from (5.8).

5.3 Performance Comparison of SpikePropAd

In this section, we will compare the performance of SpikeProp algorithm, the faster variant of SpikeProp – RProp and the adaptive learning rate modification to SpikeProp (SpikePropAd) proposed in this Chapter. We will compare the methods based on results of simulations on four benchmark datasets: XOR, Fisher’s Iris data, Wisconsin Breast Cancer data [110] and Statlog (Landsat Satellite) data [110]. As discussed in Chapter 3.7, we will compare the simulation results based on their convergence rate (cf. Chapter 3.7.2), average number of epochs required to reach a tolerance value (cf. Chapter 3.7.3), training and testing accuracy (cf. Chapter 3.7.4) and average learning curve (cf. Chapter 3.7.5).

The SNN architecture used is a three layer architecture as described in Chapter 3.1. The weights are initialized randomly based on normal distribution using the method described in Chapter 3.3.2. We will now present the simulation results on each of the datasets and discuss the performance results.

5.3.1 XOR problem

XOR classification is a basic nonlinear classification problem. The simulation parameters are listed below.

| Network Architecture | $|\mathcal{I}| = 3$, $|\mathcal{H}| = 5$, $|\mathcal{O}| = 1$ & $|\mathcal{K}| = 16$ |
|----------------------|--------------------------------------------------|
| Synaptic Delays      | $1 : 1 : 16$                                     |
| SRM time constant    | $\tau_s = 7$                                    |
| Activation Threshold | $\vartheta = 1$                                  |
| Simulation Interval  | $0 : 0.01 : 25$ units                            |
| Weight Initialization| $t_{\min} = 0.1$ units & $t_{\max} = 10$ units  |
| Dead Neuron Boosting | boost by 0.05                                    |

The binary inputs of XOR are encoded based on early and delayed spike firing times. A ‘1’ in input is represented by spike at 0 units and a ‘0’ in input is represented by spike at 6 units. Although there are only two inputs in XOR
classification, a third input is used to denote the reference start time and always fires at $t = 0$ units. Similarly, ‘1’ and ‘0’ in output neuron is represented by spike at 10 and 16 time units respectively.

1000 different simulations for XOR problem were conducted using SpikeProp ($\eta = 0.1 & 0.01$), RProp ($\eta^+ = 1.3$, $\eta^- = 0.5$) and SpikePropAd ($\lambda = 1, 0.1, 0.01 & 0.001$). A maximum of 5000 epochs is used to label the training instance as converging or non-converging. Figure 5.2 shows the plot of convergence rate versus the mean square error achieved for different methods. It is clear that SpikePropAd ($\lambda = 0.1, 0.01 & 0.001$) have better convergence rate compared to other methods over the entire mean square error tolerance range. Note, however, for $\lambda = 1$ the convergence rate is zero always. This is because the constraint on the slope in (5.1) is not satisfied and the learning rate is almost always zero as a result. For $\lambda = 0.1, 0.01 & 0.001$ the convergence rate performance is practically indistinguishable.

![Figure 5.2: Convergence Rate plot for learning XOR problem using SpikeProp, RProp and SpikePropAd.](image)

In figure 5.3, the average number of epochs required to converge to different mean square error tolerance is plotted for different methods. The lower limit of error bar indicate the minimum number of epochs required and the upper error bar indicates the standard deviation of the epochs required. The mean and standard deviation of epochs are calculated out of the instances that converged within 5000 epochs. It is evident from figure 5.3 that SpikePropAd ($\lambda = 0.01 & 0.001$) have better performance among the lot in terms of number of epochs required to reach a mean square error value. Note that the convergence rate for SpikePropAd ($\lambda = 1$) is always zero, indicating no convergence at all.
Figure 5.3: Average number of epochs required to learn XOR problem using SpikeProp, Rprop and SpikePropAd.

Figure 5.4: Average learning curves during XOR problem learning using SpikeProp, RProp and SpikePropAd.

Figure 5.4 compares average cost function required for each of the methods during 1000 different trials. SpikePropAd ($\lambda = 0.001$) has the best performance in terms of average cost function. Figure 5.5 shows a typical learning curve for XOR training. One can clearly see that RProp method performs well at first; however, during RProp training the network mostly jumps into non-firing mode\(^1\) as can be seen in figure 5.5 and it can barely recover from that state even with the employment of boosting. This explains the rise in average error of RProp in figure 5.4 after some epochs. Note that the surge jumps are small in the case of SpikePropAd and

\(^{1}\)When the output neuron does not fire during training, we associate the firing time of the neuron to a large value (10,000) compared to the simulation time. This results in large training error; however, during the calculation of average error for each method, whenever the output neuron does not fire, the cost is set to zero so that the influence of dead output neuron does not overwhelm the average cost curve.
Figure 5.5: Typical learning curves during XOR pattern learning using SpikeProp, RProp and SpikePropAd.

therefore do not affect the learning process substantially.

It is evident from the results that SpikePropAd shows similar performance for \( \lambda = 0.1, 0.01 & 0.001 \), with slightly better performance for smaller \( \lambda = 0.001 \). For \( \lambda = 1 \), the learning rate was almost always found to be zero, resulting in no significant learning at all.

Figure 5.6: Trace of learning rate of output layer neuron in a typical XOR learning using SpikePropAd with \( \lambda = 0.01 \).

Figure 5.7: Trace of learning rate of a hidden layer neuron in a typical XOR learning using SpikePropAd with \( \lambda = 0.01 \).

Figure 5.6 and figure 5.7 show the trace of learning rate in each iteration for output layer neuron and a hidden layer neuron for SpikePropAd with \( \lambda = 0.01 \) respectively. The learning rates in both the hidden layer and output layer neurons
show a periodic rise and fall in the learning rate values. It is interesting to note that the period is 4 iterations which is the number of training samples for XOR simulation. The learning rates fluctuate between 0 and 1 for hidden layer neuron whereas it is mostly on the higher side for output layer neuron, thus explaining quicker convergence.

### 5.3.2 Fisher’s Iris classification

Fisher’s Iris data consists of four numeric features based on different dimensions of the flower and the task is to delineate between three species of Fisher’s Iris flower \textit{viz. Iris Setosa, Iris Virginica and Iris Versicolor}. The simulation parameters for Fisher’s Iris classification are listed below.

| Network Architecture | $|\mathcal{I}| = 16$, $|\mathcal{H}| = 7$, $|\mathcal{O}| = 1$ & $|\mathcal{K}| = 16$ |
|----------------------|-----------------------------------------------------|
| Synaptic Delays      | 1 : 1 : 16                                           |
| SRM time constant    | $\tau_s = 7$                                        |
| Activation Threshold | $\vartheta = 1$                                      |
| Simulation Interval  | 0 : 0.01 : 25 units                                  |
| Weight Initialization| $t_{\text{min}} = 0.1$ units & $t_{\text{max}} = 10$ units |
| Dead Neuron Boosting | boost by 0.05                                       |

Since the features of this dataset are not binary, we use population encoding method (cf. Chapter 2.3.1) with 4 Gaussian receptive fields to encode the input features into spike times between $t = 0$ to 4 units for 16 input neurons. The output class was encoded by spikes at 5, 10 & 15 units for the three Iris types.

Simulations were performed on the encoded data using SpikeProp ($\eta = 0.01$ & 0.0001), RProp ($\eta^+ = 1.3$, $\eta^- = 0.5$) and SpikePropAd ($\lambda = 1$, 0.1, 0.01 & 0.001) for 100 different initial weights. We considered a maximum number of 1500 epochs and those instances that failed to meet tolerance by 1500 epochs were deemed non-converging. In figure 5.8, plot of convergence rate versus the mean square error is plotted. It is clear that SpikePropAd ($\lambda = 0.01$) has the best convergence rate among the methods. As a whole SpikePropAd demonstrates better convergence performance compared to other methods with exception for $\lambda = 0$ which has zero convergence rate.

Mean epochs required to converge to different mean square error is plotted in figure 5.9 for various methods. The lower point in error bar denotes the minimum number of epoch required and the upper length of the error bar indicates the stan-
The upper error bar position indicates the maximum accuracy achieved and the corresponding to different final mean square values for all the methods.

Figure 5.10 and figure 5.11 show the plot of training accuracy and testing accuracy respectively corresponding to different final mean square values for all the methods. The upper error bar position indicates the maximum accuracy achieved and the length of lower error bar indicates the standard deviation in accuracy for both training and testing accuracy. SpikePropAd ($\lambda = 0.1$, 0.01 & 0.001) have good performance compared to other methods.
training accuracy and consistent testing accuracy, indicating good generalization performance. However, the testing accuracy is slightly lower for smaller tolerance values. RProp on the other hand shows big dip in both training and testing accuracy with decrease in final mean square error, which is strange! The accuracy performance of SpikeProp is similar to SpikePropAd. But the real advantage of SpikePropAd is in the number of epochs required to achieve such performance.

A comparison of average mean square error attained by each of the methods during 100 different trials is shown in figure 5.12 and figure 5.13 shows a typical learning curve for Fisher’s Iris classification learning. Similar to the observation for XOR problem, it can be seen from figure 5.13 that frequent dead neuron problem plagues RProp, as a result, it looses its initial advantage as depicted by the rise of its
5.3. Performance Comparison of SpikePropAd

Figure 5.12: Average learning curves during Fisher’s Iris classification using Spike-Prop, RProp and SpikePropAd.

Figure 5.13: Typical learning curves during Fisher’s Iris classification using Spike-Prop, RProp and SpikePropAd.

average cost in figure 5.12. The performance of SpikePropAd is comparable for $\lambda = 0.1$, 0.01 & 0.001 and is superior to other methods in terms of average error. Also note the horizontal average mean square error when $\lambda = 1$. Surges are also minimal for SpikePropAd in figure 5.13.

For the parameter $\lambda$ in SpikePropAd, there is nothing much to choose between 0.1, 0.01 and 0.001. $\lambda = 0.01$ shows marginal edge among other values in terms of number of epochs required. When $\lambda = 1$, the simulation results show that there is no learning which is similar to the result observed for XOR classification.

The adaptation of learning rate in SpikePropAd during Fisher’s Iris learning is shown in figure 5.14 for output layer neuron and figure 5.15 for a hidden layer...
neuron. The learning rate of output layer neuron is mostly close to one and decreases for a small period once in a while. The hidden layer neuron learning rate on the other hand changes quite a lot throughout. The periodic behaviour as observed in the case of XOR classification (see figure 5.6 and figure 5.7) is not evident in this case.

For a sense of learning comparison between SNN learning methods and standard backpropagation on MLPs, the results for Fisher’s Iris classification learning have been summarized in Table 5.1.

### 5.3.3 Wisconsin Breast Cancer classification

Wisconsin Breast Cancer data [110] consists of nine features that characterize the cell nucleus that are malignant or benign. The simulation parameters for this classification problem are listed below.
Network Architecture \( |Z| = 64, |H| = 15, |O| = 2 \) & \( |K| = 16 \)  
Synaptic Delays \( 1 : 1 : 16 \) 
SRM time constant \( \tau_s = 7 \) 
Activation Threshold \( \vartheta = 1 \) 
Simulation Interval \( 0 : 0.01 : 25 \) units 
Weight Initialization \( t_{\text{min}} = 0.1 \) units & \( t_{\text{max}} = 10 \) units 
Dead Neuron Boosting boost by 0.05

We use population encoding with 7 Gaussian receptive fields (cf. Chapter 2.3.1) to encode the nine input features of Wisconsin Breast Cancer dataset into spike times between \( t = 0 \) to 4. The output is based on winner-takes-all principle between the two output neurons. For benign case, the output spikes are encoded at 10 units for first neuron and 16 units for second output neuron and the reverse for malignant case.

100 simulations were performed on the encoded data using SpikeProp \( (\eta = 0.01 \) & 0.0001), RProp \( (\eta^+ = 1.3, \eta^- = 0.5) \) and our SpikePropAd \( (\lambda = 1, 0.1, 0.01 \) & 0.001) starting from randomly initialized weights as described in Chapter 3.3.2. For this case, we considered 1000 epochs before labeling the simulation instance as non-converging. Figure 5.16 shows the plot of convergence rate for various mean square error tolerance values using different methods. SpikePropAd \( (\lambda = 0.1) \) exhibits better convergence rate among other methods. Note that for \( \lambda = 1 \), SpikePropAd does not converge at all.

![Figure 5.16: Convergence Rate plot for learning Wisconsin Breast Cancer data using SpikeProp, RProp and SpikePropAd.](image)

In figure 5.17, the mean number of epoch required to converge to various mean square error is plotted for different methods. The lower error bar corresponds to
Ch. 5. SPIKEPROP WITH ADAPTIVE LEARNING RATE

5.3. PERFORMANCE COMPARISON OF SPIKEPROPAD

![Graph showing Epochs to Converge vs MSE Tolerance](image)

**Figure 5.17**: Average number of epochs required to learn Wisconsin Breast Cancer data using SpikeProp, Rprop and SpikePropAd.

The minimum number of epoch and the length of upper error bar indicates the standard deviation of epochs. The mean epoch and standard deviation of epochs are calculated out of the instances that converged within 1000 epochs. From the figure, it is clear that RProp performs better in terms of number of epochs required to reach a given mean square error among the instances that converged. One should also note that the convergence of RProp is not as good as SpikePropAd. It is interesting to note that for $\eta = 0.0001$, SpikeProp has good convergence rate and performs well in terms of number of epochs required to converge as well.

![Graph showing Training Accuracy vs MSE Tolerance](image)

**Figure 5.18**: Training accuracy in Wisconsin Breast Cancer classification using SpikeProp, RProp and SpikePropAd.

Plots of Training accuracy and Testing accuracy versus final mean square error is shown in figure 5.18 and figure 5.19 respectively. The upper point of the error bar indicates the maximum accuracy achieved and the length of lower error bar...
Figure 5.19: Testing accuracy in Wisconsin Breast Cancer classification using SpikeProp, RProp and SpikePropAd.

indicates the standard deviation of both training accuracy and testing accuracy. The accuracy performance of SpikeProp and SpikePropAd is similar and show good generalization performance. On the other hand, RProp method has slightly worse performance in terms of accuracy even though it is faster. The decrease in training accuracy when the mean square error tolerance becomes smaller is also evident in this case similar to Fisher’s Iris classification.

Figure 5.20: Average learning curves during Wisconsin Breast Cancer classification using SpikeProp, RProp and SpikePropAd.

Figure 5.20 compares the average cost function attained by each of the methods and a typical learning curve is presented in figure 5.21. It is again clear that the frequent dead neuron occurrence causes the average cost for RProp to fluctuate randomly. SpikePropAd ($\lambda = 0.1$, $0.01$ & $0.001$) show very similar performance and exhibit better results compared to SpikeProp and RProp in terms of average
mean square error achieved. Note the horizontal average learning curve when \( \lambda = 1 \) for SpikePropAd. Also observe the surge jumps in Figure 5.21. There are minimal surges in SpikePropAd.

Once again, we observe that SpikePropAd with \( \lambda = 0.1, 0.01 \) & 0.001 demonstrate comparable performance. \( \lambda = 0.1 \), however, shows slightly better convergence performance. For \( \lambda = 1 \), the adaptive learning rate is almost always zero, similar to the observations for XOR and Fisher’s Iris benchmark.

The trace of learning rate for an output neuron and a hidden neuron for first 1000 iterations using SpikePropAd during Wisconsin Breast Cancer classification learning is plotted in Figure 5.22 and Figure 5.23 respectively. The output layer neuron learning rate again assumes higher values compared to hidden layer neuron learning rates. However, the learning rate varies over the range of 0 to 1 and no certain pattern is evident.

For a sense of learning comparison between SNN learning methods and standard
backpropagation on MLPs, the results for Wisconsin breast cancer classification learning have been summarized in Table 5.1.

5.3.4 Statlog (Landsat Satellite) data classification

The Statlog (Landsat Satellite) data \[110\] consists of the multi-spectral values of pixels in a satellite image. Each pixel patch corresponds to a certain soil or vegetation type and the aim is to classify between those six classes. The simulation parameters for this classification problem are listed below.

| Network Architecture | $|I| = 101, |H| = 25, |O| = 6 \& |K| = 16 |
|----------------------|----------------------------------|
| Synaptic Delays      | $1 : 1 : 16$                     |
| SRM time constant    | $\tau_s = 7$                     |
| Activation Threshold | $\vartheta = 1$                  |
| Simulation Interval  | $0 : 0.01 : 25$ units            |
| Weight Initialization| $t_{\text{min}} = 0.1$ units $\&$ $t_{\text{max}} = 10$ units |
| Dead Neuron Boosting | boost by 0.05                    |

The input features were encoded into spikes using 25 Gaussian receptive fields to a time frame of $t = 0$ to $t = 4$. The output are interpreted on a first to spike basis.

10 simulations were performed on the encoded data using SpikeProp ($\eta = 0.1 \& 0.01$), RProp ($\eta^+ = 1.3, \eta^- = 0.5$) and our SpikePropAd ($\lambda = 1, 0.1, 0.01 \& 0.001$) starting from randomly initialized weights as described in Chapter 3.3.2. A maximum of 1000 epochs were considered before labeling the simulation instance as non-converging. Figure 5.24 shows the plot of convergence rate for various mean square error tolerance values using different methods. SpikePropAd ($\lambda = 0.1$) exhibits better convergence rate among other methods. SpikePropAd ($\lambda = 0.01 \& 0.001$) also depict the same convergence rate as for $\lambda = 0.1$, and therefore are not plotted. Also note that for $\lambda = 1$, SpikePropAd does not converge at all.
The mean number of epoch required to converge to different mean square error is shown in figure 5.25. The lower error bar corresponds to the minimum number of epoch and the length of upper error bar indicates the standard deviation of epochs. The mean epoch and standard deviation of epochs are calculated out of the instances that converged within 1000 epochs. SpikeProp on the whole shows better result in terms of number of epochs, \( \lambda = 0.01 \) being slightly better than \( \lambda = 0.1 \) & 0.001. Note the opposite trend for SpikeProp (\( \eta = 0.01 \)). This is because the method converged 5, 3 and 1 times to MSE error of 10, 8 and 6 respectively. The instance that converged to MSE of 6 required substantially less number of epochs compared to other runs, as a result, this false trend is seen in figure 5.25. A larger number of trial should eliminate this false trend of number of epoch and the length of upper error bar indicates the standard deviation of epochs to converge.
of epochs required for SpikeProp ($\eta = 0.01$).

![Graph showing training accuracy versus MSE tolerance for different algorithms.]

Figure 5.26: Training accuracy in Statlog (LandSat Satellite) classification using SpikeProp, RProp and SpikePropAd.

![Graph showing testing accuracy versus MSE tolerance for different algorithms.]

Figure 5.27: Testing accuracy in Statlog (LandSat Satellite) classification using SpikeProp, RProp and SpikePropAd.

Training accuracy and Testing accuracy versus final mean square error is plotted in figure 5.26 and figure 5.27 respectively. The upper point of the error bar indicates the maximum accuracy achieved and the length of lower error bar indicates the standard deviation of both training accuracy and testing accuracy. SpikePropAd demonstrates better training accuracy results and testing accuracy statistics consistent with training accuracy results. The decrease of training and testing accuracy for RProp for the smaller tolerance value can again be seen in figure 5.26 & 5.27, an observation seen in Section 5.3.2 & 5.3.3 as well.

Figure 5.28 shows the comparison of average cost function attained by each of the methods and a typical learning curve is presented in figure 5.29. Dead neuron
Figure 5.28: Average learning curves during Statlog (Landsat Satellite) classification using SpikeProp, RProp and SpikePropAd.

Figure 5.29: Typical learning curves during Statlog (Landsat Satellite) data classification SpikeProp, RProp and SpikePropAd.

occurrence is clearly visible for RProp in the learning curves. SpikePropAd ($\lambda = 0.1, 0.01 & 0.001$) show very similar performance which is better compared to SpikeProp and RProp in terms of average mean square error achieved. Once again, note the horizontal average learning curve when $\lambda = 1$ for SpikePropAd. Also observe the surge jumps in figure 5.29. The surges in SpikePropAd are negligible compared to other methods.

We observe that SpikePropAd with $\lambda = 0.1, 0.01 & 0.001$ demonstrate comparable performance. $\lambda = 0.01$, however, shows slightly better performance on the average.
number of epochs to reach a tolerance value. For $\lambda = 1$, the adaptive learning rate is almost always zero, identical to the observations for XOR, Fisher’s Iris benchmark and Wisconsin Breast Cancer benchmark.

![Figure 5.30](image1)

Figure 5.30: Trace of learning rate of output layer neuron in a typical Statlog (Landsat Satellite) data classification using SpikePropAd with $\lambda = 0.001$ for first 1000 iterations.

![Figure 5.31](image2)

Figure 5.31: Trace of learning rate of a hidden layer neuron in a typical Statlog (Landsat Satellite) data classification using SpikePropAd with $\lambda = 0.001$ for first 1000 iterations.

Figure 5.30 shows the trace of learning rate for an output neuron and figure 5.31 shows the trace of learning rate for a hidden neuron for first 1000 iterations using SpikePropAd during Statlog (Landsat Satellite) data classification learning process. The output layer neuron learning rate again assumes higher values compared to hidden layer neuron learning rates.

For a sense of learning comparison between SNN learning methods and standard backpropagation on MLPs, the results for Statlog (Landsat Satellite) data classification learning have been summarized in Table 5.1.

### 5.4 Choice of SpikePropAd Learning Parameter

The parameter $\lambda$ in SpikePropAd is crucial in evaluating the adaptive learning rate. A larger $\lambda$ means that the slope of membrane potential, $u_j'(t_j)$, must be very large in order to yield nonzero learning rate. Although this setting is theoretically good, it is impractical because we cannot expect $u_j'(t_j)$ to be very large every
Table 5.1: Comparison of performance of SNN with standard neural network methods [2]

<table>
<thead>
<tr>
<th>Method</th>
<th>Iterations</th>
<th>Accuracy</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Training</td>
<td>Testing</td>
<td></td>
</tr>
<tr>
<td><strong>Fisher’s Iris Classification</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpikeProp, ( \eta = 0.0075 )</td>
<td>1000</td>
<td>97.4% ± 0.1</td>
<td>96.1% ± 0.1</td>
<td></td>
</tr>
<tr>
<td>MATLAB BP</td>
<td>( 2.6 \times 10^6 )</td>
<td>98.2% ± 0.9</td>
<td>95.9% ± 2.0</td>
<td></td>
</tr>
<tr>
<td>MATLAB LM</td>
<td>3750</td>
<td>99.0% ± 0.1</td>
<td>95.7% ± 0.1</td>
<td></td>
</tr>
<tr>
<td><strong>Wisconsin Breast Cancer Classification</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpikeProp, ( \eta = 0.0075 )</td>
<td>1500</td>
<td>97.6% ± 0.2</td>
<td>97.0% ± 0.6</td>
<td></td>
</tr>
<tr>
<td>MATLAB BP</td>
<td>( 9.2 \times 10^6 )</td>
<td>98.1% ± 0.4</td>
<td>96.3% ± 0.6</td>
<td></td>
</tr>
<tr>
<td>MATLAB LM</td>
<td>3500</td>
<td>97.7% ± 0.3</td>
<td>96.7% ± 0.6</td>
<td></td>
</tr>
<tr>
<td><strong>Statlog (Landsat Satellite) Classification</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpikeProp, ( \eta = 0.0075 )</td>
<td>60,000</td>
<td>87.0% ± 0.5</td>
<td>85.3% ± 0.3</td>
<td></td>
</tr>
<tr>
<td>MATLAB BP</td>
<td>110,078</td>
<td>83.6% ± 1.3</td>
<td>82.0% ± 1.5</td>
<td></td>
</tr>
<tr>
<td>MATLAB LM</td>
<td>1,108,750</td>
<td>91.2% ± 0.5</td>
<td>88.0% ± 0.1</td>
<td></td>
</tr>
</tbody>
</table>

time. In our simulations, for \( \lambda \) in the order of 1 or greater we invariably found zero learning rate as a result the learning was not effective. This was observed for all three benchmarks discussed before. As a result, the convergence rate for SpikePropAd when \( \lambda = 1 \) is always zero for all the simulation benchmarks. We have also seen in [39], which employs a special case of SpikePropAd learning rate for \( \lambda = 0 \), produces similar results to the ones discussed in previous sections for SpikePropAd, although the bound condition in (5.14) is not often true for \( \lambda = 0 \) thus theoretically loose.

The learning results for different range of \( \lambda \) for SpikePropAd learning is summarized in Table 5.2. We can clearly see that \( \lambda \) in the range of 0.1 or less work well. It is also worth noting that the results for the two values of \( \lambda \) is not significantly different. For XOR and Fisher’s Iris classification, smaller \( \lambda \) yielded slightly better result whereas for Wisconsin Breast Cancer classification, larger value had slightly better result. This indicates that \( \lambda \) in the range of 0.1 or less generally produces good result for SpikePropAd. We should, however, not make \( \lambda \) very small in theory, although it may work well in practice.
### Table 5.2: Variation of $\lambda$ in SpikePropAd.

<table>
<thead>
<tr>
<th>SpikePropAd</th>
<th>$\text{Tol = 0.01}$</th>
<th>$\text{Tol = 0.0001}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>Conv</td>
<td>Epoch</td>
</tr>
<tr>
<td>XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.997</td>
<td>148.72</td>
</tr>
<tr>
<td>0.01</td>
<td>0.996</td>
<td>74.595</td>
</tr>
<tr>
<td>0.001</td>
<td>0.999</td>
<td>73.163</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>77.726</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fisher</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>99.86</td>
</tr>
<tr>
<td>0.01</td>
<td>1</td>
<td>74.82</td>
</tr>
<tr>
<td>0.001</td>
<td>1</td>
<td>86.61</td>
</tr>
<tr>
<td>0</td>
<td>0.97</td>
<td>86.65</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wisconsin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>182.87</td>
</tr>
<tr>
<td>0.01</td>
<td>1</td>
<td>229.59</td>
</tr>
<tr>
<td>0.001</td>
<td>1</td>
<td>228.11</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>231.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statlog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>18.8</td>
</tr>
<tr>
<td>0.01</td>
<td>1</td>
<td>19.3</td>
</tr>
<tr>
<td>0.001</td>
<td>1</td>
<td>19.4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>19.7</td>
</tr>
</tbody>
</table>
Chapter 6

SpikeProp with Adaptive Delay Learning

In Spiking Neural Network, delays are an important aspect of knowledge representation. It has been observed in the works of Wolfgang Maass [1, 20, 27] that delay parameters of SNN contributes to the computational capacity of SNNs. In his works, delay adjustment is crucial to Coincidence Detector (CD) and Element Distinctness (ED) modules (cf. Table 1.1) with the use of Type A and Type B neurons (cf. Chapter 2.2.5). Further, if we are able to learn delay parameter, we can get rid of redundant delayed synaptic connections that is used in SpikeProp and SpikePropAd. This will reduce the network complexity and yields efficient SNNs.

There have been couple of efforts to learn delay parameter in an SNN. In [85], delay learning has been proposed for Reservoir Computing. SpikeProp has also been extended for delay learning as well (cf. Chapter 3.4.1). In this chapter, we will focus on the latter. Similar to vanilla SpikeProp, the delay learning extension of SpikeProp also faces stability issues in the form of surges and hair trigger situation. In this Chapter, we will follow the idea of weight convergence analysis to tackle the problem of surges and hair trigger situation, similar to Chapter 5, and speed up the learning process. We will propose an adaptive learning rate for delay learning in this Chapter. Along with it, the adaptive learning rate for weight learning in Chapter 5 constitutes the final algorithm.

In the following section, we will present the adaptive delay learning extension to SpikeProp – SpikePropAdDel [41]. We will follow it by delay convergence analysis
6.1 The SpikePropAdDel Algorithm

In this section, we will present the SpikePropAdDel algorithm for learning delays in multilayer feedforward SNN as an extension to SpikePropDel algorithm. The adaptive learning rate for learning weights is the same as defined in (5.1), i.e.

$$\eta^\text{opt}_j(p) = \begin{cases} \tanh \left( \frac{u_j'(t_j)(u_j'(t_j) - \lambda_w)}{\|y_j\|^2} \right) & \text{if } u_j'(t_j) > \lambda_w \\ 0 & \text{otherwise} \end{cases}$$ \hspace{1cm} (6.1)

where \( \lambda_w > 0 \) is a positive bound on minimum slope of membrane potential to allow for weight update. The adaptive learning rate for learning delay is defined for each individual neuron and varies depending on the current state of the neuron similar to SpikePropAd. It is defined as follows:

$$\alpha^\text{opt}_j(p) = \begin{cases} \tanh \left( \frac{u_j'(t_j)(u_j'(t_j) - \lambda_d)}{\|\text{diag}(w_j)\dot{y}_j\|^2} \right) & \text{if } u_j'(t_j) > \lambda_d \\ 0 & \text{otherwise} \end{cases}$$ \hspace{1cm} (6.2)

where \( \lambda_d > 0 \) is a positive bound on minimum slope of membrane potential to allow for delay update. The SpikePropAdDel weight update rule at iteration \( p \) follows the same form as (5.3) i.e.

$$w_{jf}(p + 1) = w_{jf}(p) - \eta^\text{opt}_j(p) \delta_j y_{jf}. \hspace{1cm} (6.3)$$

and the SpikePropAdDel delay update rule at iteration \( p \) is (cf. Chapter 3.4.1)

$$d_{jf}(p + 1) = d_{jf}(p) + \alpha^\text{opt}_j(p) \delta_j \text{diag}(w_{jf})\dot{y}_{jf} \hspace{1cm} (6.4)$$

Note that the quadratic term of \( u_j'(t_j) \) in numerator of \( \alpha^\text{opt}_j(p) \) suppresses the \( u_j'(t_j) \) term in the denominator of \( \delta_j \). Further, when \( u_j'(t_j) \leq \lambda_w, \lambda_d \) the learning rate is 0. As a result, the hair trigger problem is not an issue in SpikePropAdDel as well.

The algorithm for SpikePropAdDel learning is presented in Algorithm 5.
Algorithm 5 SpikePropAdDel Learning

1: set up network connections
2: initialize weights
3: repeat
4: \[ \text{MSE} \leftarrow 0 \]
5: for all training sample do \( \triangleright \) assume \( m \) samples
6: for all \( N_i \in I \) do
7: \( t_i \leftarrow \text{input}_i \)
8: end for
9: for all \( N_j \in \mathcal{H}, \mathcal{O} \) do
10: \( t_j \leftarrow \text{CALC_TFIRE}(N_j) \) \( \triangleright \) algorithm 1
11: end for
12: for all \( N_j \in \mathcal{O}, \mathcal{H} \) do
13: calculate \( \delta_j \) \( \triangleright \) equations (3.29) & (3.31)
14: calculate \( \eta_j^{\text{opt}} \) \( \triangleright \) equation (6.1)
15: calculate \( \alpha_j^{\text{opt}} \) \( \triangleright \) equation (6.2)
16: for \( k \leftarrow 1 : |\mathcal{K}| \) \& \( \forall \ i \in \Gamma_j \) do
17: \( w_{ji}^{(k)} \leftarrow w_{ji}^{(k)} - \eta_j^{\text{opt}} \delta_j y_{ji}^{(k)}(t_j) \)
18: \( d_{ji}^{(k)} \leftarrow d_{ji}^{(k)} + \alpha_j^{\text{opt}} \delta_j w_{ji}^{(k)} \frac{\partial y_{ji}^{(k)}(t_j)}{\partial t} \)
19: end for
20: if \( N_j \in \mathcal{O} \) then
21: \( \text{MSE} \leftarrow \text{MSE} + E/m \) \( \triangleright \) equation (3.26)
22: end if
23: end for
24: end for
25: until \( \text{MSE} < \text{TOL} \) OR \( \text{MAX_EPOCH} \) is exceeded

Similar to SpikePropAd, from the point of view of implementation, there is not much additional computational overhead in using SpikePropAdDel. The additional step compared to SpikePropDel learning is the computation of adaptive learning rate term, whose parameters \( u'_{ij}(t_j) \) and \( \frac{\partial y_{ji}^{(k)}(t_j)}{\partial t} \) are precomputed for SpikePropDel as well.

The adaptive weight update formulation is weight convergent from Chapter 5.2. Next we will show that learning using SpikePropAdDel is convergent in the sense of delay parameters.
6.2 Delay Convergence Analysis of SpikePropAdDel

Consider the ideal delay vectors for hidden layer and output layer neurons: \( \bar{d}_{hl} \) and \( \bar{d}_{oH} \). Then

\[
\bar{y}_{ji} = \left[ \cdots, \varepsilon(t_j - t_i - d^{(k)}_{ji}), \cdots \right]^T \in \mathbb{R}^{\hat{I}|K|} \tag{6.5}
\]

The ideal membrane potential for hidden layer neuron and output layer neuron is

\[
\bar{u}_{ji}(t) = w^T_{ji} \bar{y}_{ji}(t) \tag{6.6}
\]

The approach for delay convergence analysis of SpikePropAdDel uses the Lyapunov function of similar form as for SpikePropAd. The Lyapunov function is defined as the difference between norm of actual and ideal delay vectors in two adjacent iterations. If we show Lyapunov stability of the delay based Lyapunov function, the current weight matrices or vectors always move towards ideal delay, or at least maintain the same distance from ideal delays, at each iteration. We will consider individual delay learning rate for each neuron \( \alpha_j(p) \) and relate it to the optimal learning rate \( \alpha_{j, opt}^d(p) \) later.

Denote the difference between actual and ideal delay matrix as \( \tilde{d}_{ji}(p) = d_{ji}(p) - \bar{d}_{ji} \).

Now, define the Lyapunov function

\[
V^d_j = \left\| \tilde{d}_{ji}(p) \right\|^2 \tag{6.7}
\]

\[
\Delta V^d_j = \left\| \tilde{d}_{ji}(p + 1) \right\|^2 - \left\| \tilde{d}_{ji}(p) \right\|^2 \tag{6.8}
\]

**Theorem 6.1.** The delay vector \( d_{ji} \) is convergent in the sense of Lyapunov function \( V^d_j \) such that \( \Delta V^d_j \leq 0 \) if for a fixed parameter \( \lambda_d \geq 0 \), the SpikePropDel delay learning rate of neuron \( N_j \) at iteration \( p \) satisfies

\[
0 \leq \alpha_j(p) \leq \begin{cases} 
\frac{u_j'(t_j)(u_j'(t_j) - \lambda)}{\|\text{diag}(w_{ji})\bar{y}_{ji}\|^2} & \text{if } u_j'(t_j) > \lambda_d \\
0 & \text{otherwise}
\end{cases} \tag{6.9}
\]
Proof. Consider

\[
\Delta V_j^d = \left\| \tilde{d}_{ji}(p + 1) \right\|^2 - \left\| \tilde{d}_{ji}(p) \right\|^2 \\
= \left\| d_{ji}(p + 1) - \tilde{d}_{ji} \right\|^2 - \left\| d_{ji}(p) - \tilde{d}_{ji} \right\|^2 \\
= (d_{ji}(p + 1) - \tilde{d}_{ji})^T (d_{ji}(p + 1) - \tilde{d}_{ji}) \\
\quad - (d_{ji}(p) - \tilde{d}_{ji})^T (d_{ji}(p) - \tilde{d}_{ji}) \\
= d_{ji}(p + 1)^T d_{ji}(p + 1) + \tilde{d}_{ji}^T \tilde{d}_{ji} - 2 \tilde{d}_{ji}^T d_{ji}(p + 1) \\
\quad - d_{ji}(p)^T \tilde{d}_{ji}(p) - \tilde{d}_{ji}^T \tilde{d}_{ji} + 2 \tilde{d}_{ji}^T d_{ji}(p) \\
= d_{ji}(p + 1)^T d_{ji}(p + 1) - 2 \tilde{d}_{ji}^T d_{ji}(p + 1) \\
\quad - d_{ji}(p)^T \tilde{d}_{ji}(p) + 2 \tilde{d}_{ji}^T d_{ji}(p)
\]

by using (6.4), we get

\[
\Delta V_j^d = \alpha_j(p)^2 \delta_j^2 \left\| \text{diag}(w_{ji}) \hat{y}_{ji} \right\|^2 + 2 \alpha_j(p) \delta_j \tilde{d}_{ji}(p)^T \text{diag}(w_{ji}) \hat{y}_{ji}
\]

Now consider the Taylor polynomial expansion

\[
\varepsilon(t_j - t_i - \tilde{d}_{ji}^{(k)}) = \varepsilon(t_j - t_i - d_{ji}^{(k)} + \tilde{d}_{ji}^{(k)}) \\
= \varepsilon(t_j - t_i - d_{ji}^{(k)}) + \tilde{d}_{ji}^{(k)} \varepsilon'(t_j - t_i - d_{ji}^{(k)}) + \frac{1}{2} \left( \tilde{d}_{ji}^{(k)} \right)^2 \varepsilon''(\zeta_j)
\]

where \( \zeta_j = t_j - t_i - (1-c) d_{ji}^{(k)} - c \tilde{d}_{ji}^{(k)} \), \( c \in (0,1) \). Now, arranging (6.11) in vector form, we get

\[
\tilde{y}_{ji} = y_{ji} + \text{diag}(\tilde{d}_{ji}) \hat{y}_{ji} + \frac{1}{2} \text{diag}(\tilde{d}_{ji})^2 \varepsilon''(\zeta)
\]

where \( \zeta = t_j - t_i - (1-c) d_{ji} - c \tilde{d}_{ji} \), \( c \in (0,1) \) and \( \varepsilon''(\zeta) = [\cdots, \varepsilon''(\zeta_j), \cdots]^T \).

Multiplying both sides by \( w_{ji}^T \) and by using (3.11) and (5.4), we get

\[
\bar{u}_j(t_j) = u_j(t_j) + w_{ji}^T \text{diag}(\tilde{d}_{ji}) \hat{y}_{ji} + \frac{1}{2} w_{ji}^T \text{diag}(\tilde{d}_{ji})^2 \varepsilon''(\zeta) \\
\quad \Rightarrow \tilde{d}_{ji}(p)^T \text{diag}(w_{ji}) \hat{y}_{ji} = \bar{u}_j(t_j) - u_j(t_j) - \frac{1}{2} w_{ji}^T \text{diag}(\tilde{d}_{ji}(p))^2 \varepsilon''(\zeta)
\]
Then (6.10) becomes
\[
\Delta V^d_j = \alpha_j(p)^2 \delta_j^2 \left\| \text{diag}(w_{jf}) \dot{y}_{jf} \right\|^2 - 2\alpha_j(p) \delta_j (u_j(t_j) - \tilde{u}_j(t_j)) \\
- \alpha_j(p) \delta_j w_{jf}^T \text{diag}(\tilde{d}_{jf}(p))^2 \varepsilon''(\zeta)
\]  
(6.14)

By definition, we have
\[
e_o = \tilde{t}_o - t_o = \tilde{t}_o - t_o
\]  
(6.15)
for output layer neuron. Similarly, the error for hidden layer neuron can be written as
\[
e_h = \tilde{t}_h - t_o = -\frac{\partial E_h}{\partial t_h} \approx -\frac{\partial E}{\partial t_h} = \frac{\partial E}{\partial t_o} = e_O
\]  
(6.16)
where \( E_h = \frac{1}{2} e_h^2 \) and the backpropagation estimate of the local error of hidden layer neuron is \( e_h \approx -\frac{\partial E}{\partial t_h} \). Note that, the membrane potential must be equal to the threshold value when the neuron fires i.e. \( u_j(t_j) = \tilde{u}_j(t_j) = \vartheta \). Then
\[
u_j(t_j) - \tilde{u}_j(t_j) = \tilde{u}_j(\tilde{t}_j) - \tilde{u}_j(t_j) = \tilde{u}_j'(\xi)e_j
\]  
(6.17)
where \( \min(t_j, \tilde{t}_j) < \xi < \max(t_j, \tilde{t}_j) \). From (3.29), (3.31) and (5.12), (6.14) becomes
\[
\Delta V^d_j = \alpha_j(p)^2 \frac{e_j^2}{u_j'(t_j)^2} \left\| \text{diag}(w_{jf}) \dot{y}_{jf} \right\|^2 - 2\alpha_j(p) e_j^2 \frac{\tilde{u}_j'(\xi)}{u_j'(t_j)} \\
- \alpha_j(p) \delta_j w_{jf}^T \text{diag}(\tilde{d}_{jf}(p))^2 \varepsilon''(\zeta)
\]
\[
= \frac{\alpha_j(p) e_j^2}{u_j'(t_j)^2} \left[ \alpha_j(p) \left\| \text{diag}(w_{jf}) \dot{y}_{jf} \right\|^2 - 2 \tilde{u}_j'(\xi) u_j'(t_j) \\
- \frac{\tilde{u}_j'(t_j)}{e_j} w_{jf}^T \text{diag}(\tilde{d}_{jf}(p))^2 \varepsilon''(\zeta) \right]
\]  
(6.18)
Note that as \( e_j \to 0 \), \( \tilde{d}_{ji}^{(k)} \to 0 \). The factor \( \frac{1}{e_j} \) in the third term is problematic. However,
\[
\lim_{e_j \to 0} \left( \frac{\tilde{d}_{ji}^{(k)}}{e_j} \right)^2 = \lim_{e_j \to 0} 2\tilde{d}_{ji}^{(k)} \frac{dd_{ji}^{(k)}}{de_j} \bigg|_{e_j=0} = 0
\]  
(6.19)
This means the term \( \frac{1}{e_j} w_{jf}^T \text{diag}(\tilde{d}_{jf}(p))^2 \varepsilon''(\zeta) \) is finitely bounded, provided \( \tilde{d}_{ji}^{(k)} \) is
finite $\forall i, k$. If we define a positive bound $\lambda_d$ such that

$$\left| u'_j(t_j) - \bar{u}'(\xi) - \frac{1}{2e_j} \mathbf{w}^T_{jj} \text{diag}(\tilde{d}_{jj}(p)) \varepsilon''(\xi) \right| \leq \lambda_d$$

Then from (6.18)

$$\Delta V_j^d \leq \frac{\alpha_j(p) e_j^2}{u'_j(t_j)^2} \left[ \alpha_j(p) \left\| \text{diag}(\mathbf{w}_{jj}) \dot{y}_{jj} \right\|^2 - 2 u'_j(t_j)^2 + 2 \lambda_d u'_j(t_j) \right] \leq 0$$

$$\implies \alpha_j(p) \leq \begin{cases} 2 \frac{u'_j(t_j)(u'_j(t_j) - \lambda_d)}{\left\| \text{diag}(\mathbf{w}_{jj}) \dot{y}_{jj} \right\|^2} & \text{if } u'_j(t_j) > \lambda_d \\ 0 & \text{otherwise} \end{cases} \square$$

**Corollary 6.1.1.** The SpikePropAdDel adaptive delay learning rate (6.2) satisfies Theorem 6.1.

**Proof.** Note that $0 \leq \tanh x \leq x$, $\forall x \geq 0$ and the result follows from (6.9). \square

### 6.3 Performance Comparison of SpikePropAdDel

In this section, we will compare the performance of SpikeProp with delay adaptation (SpikePropDel) the adaptive delay learning modification to SpikeProp (SpikePropAdDel) proposed in this Chapter. We will compare the two methods based on results of simulations on four benchmark datasets: XOR, Fisher’s Iris data, Wisconsin Breast Cancer data [110] and Statlog (Landsat Satellite) data [110]. As discussed in Chapter 3.7, we will compare the simulation results based on their convergence rate (cf. Chapter 3.7.2), average number of epochs required to reach a tolerance value (cf. Chapter 3.7.3), training and testing accuracy (cf. Chapter 3.7.4) and average learning curve (cf. Chapter 3.7.5).

The SNN architecture used is a three layer architecture as described in Chapter 3.1. The weights are initialized randomly based on normal distribution using the method described in Chapter 3.3.2. The delays are initialized using uniform random distribution. We will now present the simulation results on each of the datasets and discuss the performance results.
6.3.1 XOR problem

The simulation parameters for XOR classification are listed below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Architecture</td>
<td>$</td>
</tr>
<tr>
<td>SRM time constant</td>
<td>$\tau_s = 7$</td>
</tr>
<tr>
<td>Activation Threshold</td>
<td>$\vartheta = 1$</td>
</tr>
<tr>
<td>Simulation Interval</td>
<td>$0 : 0.01 : 25$ units</td>
</tr>
<tr>
<td>Weight Initialization</td>
<td>$t_{\text{min}} = 0.1$ units $&amp; t_{\text{max}} = 10$ units</td>
</tr>
<tr>
<td>Dead Neuron Boosting</td>
<td>boost by 0.05</td>
</tr>
</tbody>
</table>

The input output spike encoding is same as in Chapter 5.3.1.

Simulations were performed on the XOR problem using SpikePropDel and SpikePropAdDel from 1000 different initial weights. A maximum cap of 5000 epochs was used to classify the trial as non-converging. Figure 6.1 shows the plot of convergence rate versus the mean square error tolerance achieved. SpikePropAdDel shows better convergence rate performance than SpikePropDel.

![Figure 6.1: Convergence Rate plot for learning XOR pattern using SpikePropDel and SpikePropAdDel.](image)

The average number of epochs required to converge to different mean square error tolerance is plotted in figure 6.2. The lower limit of error indicates the minimum number of epochs required and the upper bar indicates the standard deviation of the number of epochs required. It is clear from figure 6.2 that SpikePropAdDel achieves required MSE tolerance error values faster than SpikePropDel in general.

In figure 6.3, average learning MSE error attained by SpikePropDel and SpikePropAdDel during 1000 different trials is plotted. Clearly SpikePropAdDel shows...
6.3. Performance Comparison of SpikePropAdDel

Figure 6.2: Average number of epochs required to learn XOR pattern using SpikePropDel and SpikePropAdDel.

Figure 6.3: Average learning curves for XOR problem using SpikePropDel and SpikePropAdDel.

A better result. Note the numerous jumps in average learning curve for SpikePropDel. These jumps are due to big surges occurring in individual training process. Figure 6.4 shows a typical learning curve for SpikePropDel and SpikePropAdDel. SpikePropAdDel clearly shows minimal surge during training.

A comparison of performance of different methods during XOR learning including SpikePropAd for different number of delayed synaptic contacts is listed in table 6.1. We can see that SpikePropAdDel, $\lambda_w = 0.001$, $\lambda_d = 100$, $|K| = 16$ has better performance in terms of number of epochs required. It does not have the best convergence rate, however, it is close.
| Method                  | $\eta_w$ | $\eta_d$ | $|K|$ | Epoch | Time (s) |
|------------------------|----------|----------|------|-------|----------|
| SpikePropDel, $\lambda$ | 0.001    | 0.001    | 16   | 0.1   | 0.919    |
|                        | 0.001    | 0.001    | 4    | 0.001 | 0.574    |
| SpikePropAdDel, $\lambda$ | 0.001 | 100      | 16   | 0.1   | 0.955    |
|                        | 0.001    | 10       | 4    | 0.001 | 0.663    |
| SpikePropAd, $\lambda$ | 0.1      | 0.1      | 16   | 0.1   | 0.999    |
|                        | 0.001    | 100      | 4    | 0.001 | 0.999    |
| RProp                   | 1.3      | 0.5      | 16   | 0.1   | 0.983    |

Table 6.1: SpikePropAdDel Simulation Results: XOR problem
6.3. Performance Comparison of SpikePropAdDel

Figure 6.4: Typical learning curves for XOR problem using SpikePropDel and SpikePropAdDel.

6.3.2 Fisher’s Iris classification

The simulation parameters for Fisher’s Iris classification are listed below.

- **Network Architecture**: $|I| = 16$, $|H| = 7$ & $|O| = 1$
- **SRM time constant**: $\tau_s = 7$
- **Activation Threshold**: $\vartheta = 1$
- **Simulation Interval**: $0 : 0.01 : 25$ units
- **Weight Initialization**: $t_{\text{min}} = 0.1$ units & $t_{\text{max}} = 10$ units
- **Dead Neuron Boosting**: boost by 0.05

The input and output spikes are encoded similarly as in Chapter 5.3.2.

100 different initial weights were used for different trials for SpikePropDel and SpikePropAdDel. A maximum of 1500 epochs was considered to label converged and non-converged trials. Convergence rate versus mean square error tolerance is plotted in figure 6.5. SpikePropAdDel shows consistent convergence performance.

Figure 6.6 shows the plot of mean number of epochs required to reach different MSE tolerance values using SpikePropDel and SpikePropAdDel. Based on the number of epochs required, SpikePropAdDel shows better performance.

Figure 6.7 and 6.8 show the plot of training accuracy and testing accuracy achieved for different MSE tolerance values. The upper error bar position indicates the maximum accuracy achieved and the length of lower bar indicates the standard deviation in accuracy for both training and testing accuracy. SpikePropAdDel shows better result for training accuracy and testing accuracy as well.
6.3. Performance Comparison of SpikePropAdDel

Figure 6.5: Convergence Rate plot for learning Fisher’s Iris data using SpikePropDel and SpikePropAdDel.

Figure 6.6: Average number of epochs required to learn Fisher’s Iris data using SpikePropDel and SpikePropAdDel.

The plot of average learning curve over 100 different trials of SpikePropDel and SpikePropAdDel for Fisher’s Iris learning is shown in figure 6.9. We can see that initially SpikePropDel has better performance, however, big jumps due to surge disturbs its performance. SpikePropAdDel, on the other hand, shows consistent learning performance. A typical learning curve for Fisher’s Iris learning using both methods is plotted in figure 6.10. Note the jumps due to surge in learning using SpikePropDel. In comparison, surges are non-existent in learning using SpikePropAdDel.

Simulation results for different methods and different number of synaptic connections for SpikePropAdDel is summarized in table 6.2. It can be seen that for Fisher’s Iris classification, SpikePropAdDel produces good results in terms of
Figure 6.7: Training accuracy in Fisher’s Iris classification using SpikePropDel and SpikePropAdDel.

Figure 6.8: Testing accuracy in Fisher’s Iris classification using SpikePropDel and SpikePropAdDel.

training and testing accuracy. It does not have the best convergence rate. On the other hand the number of epoch required is competitive.

6.3.3 Wisconsin Breast Cancer classification

The simulation parameters for Wisconsin Breast Cancer classification problem are listed below.
**Table 6.2: SpikePropAdDel Simulation Results: Fisher’s Iris classification**

| Method               | Epoch | Training Accuracy | Testing Accuracy | Convergence
|----------------------|-------|-------------------|------------------|--------------
| RProp, $\eta^+$ = 1.3, $\eta^-$ = 0.5 | 196 | 0.854 | 0.0 | 100%
| SpikePropAdDel, $\eta^+$ = 0.01, $\eta^-$ = 10 | 0.0 | 866 | 0.0 | 96%
| SpikePropAdDel, $\eta^+$ = 0.01, $\eta^-$ = 0.1 | 0.0 | 966 | 0.0 | 96%
| SpikePropAdDel, $\eta^+$ = 0.01, $\eta^-$ = 0.01 | 0.0 | 966 | 0.0 | 96%
| SpikePropAdDel, $\eta^+$ = 0.01, $\eta^-$ = 0.0001 | 0.0 | 966 | 0.0 | 96%

<table>
<thead>
<tr>
<th>Mean</th>
<th>SD</th>
<th>Max</th>
<th>SD</th>
<th>Mean</th>
<th>SD</th>
<th>Max</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epoch</td>
<td>Convergence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training</td>
<td>Testing</td>
<td></td>
<td></td>
<td>Training</td>
<td>Testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td>Accuracy</td>
<td></td>
<td></td>
<td>Accuracy</td>
<td>Accuracy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The table above shows the simulation results for Fisher’s Iris classification using various methods. The table includes the epoch number, the mean and standard deviation (SD) of the training and testing accuracies, along with the maximum accuracy achieved. The Convergence column indicates the point at which the model converged.
6.3. Performance Comparison of SpikePropAdDel

Figure 6.9: Average learning curves during Fisher’s Iris classification using SpikePropDel and SpikePropAdDel.

Figure 6.10: Typical learning curves during Fisher’s Iris classification using SpikePropDel and SpikePropAdDel.

Network Architecture  $|I| = 64$, $|H| = 15$ & $|O| = 2$
SRM time constant  $\tau_s = 7$
Activation Threshold  $\vartheta = 1$
Simulation Interval  $0 : 0.01 : 25$ units
Weight Initialization  $t_{\text{min}} = 0.1$ units & $t_{\text{max}} = 10$ units
Dead Neuron Boosting  boost by 0.05

The input and output spikes are encoded similarly as in Chapter 5.3.2.

100 different trials from different random weights were simulated on the encoded spike data using SpikePropDel and SpikePropAdDel for Wisconsin classification problem. Maximum cap of 1000 epochs was used to label converging and non-converging simulation trials. The convergence rate versus MSE error is plotted.
in figure 6.11 for both methods. It is clear that SpikePropAdDel demonstrates superior convergence rate.

Figure 6.11: Convergence Rate plot for learning Wisconsin Breast Cancer data using SpikePropDel and SpikePropAdDel.

Figure 6.12: Average number of epochs required to learn Wisconsin Breast Cancer data using SpikePropDel and SpikePropAdDel.

Figure 6.12 shows the comparison of number of epochs required to achieve different MSE error values for both methods. The lower bar indicates the minimum number of epochs required and the length of upper bar shows the standard deviation in the average number of epochs required. The number of epochs required seem to be comparable for both SpikePropDel and SpikePropAdDel for the trials that converged in Wisconsin classification.

The training accuracy and testing accuracy corresponding to different MSE error during Wisconsin classification learning are plotted in figure 6.13 and 6.14 respectively for SpikePropDel and SpikePropAdDel. The upper bar in the plot indicates
6.3. Performance Comparison of SpikePropAdDel

![Figure 6.13](image1.png)

Figure 6.13: Training accuracy in Wisconsin Breast Cancer classification using SpikePropDel and SpikePropAdDel.

![Figure 6.14](image2.png)

Figure 6.14: Testing accuracy in Wisconsin Breast Cancer classification using SpikePropDel and SpikePropAdDel.

the maximum accuracy achieved and the length of lower bar indicates the standard deviation of accuracy. The plots show that both methods achieve similar accuracy results.

The average learning curve for Wisconsin Breast Cancer learning using SpikePropDel and SpikePropAdDel is plotted in figure 6.15. It is clear that SpikePropAdDel shows consistently better learning performance than SpikePropDel. Figure 6.16 shows the typical learning curve during Wisconsin Breast Cancer learning. Note the surges occurring in learning using SpikePropDel. In comparison, there is very little surge during SpikePropAdDel.

Simulation results for different methods and different number of synaptic connec-
6.3. Performance Comparison of SpikePropAdDel

Figure 6.15: Average learning curves during Wisconsin Breast Cancer classification using SpikePropDel and SpikePropAdDel.

Figure 6.16: Typical learning curves during Wisconsin Breast Cancer classification using SpikePropDel and SpikePropAdDel.

tions for SpikePropAdDel is summarized in table 6.3. We can see that SpikePropAdDel shows very good results among the lot.

6.3.4 Statlog (Landsat Satellite) data classification

The simulation parameters for Wisconsin Breast Cancer classification problem are listed below.
### Table 6.3: SpikePropAdDel Simulation Results: Wisconsin Breast cancer classification

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mean</td>
<td>SD</td>
<td>max</td>
</tr>
<tr>
<td>SpikePropDel, $\eta = 0.01$, $\eta_d = 0.1$, $</td>
<td>K</td>
<td>= 16$</td>
<td>0.97</td>
<td>105.58</td>
</tr>
<tr>
<td>SpikePropDel, $\eta = 0.01$, $\eta_d = 0.01$, $</td>
<td>K</td>
<td>= 4$</td>
<td>0.47</td>
<td>687.69</td>
</tr>
<tr>
<td>SpikePropAdDel, $\lambda = 0.1$, $\lambda_d = 0.01$, $</td>
<td>K</td>
<td>= 16$</td>
<td>1.00</td>
<td>73.44</td>
</tr>
<tr>
<td>SpikePropAdDel, $\lambda = 0.1$, $\lambda_d = 0.001$, $</td>
<td>K</td>
<td>= 4$</td>
<td>0.86</td>
<td>260.8</td>
</tr>
<tr>
<td>SpikePropAd, $\lambda = 0.1$</td>
<td>1.00</td>
<td>182.87</td>
<td>70.67</td>
<td>1</td>
</tr>
<tr>
<td>RProp, $\eta^+ = 1.3$, $\eta^- = 0.5$</td>
<td>1.00</td>
<td>71.38</td>
<td>26.48</td>
<td>1</td>
</tr>
<tr>
<td>SpikePropDel, $\eta = 0.01$, $\eta_d = 0.1$, $</td>
<td>K</td>
<td>= 16$</td>
<td>0.93</td>
<td>220.42</td>
</tr>
<tr>
<td>SpikePropDel, $\eta = 0.01$, $\eta_d = 0.01$, $</td>
<td>K</td>
<td>= 4$</td>
<td>0.18</td>
<td>921.29</td>
</tr>
<tr>
<td>SpikePropAdDel, $\lambda = 0.1$, $\lambda_d = 0.01$, $</td>
<td>K</td>
<td>= 16$</td>
<td>1.00</td>
<td>168.42</td>
</tr>
<tr>
<td>SpikePropAdDel, $\lambda = 0.1$, $\lambda_d = 0.001$, $</td>
<td>K</td>
<td>= 4$</td>
<td>0.27</td>
<td>807.42</td>
</tr>
<tr>
<td>SpikePropAd, $\lambda = 0.1$</td>
<td>0.72</td>
<td>643.4</td>
<td>254.29</td>
<td>1</td>
</tr>
<tr>
<td>RProp, $\eta^+ = 1.3$, $\eta^- = 0.5$</td>
<td>0.55</td>
<td>656</td>
<td>204.03</td>
<td>1</td>
</tr>
</tbody>
</table>
6.3. Performance Comparison of SpikePropAdDel

Network Architecture $|\mathcal{I}| = 101$, $|\mathcal{H}| = 25 \& |\mathcal{O}| = 6$
SRM time constant $\tau_s = 7$
Activation Threshold $\vartheta = 1$
Simulation Interval $0 : 0.01 : 25$ units
Weight Initialization $t_{\text{min}} = 0.1$ units $\& t_{\text{max}} = 10$ units
Dead Neuron Boosting boost by 0.05

The input and output spikes are encoded similarly as in Chapter 5.3.2.

10 different simulation trials were conducted using SpikePropDel and SpikePropAdDel starting from different random weight initializations for Landsat classification. A maximum cap of 1000 epochs was considered to label converging and non-converging trials. Figure 6.17 shows the comparison of convergence rate between SpikePropDel and SpikePropAdDel for different MSE tolerance values. It is clear that SpikePropAdDel has better convergence rate performance.

![Figure 6.17: Convergence Rate plot for learning Statlog (Landsat Satellite) data using SpikePropDel and SpikePropAdDel.](image)

Figure 6.18 shows the plot of average number of epochs required to reach different tolerance values. The lower bar indicates the minimum number of epochs required and the length of upper bar indicates the standard deviation of the number of epochs. SpikePropAdDel, in this case demonstrates significant speed-up in terms of learning epochs.

The training accuracy and testing accuracy achieved for different MSE tolerance values is depicted in figure 6.19 and 6.20 respectively. It can be seen that SpikePropAd has better accuracy performance as well.

Figure 6.21 shows the average learning curve during Statlog (Landsat Satellite)
6.3. Performance Comparison of SpikePropAdDel

Figure 6.18: Average number of epochs required to learn Statlog (Landsat Satellite) data using SpikePropDel and SpikePropAdDel.

Figure 6.19: Training accuracy in Statlog (LandSat Satellite) classification using SpikePropDel and SpikePropAdDel.

Data classification learning using SpikePropDel and SpikePropAdDel. We can see that SpikePropAdDel demonstrates better learning performance. Note the big jumps in average learning curve due to surges in different trials. A typical learning curve for Landsat classification using SpikePropDel and SpikePropAdDel is shown in figure 6.22. Surges are frequent for SpikePropDel, but virtually non-existent in SpikePropAdDel learning.

Simulation results for different methods and different number of synaptic connections for SpikePropAdDel is summarized in table 6.4. It is evident from the table that SpikePropAdDel shows very good performance in terms of convergence rate, epochs required, training accuracy and testing accuracy.
### Table 6.4: SpikePropAdDel Simulation Results: Landsat data classification

<table>
<thead>
<tr>
<th>Method</th>
<th>Conv. Epoch</th>
<th>Training Accuracy</th>
<th>Testing Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpikePropDel, ( \eta_w = 0.01 ), ( \eta_d = 0.1 ), (</td>
<td>K</td>
<td>= 16 )</td>
<td>0.8091 0.8276 0.833 0.841 0.842 0.844 0.847 0.849 0.852 0.854</td>
</tr>
<tr>
<td>SpikePropAdDel, ( \lambda_w = 0.1 ), ( \lambda_d = 0.01 ), (</td>
<td>K</td>
<td>= 16 )</td>
<td>0.8091 0.8276 0.833 0.841 0.842 0.844 0.847 0.849 0.852 0.854</td>
</tr>
<tr>
<td>SpikePropAd, ( \lambda = 0.1 )</td>
<td>0.8091 0.8276 0.833 0.841 0.842 0.844 0.847 0.849 0.852 0.854</td>
<td>0.782 0.794 0.805 0.815 0.828 0.838 0.845 0.854 0.861 0.870</td>
<td>0.767 0.776 0.781 0.786 0.791 0.795 0.800 0.804 0.807 0.811</td>
</tr>
<tr>
<td>RProp, ( \eta^+ = 1.3, \eta^- = 0.5 )</td>
<td>0.8091 0.8276 0.833 0.841 0.842 0.844 0.847 0.849 0.852 0.854</td>
<td>0.782 0.794 0.805 0.815 0.828 0.838 0.845 0.854 0.861 0.870</td>
<td>0.767 0.776 0.781 0.786 0.791 0.795 0.800 0.804 0.807 0.811</td>
</tr>
</tbody>
</table>

**Note:** The values are rounded for simplicity.
6.4 Choice of SpikePropAdDel Learning Parameter

SpikePropAdDel has two free parameters $\lambda_w$ and $\lambda_d$. These parameters behave similar to $\lambda$ for SpikePropAd. In fact $\lambda_w$ is same as $\lambda$. $\lambda_d$ also has similar properties as $\lambda_w$. The difference being the minimum slope of membrane potential, $u'_j(t_j)$, for delay update and weight update are different in each case. Similar to SpikePropAd, a larger $\lambda_d$ is theoretically good, it is impractical because we cannot expect $u'_j(t_j)$ to be very large every time. In our simulations, for $\lambda_d$ in the order of 10 or greater we invariably found zero learning rate as a result the learning result was found to be similar to the case in which just the weights were learned. This was observed
6.4. Choice of SpikePropAdDel Learning Parameter

Figure 6.22: Typical learning curves during Statlog (Landsat Satellite) classification using SpikePropDel and SpikePropAdDel.

for all three benchmarks discussed before. $\lambda_d = 0$ is also possible, although the bound condition in is not often true for $\lambda_d = 0$ thus theoretically loose. The results of variation in $\lambda_d$ are summarized in Table 6.5.

<table>
<thead>
<tr>
<th>$\lambda_w$</th>
<th>$\lambda_d$</th>
<th>Conv</th>
<th>Epoch</th>
<th>Conv</th>
<th>Epoch</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>100</td>
<td>0.938</td>
<td>117.57</td>
<td>0.924</td>
<td>152.95</td>
</tr>
<tr>
<td>0.001</td>
<td>10</td>
<td>0.849</td>
<td>116.49</td>
<td>0.834</td>
<td>135.43</td>
</tr>
<tr>
<td>0.001</td>
<td>1</td>
<td>0.849</td>
<td>116.49</td>
<td>0.834</td>
<td>135.43</td>
</tr>
<tr>
<td>0.001</td>
<td>0.1</td>
<td>0.874</td>
<td>70.15</td>
<td>0.861</td>
<td>88.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XOR</th>
<th>Tol = 0.1</th>
<th>Tol = 0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>10</td>
<td>0.86</td>
</tr>
<tr>
<td>0.01</td>
<td>0.1</td>
<td>0.85</td>
</tr>
<tr>
<td>0.01</td>
<td>0.01</td>
<td>0.87</td>
</tr>
<tr>
<td>0.01</td>
<td>0.001</td>
<td>0.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fisher</th>
<th>Tol = 0.1</th>
<th>Tol = 0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>0.1</td>
<td>0.01</td>
<td>1</td>
</tr>
<tr>
<td>0.1</td>
<td>0.001</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wisconsin</th>
<th>Tol = 6</th>
<th>Tol = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0.01</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>0.01</td>
<td>0.01</td>
<td>1</td>
</tr>
<tr>
<td>0.01</td>
<td>0.001</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.5: Variation of $\lambda_d$ in SpikePropAdDel.
Chapter 7

Robust Learning in SpikeProp

The adaptive learning rate modification on SpikeProp based on weight convergence analysis in Chapter 5 provided a nice way to choose a proper learning rate that judiciously chooses a suitable learning rate for a neuron depending on the current state of the neuron and yields better learning performance, which is backed up by the simulation results as well. The effect of weight convergence was that the learning was successful and even faster than the speedier variant of SpikeProp: RProp. Further, in the learning curves, surges were minimal and under control. However, one important issue still remains: robustness to disturbances during learning.

In learning systems, there are always some forms of disturbance in the system. The disturbance can be either external or internal. The external disturbance is present in the form of measurement error in learning samples i.e. the desired firing time $\hat{t}_o$ is mixed with the ideal firing time $\bar{t}_o$ and some form of external disturbance $\epsilon^{\text{ext}}_o$. The desired firing time is the target firing time in dataset input output tuple, whereas, the ideal firing time is the optimal/sub-optimal firing time that is attainable. It is reasonable to assume additive disturbance signal as indicated in (3.10). In real world data, this external disturbance cannot be avoided, although, it can be controlled to a minimum level by meticulous measurement of the data. It can however be avoided in case of synthetically generated data. Internal disturbance, on the other hand is due to modeling imperfection and can never be avoided during learning process. There is no internal disturbance in the system only if we have achieved an ideal parameters to a system. In such a situation, however, there is no point of learning.
In this Chapter, we will present how robustness to both external and internal disturbance can be achieved during SpikeProp learning. We will perform convergence analysis of SpikeProp in presence of external disturbance. We will formally define the error system of SNN learning process, prove stability results of error signals in $L_2$ space and extend the $L_2$ stability results into $L_\infty$ space. The extension to $L_\infty$ space is important because $L_2$ stability requires the disturbance to die out to zero which is not always feasible in practice. $L_\infty$ extension relaxes this condition to boundedness of disturbance throughout the learning process which makes much more practical sense. The stability results and convergence results mean that learning system is stable in presence of non-zero external disturbance throughout the learning process.

We will propose two different approaches to robust stability in this Chapter. First, we will consider the error system for an individual spiking neuron and proceed to stability results from there on. This will form the SpikePropR algorithm [42]. Second, we will consider the total error of the entire SNN system during learning process and show stability results for it. This will lead to the SpikePropRT algorithm [43]. We will compare the effectiveness of these two strategy at the end of this Chapter via simulation results of different benchmark problems.

### 7.1 The SpikePropR Algorithm

We will now present the details of SpikePropR algorithm in this section as a robust adaptive learning extension to SpikeProp for learning weights. The adaptive learning rate is defined for each individual neuron and varies depending upon the current state of the neuron akin to SpikePropAd. The robust adaptive learning rate is defined as

\[
\eta^R_j(p) = \begin{cases} 
\tanh\left(\frac{u_j(t_j)^2}{\|y_j\|^2}\right) & \text{if } (1 - \sigma_j)e_j^2 \geq \epsilon_m^2 \\
0 & \text{otherwise}
\end{cases}
\]  

(7.1)

where

\[
\sigma_j = \tanh\left(\frac{u_j(t_j)^2}{\|y_j\|^2}\right) \frac{\|y_j\|^2}{u_j(t_j)^2}.
\]  

(7.2)
and $\epsilon_m > 0$ is a constant bound on the disturbance to the neuron as we will see in next section.

Therefore the SpikePropR weight update rule is

$$W_{ji}(p + 1) = W_{ji}(p) - \eta_j^R(p) \text{diag}(\delta_j) Y_{ji}$$

(7.3)

where $\eta_j^R(p) = \text{diag}([\cdots, \eta_j^R(p), \cdots]^T)$ and the SpikePropR weight update rule for a neuron $N_j$ is

$$w_{ji}(p + 1) = w_{ji}(p) - \eta_j^R(p) \delta_j y_{ji}.$$  

(7.4)

This formulation of learning rate $\eta_j^R(p)$ is similar to the adaptive learning rate scheme in Chapter 5. From theoretical point of view, small value of $\epsilon_m$ yields learning rates similar to that in (5.1) (when $\epsilon_m = 0$). However, larger value of $\epsilon_m$ is desired for robustness to disturbance. There is an trade off between robustness and speed as always.

Note that the quadratic term of $u_j'(t_j)$ in numerator of $\eta_j^R(p)$ suppresses the $u_j'(t_j)$ term in the denominator of $\delta_j$. Further, as we will see in Remark 7.1 that for non-zero $\eta_j^R$, the membrane potential satisfies $u_j'(t_j) > \eta_j(p) \|y_{ji}\|^2$. As a result, the hair trigger problem is not an issue in SpikePropR.

The algorithm for SpikePropR learning is presented in Algorithm 6.

From the point of view of implementation, there is not much additional computational overhead in using SpikePropR. The additional step compared to SpikeProp learning is the computation of adaptive learning rate term, whose parameters $u_j'(t_j)$ and $y_{ji}^{(k)}(t_j)$ are precomputed for SpikeProp as well. Further, compared to forward propagation, the weight update process requires negligible computational time (cf. Chapter 3.2).

Next we will show that learning using SpikePropR is convergent in the sense of weight parameters.

### 7.2 Error System Formulation for SpikePropR

In this section, we will formulate the error feedback system of SNN training in presence of external disturbance. The error system formulation forms an error
7.2. Error System Formulation for SpikePropR

Algorithm 6 SpikePropR Learning

1: set up network connections
2: initialize weights
3: choose disturbance bound $\epsilon_m$
4: repeat
5: $MSE \leftarrow 0$
6: for all training sample do $\triangleright$ assume $m$ samples
7: for all $N_i \in I$ do
8: $t_i \leftarrow input_i$
9: end for
10: for all $N_j \in H, O$ do $\triangleright$ Forward Propagation
11: Evaluate firing time of $N_j$
12: end for
13: for all $N_j \in O, H$ do
14: calculate $\delta_j$ $\triangleright$ equations (3.29) & (3.31)
15: calculate $\eta^R_j$ $\triangleright$ equation (7.1)
16: for $\forall k \leftarrow 1 : |K| \ & i \in \Gamma_j$ do
17: $w^{(k)}_{ji} \leftarrow w^{(k)}_{ji} - \eta^R_j \delta_j y^{(k)}_{ji}(t_j)$
18: end for
19: if $N_j \in O$ then
20: $MSE \leftarrow MSE + E/m$ $\triangleright$ equation (3.26)
21: end if
22: end for
23: end for
24: until $MSE < TOL$ OR MAX_EPOCH is exceeded

feedback for each individual neuron leading to individual error approach to stability.

For robust stability analysis, we will decompose the desired firing time as indicated in (3.10)

$$\hat{t}_o = t_o + \epsilon_o^{ext} \quad (7.5)$$
$$\hat{t}_O = t_O + \epsilon_O^{ext} \quad (7.6)$$

Consider the ideal weight matrices for hidden layer and output layer: $\bar{W}_{HI}$ and $\bar{W}_{OH}$. Then the weight vector for a hidden layer neuron and an output layer neuron is $\bar{w}_{hi}$ and $\bar{w}_{oH}$ respectively. The ideal membrane potential for hidden layer neuron and output layer neuron is

$$\bar{u}_j(t) = \bar{w}_{ji}^T y_{ji}(t) \quad (7.7)$$
the ideal output layer firing time vector is
\[
\bar{t}_O = f_d\left(W_{OH} Y_{OH} \left(f_d\left(W_{HI} Y_{HI}(t_I)^T\right)\right)^T\right) \tag{7.8}
\]
and the ideal hidden layer firing time vector is
\[
\bar{t}_H = f_d\left(W_{HI} Y_{HI}(t_I)^T\right) \tag{7.9}
\]

### 7.2.1 Output Layer Error System

Consider the system error of SNN.

\[
e_O(p) = \bar{t}_O - t_O + \epsilon^\text{ext}_O
= f_d\left(W_{OH} Y_{OH} \left(f_d\left(W_{HI} Y_{HI}(t_I)^T\right)\right)^T\right)
- f_d\left(W_{OH} Y_{OH} \left(f_d\left(W_{HI} Y_{HI}(t_I)^T\right)\right)^T\right) + \epsilon^\text{ext}_O
= f_d\left(W_{OH} Y_{OH} \left(f_d\left(W_{HI} Y_{HI}(t_I)^T\right)\right)^T\right)
- f_d\left(W_{OH} Y_{OH} \left(f_d\left(W_{HI} Y_{HI}(t_I)^T\right)\right)^T\right) + \epsilon^\text{ext}_O
= -\epsilon_O(p) + \epsilon_O(p) \tag{7.10}
\]

Here,

\[
\epsilon_O(p) = f_d\left(W_{OH} Y_{OH} \left(f_d\left(W_{HI} Y_{HI}(t_I)^T\right)\right)^T\right)
- f_d\left(W_{OH} Y_{OH} \left(f_d\left(W_{HI} Y_{HI}(t_I)^T\right)\right)^T\right) \tag{7.11}
\]
is the regression error due to output layer weight estimation error and

\[
\epsilon_O(p) = f_d\left(W_{OH} Y_{OH} \left(f_d\left(W_{HI} Y_{HI}(t_I)^T\right)\right)^T\right)
- f_d\left(W_{OH} Y_{OH} \left(f_d\left(W_{HI} Y_{HI}(t_I)^T\right)\right)^T\right) + \epsilon^\text{ext}_O \tag{7.12}
\]
is the system disturbance consisting of internal modeling disturbance due to imperfect hidden layer weights and external disturbance. It is reasonable to assume that \(\epsilon_O(p)\) is finitely bounded. Therefore, we can have a positive constant \(\epsilon_m\) that...
bounds the elements of $\epsilon_O(p)$, i.e. $\|\epsilon_o(p)\|_\infty = \epsilon_m$.

Now, denote the $o^{th}$ row of matrices $W_{OH}$ and $Y_{OH}$ by $w_{oH}$ and $y_{oH}$ respectively and using mean value theorem on the regression error pertaining to output neuron $N_o$

$$\tilde{e}_o(p) = f\left(w_{oH}y_{oH}\left(f_d\left(W_{HI}Y_{HI}(t_i)^T\right)\right)^T\right)$$

$$- f\left(\bar{w}_{oH}y_{oH}\left(f_d\left(W_{HI}Y_{HI}(t_i)^T\right)\right)^T\right)$$

$$= \frac{\partial f(u_o)}{\partial u_o} \frac{\partial u_o}{\partial w_{oH}} \bigg|_{w_{oH} = \omega} (w_{oH} - \bar{w}_{oH})^T$$

where $\omega = (1 - c)w_{oH} + cw_{oH}$, $c \in (0, 1)$

$$= -\frac{1}{u'_o(t_o)}y_{oH}(w_{oH} - \bar{w}_{oH})^T$$

$$= -\frac{1}{u'_o(t_o)}(u_o(t_o) - \bar{u}_o(t_o))$$

(7.13)

Denote the difference between current and ideal weight by $\hat{W}_{OH}(p) = W_{OH}(p) - W_{OH}$. Then the regression error vector is

$$\hat{e}_O(p) = \frac{\partial f(u_O)}{\partial u_O} \text{diag} \left( \hat{W}_{OH}(p) Y_{OH}^T \right).$$

(7.14)

### 7.2.2 Hidden Layer Error System

Consider the system error again

$$e_O(p) = \tilde{t}_O - t_O + \epsilon^{ext}_O$$

$$= f_d\left(\hat{W}_{OH}Y_{OH}\left(f_d(\hat{W}_{HI}Y_{HI}(t_i)^T)\right)^T\right)$$

$$- f_d\left(W_{OH}Y_{OH}\left(f_d(W_{HI}Y_{HI}(t_i)^T)\right)^T\right) + \epsilon^{ext}_O$$

$$= f_d\left(W_{OH}Y_{OH}\left(f_d(W_{HI}Y_{HI}(t_i)^T)\right)^T\right)$$

$$- f_d\left(W_{OH}Y_{OH}\left(f_d(W_{HI}Y_{HI}(t_i)^T)\right)^T\right) + f_d\left(\hat{W}_{OH}Y_{OH}\left(f_d(\hat{W}_{HI}Y_{HI}(t_i)^T)\right)^T\right)$$

$$- f_d\left(W_{OH}Y_{OH}\left(f_d(W_{HI}Y_{HI}(t_i)^T)\right)^T\right) + \epsilon^{ext}_O$$

$$= -\hat{e}_{O|H}(p) + \epsilon_{O|H}(p)$$

(7.15)
where

\[ \epsilon_{O|H}(p) = f_d(\overline{W}_{OH} Y_{OH} \left( f_d(\overline{W}_{HI} Y_{HI}(t_I))^T \right) \]
\[ - f_d(\overline{W}_{OH} Y_{OH} \left( f_d(\overline{W}_{HI} Y_{HI}(t_I))^T \right) + \epsilon_{ext}^O \]  

(7.16)

is the system disturbance due to external disturbance and internal disturbance caused by imperfect output layer weights and

\[ \hat{\epsilon}_{O|H}(p) = f_d(\overline{W}_{OH} Y_{OH} \left( f_d(\overline{W}_{HI} Y_{HI}(t_I))^T \right) \]
\[ - f_d(\overline{W}_{OH} Y_{OH} \left( f_d(\overline{W}_{HI} Y_{HI}(t_I))^T \right) \]
\[ = M \left( f_d(W_{HI} Y_{HI}(t_I))^T - f_d(W_{HI} Y_{HI}(t_I))^T \right) \]  

(7.17)

is the regression error due to hidden layer weight estimation and \( M \in \mathbb{R}^{O \times H} \) is the mean value matrix which evaluates to the following.

\[ M = \left( \frac{\partial t_O}{\partial \hat{t}_H} \right)_{t_H=(1-c_t) t_H + c_t \hat{t}_H} \]
\[ = \left[ \left( \frac{1}{w_{ho}(t_o) \sum_h \sum_k w_{oh}^{(k)} \epsilon'(t_o - \hat{t}_h + d_{oh}^{(k)})} \right)_{o,h} \right] \]  

(7.18)

Note that the mean value matrix \( M \) is dependent only on output layer weights and the kernel function \( \epsilon'(\cdot) \) which is bounded. Therefore, if we temporarily assume that output layer weights are bounded (which we will show in Section 7.3 for \( j = o \)) then \( M \) is bounded in the sense that its elements are bounded.

Now multiplying both sides of (7.15) by \( M^\dagger \), we obtain

\[ M^\dagger \epsilon_O(p) = -\hat{\epsilon}_H(p) + M^\dagger \epsilon_{O|H}(p) \]
\[ \implies \epsilon_H(p) = -\hat{\epsilon}_H(p) + \epsilon_H(p) \]  

(7.19)

where

\[ \epsilon_H(p) = M^\dagger \left( f_d(\overline{W}_{OH} Y_{OH} \left( f_d(\overline{W}_{HI} Y_{HI}(t_I))^T \right) \]
\[ - f_d(\overline{W}_{OH} Y_{OH} \left( f_d(\overline{W}_{HI} Y_{HI}(t_I))^T \right) + \epsilon_{ext}^O \)  

(7.20)
is the disturbance in hidden layer,

\[
\tilde{\epsilon}_H(p) = f_d(W_{HI}Y_{HI}(t_I)^T) - f_d(W_{HI}Y_{HI}(t_I)^T) \tag{7.21}
\]

is the regression error in hidden layer and

\[
e_H(p) = M^\dagger e_O(p) = \bar{t}_H - t_H + \epsilon_H(p) \tag{7.22}
\]

is the total error in hidden layer due to difference in actual firing time and ideal firing time of hidden layer neurons mixed with disturbance.

Now, if we show that the error signals \(e_H(p)\) and \(\tilde{\epsilon}_H(p)\) in (7.20) are bounded in \(L_2\) and \(L_\infty\) space (which we will show in Section 7.4), the error signals \(e_O(p)\) and \(\tilde{e}_{O|H}(p)\) in (7.15) are also bounded given \(\epsilon_{O|H}(p)\) is bounded in \(L_2\) and \(L_\infty\).

The matrix \(M^\dagger\) redistributes error from output layer to hidden layer. Therefore, \(e_H(p)\) is bounded since \(\epsilon_{O|H}(p)\) is bounded. We can again define a positive constant that bounds the elements of \(e_H(p)\), i.e. \(\|\epsilon_H(p)\|_\infty = \epsilon_m\).

For \(e_H(p)\), the backpropagation estimate of hidden layer neuron error is given by

\[
e_H(p) = -\partial E / \partial t_H = -\partial u_O / \partial t_H \delta_O \tag{7.23}
\]

and for regression error \(\tilde{\epsilon}_H(p)\), consider the \(h^{th}\) row of matrices \(W_{HI}\) and \(Y_{HI}\), denote them by \(w_{hi}\) and \(y_{hi}\) respectively, and using mean value theorem and following similarly as in (7.13) for output layer regression error, we obtain the following.

\[
\hat{\epsilon}_h(p) = -\frac{1}{u'_h(t_h)}(u_h(t_h) - \bar{u}_h(t_h)) \tag{7.24}
\]

And writing the difference between current and ideal weight by \(\tilde{W}_{HI}(p) = W_{HI}(p) - \bar{W}_{HI}\), we get

\[
\tilde{\epsilon}_H(p) = \frac{\partial f(u_H)}{\partial u_H} \text{diag} \left( \tilde{W}_{HI}(p)Y_{HI}^T \right). \tag{7.25}
\]
The approach for weight convergence analysis of SpikePropR uses the Lyapunov function of the same form in Chapter 5.2. The Lyapunov function is defined as the difference between norm of actual and ideal weight vectors in two adjacent iterations. If we show Lyapunov stability of the weight based Lyapunov function, the current weight matrices or vectors always move towards ideal weight, or at least maintain the same distance from ideal weight, at each iteration. The difference is that we will include the external disturbance \( \epsilon_{\text{ext}} \) in the analysis. From the error system analysis in previous subsections, (7.10) and (7.19) mean that the error system of an individual neuron can be written as

\[
e_j(p) = -\dot{e}_j(p) + \epsilon_j(p)
\]

with

\[
\|\epsilon_j(p)\|_{\infty} = \epsilon_m.
\]

Denote the difference between actual and ideal weight matrix as \( \tilde{w}_{jf}(p) = w_{jf}(p) - \bar{w}_{jf} \).

Now, define the Lyapunov function

\[
V_j = \| \tilde{w}_{jf}(p) \|^2
\]

\[
\Delta V_j = \| \tilde{w}_{jf}(p + 1) \|^2 - \| \tilde{w}_{jf}(p) \|^2
\]

**Theorem 7.1.** The weight matrix \( W_{jf}(p) \) is convergent in the sense of Lyapunov function \( \Delta V_j \leq 0 \) if the learning rate, \( \eta_j(p) \), for each neuron at iteration \( p \) satisfies

\[
e_j(p)^2 \left( 1 - \frac{\eta_j(p)}{w'_j(t_j)} \| y_{jf} \|^2 \right) \geq \epsilon_m^2
\]

whenever \( \eta_j(p) \) is nonzero.

**Proof.** When \( \eta_j(p) = 0 \), it is easy to show that \( \Delta V_j = 0 \). For nonzero \( \eta_j(p) \), consider the Lyapunov function

\[
\Delta V_j = \| w_{jf}(p + 1) - \bar{w}_{jf} \|^2 - \| w_{jf}(p) - \bar{w}_{jf} \|^2
\]
\[
\Delta V_j = (\mathbf{w}_{ji}(p) - \eta_j(p) \delta_j \mathbf{y}_{ji})(\mathbf{w}_{ji}(p) - \eta_j(p) \delta_j \mathbf{y}_{ji})^T \\
+ 2\eta_j(p) \delta_j \mathbf{y}_{ji} \tilde{\mathbf{w}}_{ji}^T - \mathbf{w}_{ji}(p) \mathbf{w}_{ji}(p)^T
\]

by using (3.33), we get

\[
\Delta V_j = \eta_j(p)^2 \frac{\epsilon_j(p)^2}{u_j'(t_j)^2} \| \mathbf{y}_{ji} \|^2 + 2\eta_j(p) \epsilon_j(p) \tilde{\epsilon}_j(p)
\]

Now, replacing \(\tilde{\epsilon}_j(p) = \epsilon_j(p) - e_j(p)\) using equation (7.26)

\[
\Delta V_j = \eta_j(p)^2 \frac{\epsilon_j(p)^2}{u_j'(t_j)^2} \| \mathbf{y}_{ji} \|^2 + 2\eta_j(p) (\epsilon_j(p) \epsilon_j(p) - e_j(p)^2)
\]

\[
\leq \eta_j(p)^2 \frac{\epsilon_j(p)^2}{u_j'(t_j)^2} \| \mathbf{y}_{ji} \|^2 + \eta_j(p) \epsilon_j(p)^2 + \eta_j(p) \epsilon_j(p)^2
\]

\[
- 2\eta_j(p) \epsilon_j(p)^2
\]

\[
\leq -\eta_j(p) \left[ -\epsilon_m^2 + \left( 1 - \frac{\eta_j(p)}{u_j'(t_j)^2} \right) \| \mathbf{y}_{ji} \|^2 \right] \epsilon_j(p)^2 \]  

(7.32)

when \(\eta_j > 0\), to have \(\Delta V_j \leq 0\), we must have

\[
-\epsilon_m^2 + \left( 1 - \frac{\eta_j(p)}{u_j'(t_j)^2} \right) \| \mathbf{y}_{ji} \|^2 \epsilon_j(p)^2 \geq 0
\]

which completes the proof.

\(\Box\)

**Remark 7.1.** Note that for (7.30) to be true, there is an implicit requirement for non-zero learning rate that

\[
\eta_j(p) < \frac{u_j'(t_j)^2}{\| \mathbf{y}_{ji} \|^2}
\]

(7.33)
Corollary 7.1.1. The SpikePropR adaptive learning rate (7.1) satisfies Theorem 7.1.

Proof. Note that
\[
\frac{u_j'(t_j)^2}{\|y_{ji}\|^2} > 0 \quad (7.34)
\]
and the fact that \( \tanh x < x \) for \( x > 0 \) and (7.2) implies
\[
\sigma_j < 1 \quad (7.35)
\]
From the definition (7.1), when \( \eta_j^R(p) > 0 \), we have
\[
(1 - \sigma_j)e_j^2 = \left(1 - \frac{\eta_j^R(p)}{u_j'(t_j)^2} \|y_{ji}\|^2\right) e_j^2 \geq e_m^2 \quad (7.36)
\]
which is in agreement with (7.30) i.e. Theorem 7.1. \(\square\)

7.4 Robust Stability Analysis of SpikePropR

From Section 7.2, the error and disturbance signals \( e_j, \hat{e}_j \) and \( e_j \) form a feedback system for SpikePropR parameter adaptation algorithm as similar to (4.1)
\[
\begin{align*}
    e_j(p) &= e_j(p) - r_j(p) = -H_2 \hat{e}_j(p) + e_j(p) \quad (7.37a) \\
    \hat{e}_j(p) &= H_1 e_j(p) \quad (7.37b) \\
    r_j(p) &= H_2 \hat{e}_j(p) \quad (7.37c)
\end{align*}
\]
with \( H_1, H_2 : L_{2e} \rightarrow L_{2e} \) and \( e_j(p), \hat{e}_j(p), e_j(p) \in L_{2e} \). The operator \( H_1 \) represents the SpikePropR learning algorithm and (7.37a) results from (7.26) with operator \( H_2 = 1 \), i.e. \( r_j(p) = \hat{e}_j(p) \). The feedback operator \( H_2 = 1 \) because there is no linear
controller in the weight update of SpikeProp. The closed loop feedback system is depicted in figure 7.1.

We will now show $L_2$ stability of normalized signals with the normalization factor defined as

$$\rho_j(p) = \mu \rho_j(p - 1) + \max \left( \frac{\|y_{ij}\|^2}{u_j'(t_j)^2}, \tilde{\rho} \right) , \ \tilde{\rho} > 0, \mu \in (0, 1) \tag{7.38}$$

which is of the similar form as described in (4.18).

**Theorem 7.2.** If SNN is trained using the SpikePropR adaptive learning rate defined in (7.1) and the disturbance signal $\epsilon_j(p) \in L_2$, then the normalized signals $\tilde{e}_j^n(p), e_j^n(p) \in L_2$ where the normalization factor is defined as in (7.38).

**Proof.** From the definition of norm, $\epsilon_j(p) \in L_2 \implies \epsilon_j^n(p) \in L_2$. We can clearly see from (4.18)

$$\frac{1}{\rho_j(p) u_j'(t_j)^2} \|y_{ij}\|^2 \leq 1$$

From Corollary 7.1.1, SpikePropR satisfies Theorem 7.1, which means $\Delta V \leq 0$. Now multiplying both sides by $\Delta V \leq 0$ from Theorem 7.1 and by using (7.31)

$$\Delta V \leq \frac{\|y_{ij}\|^2}{\rho_j(p) u_j'(t_j)^2} \left( \eta_j(p)^2 \frac{e_j(p)^2}{u_j'(t_j)^2} \|y_{ij}\|^2 + 2\eta_j(p) e_j(p) \tilde{e}_j(p) \right)$$

$$= 2\eta_j(p) \frac{\|y_{ij}\|^2}{\rho_j(p) u_j'(t_j)^2} \left( \eta_j(p) e_j(p)^2 \frac{2u_j'(t_j)^2}{2u_j'(t_j)^2} \|y_{ij}\|^2 + e_j(p) \tilde{e}_j(p) \right)$$

Note that the term inside parenthesis is negative due to Theorem 7.1. For non-zero learning rate, we can have arbitrary small positive constant $\kappa$ such that $\eta_j(p) > \kappa u_j'(t_j)^2/\|y_{ij}\|^2$ then, we can further extend the result to

$$\Delta V \leq 2\kappa \rho_j(p)^{-1} \left( \frac{\sigma_j}{2} e_j(p)^2 + e_j(p) \tilde{e}_j(p) \right). \tag{7.39}$$

Here,

$$\sigma_j = \frac{\eta_j(p)}{u_j'(t_j)^2} \|y_{ij}\|^2 > 0 \tag{7.40}$$
when \( \eta_j > 0 \). Furthermore from (7.1), whenever Theorem 7.1 is satisfied, \( \sigma < 1 \) is guaranteed. Summing both sides from \( p = 0 \) to \( N \) and using the definition of normalized signals (cf. Chapter 4.1.6), we get

\[
\sum_{p=0}^{N} \left[ \frac{\sigma_j}{2} e_j^n(p)^2 + e_j^n(p) \tilde{e}_j(p) + e_j^n(p) \right] \\
\geq \frac{1}{2\kappa} \sum_{p=0}^{N} \left( \| \tilde{w}_{jI}(p+1) \|^2 - \| \tilde{w}_{jI}(p) \|^2 \right) \\
\geq -\frac{1}{2\kappa} \| \tilde{w}_{jI}(0) \|^2 = -\gamma^R
\]

where

\[
\gamma^R = \frac{1}{2\kappa} \| \tilde{w}_{jI}(0) \|^2 \geq 0.
\]

(7.41)

Therefore, condition (4.2a) of Theorem 4.3 is satisfied. For \( H_{n1}^a = H_2 = 1 \), condition (4.2b) implies

\[
\left( \frac{\sigma_j}{2} - 1 \right) \| e_j^n(p) \|^2_N \leq -2\omega \| e_j^n(p) \|^2_N
\]

(7.43)

must be satisfied. Whenever \( \sigma_j/2 - 1 < 0 \) \( \implies \sigma_j < 2 \), it is possible to choose arbitrary \( \omega > 0 \) to satisfy inequality (7.43). Note that from (7.1) and Theorem 7.1, we have \( \sigma_j < 1 \). As a result, condition (4.2b) is also satisfied. This implies, from Theorem 4.3, \( e_j^n(p), \tilde{e}_j(p) \in L^2 \).

Now we will relax the stability conditions by extending \( L^2 \) stability result in \( L^\infty \) space.

**Theorem 7.3.** If SNN is trained using the SpikePropR adaptive learning rate defined in (7.1) and the disturbance signal \( \epsilon_j(p) \in L^\infty \) then the signals \( \tilde{e}_j(p), e_j(p) \in L^\infty \).

**Proof.** First, consider the normalized exponentially weighted signals \( e_j^n(p)\beta, \tilde{e}_j^n(p)\beta \) and \( \epsilon_j^n(p)\beta \). Following similarly as in Theorem 7.2 and transforming the error signals into exponentially weighted signals, we can show the \( L^2 \) stability of the normalized exponentially weighted signals as well i.e. \( (H_1^n)^\beta + \sigma_j/2 \), where \( (H^n_1)^\beta = \beta^p H_1^a / \beta^{-p} \), is passive (condition (4.2a)) and \( (H_2^n)^\beta = \beta^p H_2^a / \beta^{-p} = 1 \) is strictly inside \( \text{Cone}(\sigma_j^{-1}, \sigma_j^{-1}) \) (condition (4.2b)) for SpikePropR leaning. Note that it is

Nanyang Technological University Singapore
the same \( \sigma_j \) defined in (7.2). Also note that \( \| \epsilon_j^n(p) \|_\infty \leq \| \epsilon_j(p) \|_\infty / \sqrt{\bar{\rho}} \) from (4.18).

Now, from Theorem 4.7, \( L_\infty \) stability of \( \tilde{e}_j(p) \) is guaranteed with \( \beta^2 = \mu^{-1} \). With \( \tilde{e}_j(p) \in L_\infty \), it follows from the feedback system that \( e_j(p) \in L_\infty \). This completes the proof. \( \square \)

The only requirement for \( L_\infty \) stability of SpikePropR is the boundedness of the disturbance \( \epsilon_j(p) \). As discussed before, this \( L_\infty \) relaxation of stability result allows the disturbance \( \epsilon_j(p) \) to be within fixed bounds rather than requiring it to be zero after some time for \( L_2 \) stability. Therefore, this \( L_\infty \) extension is rather useful from practical standpoint.

### 7.5 The SpikePropRT Algorithm

In this section, we will present the SpikePropRT algorithm for learning weights in multilayer feedforward SNN as an extension to SpikeProp algorithm. The adaptive learning rate based on dead zone on-off criterion is defined for the whole output layer neurons and hidden layer neurons depending on their current state. It is defined as follows.

Define the normalization factor for output and hidden layer of similar form as in (4.18)

\[
\rho_O(p) = \mu \rho_O(p) + \max \left( \sum_o \frac{\| y_{oH} \|^2}{u_{o}^2(t_o)^2}, \bar{\rho} \right)
\]

(7.44)

and

\[
\rho_H(p) = \mu \rho_H(p) + \max \left( \sum_h \frac{\| y_{hI} \|^2}{u_{h}^2(t_h)^2} \left\| \frac{\partial t_O}{\partial t_h} \right\|^2, \bar{\rho} \right)
\]

(7.45)

where \( \bar{\rho} > 0 \) & \( \mu \in (0, 1) \). The normalized learning rates for output and hidden layer can now be defined as

\[
\eta_O(p) = \begin{cases} \dfrac{\eta}{\rho_O(p)} & \text{if } \sum_o \left( 1 - \dfrac{\| y_{oH} \|^2}{\rho_O(p) u_o^2(t_o)^2} \right) e_o(p)^2 \geq \epsilon_m^2 \\ 0 & \text{otherwise} \end{cases}
\]

(7.46)
and

\[
\eta_H(p) = \begin{cases} 
\frac{\eta}{\rho_H(p)} & \text{if } \left(1 - \sum_h \frac{\eta\|y_h\|^2}{\rho_H(p)\|e_O(p)\|} \|y_h\| \right)^2 \geq \epsilon_m^2 \\
0 & \text{otherwise}
\end{cases}
\] (7.47)

Here \(0 < \eta \leq 1\) is the allowable range of learning rate. The dead zone turn on-off mechanism ensures that the learning system converges in the sense of weight parameter and since the normalization factor is baked in the learning system, the normalized system error is \(L_2\) stable and the denormalized system error feedback system is \(L_\infty\) stable as we will show in next section. The parameter \(\bar{\rho}\) sets the lower bound on the normalization factor and \(\mu\) is the constant that determines the compromise between alertness and robustness of learning [105]. \(\epsilon_m\) on the other hand is the bound of overall disturbances present in the system contributing to total system error.

Therefore the SpikePropRT weight update rule is

\[
W_{ji}(p + 1) = W_{ji}(p) - \eta_j(p) \text{diag}(\delta_j) Y_{ji}
\] (7.48)

and the SpikePropRT weight update rule for a neuron \(N_j\) is

\[
w_{jf}(p + 1) = w_{jf}(p) - \eta_j(p) \delta_j y_{jf}.
\] (7.49)

We can see from the choice of normalization factor for output and hidden layer defined in (7.44) and (7.44) respectively that whenever hair trigger situation arises, i.e. \(u_j'(t_j) \rightarrow 0\) the normalization factor also takes large value which compensates for large \(\delta_j\). As a result, hair trigger problem is always in check.

The SpikePropRT algorithm is presented in Algorithm 7.

Next we will show that learning using SpikePropRT is convergent in the sense of weight parameters.

### 7.6 Error System Formulation for SpikePropRT

In this section, we will formulate the error feedback system of SNN training in presence of external disturbance. We will consider the disturbance to the system as a whole in this formulation.
Algorithm 7 SpikePropRT Learning

1: set up network connections
2: initialize weights
3: choose $\eta$, $\bar{\rho}$, $\epsilon_m$ and $\mu$
4: repeat
5: $MSE \leftarrow 0
6: \text{for all} \ training \ sample \ do
7: \hspace{1em} \text{for all} \ N_i \in I \ do
8: \hspace{2em} t_i \leftarrow \text{input}_i
9: \hspace{1em} \text{end for}
10: \text{for all} \ N_j \in H, O \ do
11: \hspace{1em} \text{Evaluate firing time of } N_j \hspace{1em} \triangleright \text{Fwd Prop}
12: \text{end for}
13: \text{calculate } \eta_O(p) \text{ and } \eta_H(p) \hspace{1em} \triangleright\text{eqns (7.46) & (7.47)}
14: \text{for all} \ N_j \in O, H \ do
15: \hspace{1em} \text{calculate } \delta_j \hspace{1em} \triangleright\text{eqns (3.29) & (3.31)}
16: \hspace{1em} \text{for } \forall \ k \leftarrow 1:|K| \ \& \ \tilde{i} \in \Gamma_j \text{ do}
17: \hspace{2em} w_{ji}^{(k)} \leftarrow w_{ji}^{(k)} - \eta_j(p)\delta_j y_{ji}^{(k)}(t_j) \hspace{1em} \triangleright \tilde{j} = O \text{ or } H
18: \text{end for}
19: \hspace{1em} \text{if } N_j \in O \text{ then}
20: \hspace{2em} \text{MSE} \leftarrow \text{MSE} + E/m \hspace{1em} \triangleright\text{eqn (3.26)}
21: \text{end if}
22: \text{end for}
23: \text{end for}
24: \text{until } MSE < TOL \text{ OR } MAX\_EPOCH \text{ is exceeded}

Similar to Section 7.2, for robust stability analysis, we will decompose the desired firing time as indicated in (3.10)

$$\hat{t}_o = \bar{t}_o + \epsilon_o^{ext} \quad (7.50)$$

$$\hat{t}_O = \bar{t}_O + \epsilon_O^{ext} \quad (7.51)$$

Consider the ideal weight matrices for hidden layer and output layer: $\mathbf{W}_{HI}$ and $\mathbf{W}_{OH}$. Then the weight vector for a hidden layer neuron and an output layer neuron is $\mathbf{w}_{hi}$ and $\mathbf{w}_{oh}$ respectively. The ideal membrane potential for hidden layer neuron and output layer neuron is

$$\bar{u}_j(t) = \mathbf{w}_{ji}^r y_{ji}(t) \quad (7.52)$$
the ideal output layer firing time vector is

\[ \bar{t}_O = f_d \left( \mathbf{W}_{OH} \mathbf{Y}_{OH} \left( f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I) \right) \right)^T \right) \] (7.53)

and the ideal hidden layer firing time vector is

\[ \bar{t}_H = f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I)^T \right) \] (7.54)

### 7.6.1 Output Layer Error System

Consider the error signal

\[
e_O(p) = \bar{t}_O - t_O + \epsilon_{O}^{\text{ext}}
= f_d \left( \mathbf{W}_{OH} \mathbf{Y}_{OH} \left( f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I) \right) \right)^T \right)
- f_d \left( \mathbf{W}_{OH} \mathbf{Y}_{OH} \left( f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I) \right) \right)^T \right) + \epsilon_{O}^{\text{ext}}
= f_d \left( \mathbf{W}_{OH} \mathbf{Y}_{OH} \left( f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I) \right) \right)^T \right)
- f_d \left( \mathbf{W}_{OH} \mathbf{Y}_{OH} \left( f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I) \right) \right)^T \right)
+ f_d \left( \mathbf{W}_{OH} \mathbf{Y}_{OH} \left( f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I) \right) \right)^T \right)
- f_d \left( \mathbf{W}_{OH} \mathbf{Y}_{OH} \left( f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I) \right) \right)^T \right) + \epsilon_{O}^{\text{ext}}
= -\check{\epsilon}_O(p) + \epsilon_O(p) \] (7.55)

Here,

\[
\check{\epsilon}_O(p) = f_d \left( \mathbf{W}_{OH} \mathbf{Y}_{OH} \left( f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I) \right) \right)^T \right)
- f_d \left( \mathbf{W}_{OH} \mathbf{Y}_{OH} \left( f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I) \right) \right)^T \right) \] (7.56)

is the regression error due to output layer weight estimation error and

\[
\epsilon_O(p) = f_d \left( \mathbf{W}_{OH} \mathbf{Y}_{OH} \left( f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I) \right) \right)^T \right)
- f_d \left( \mathbf{W}_{OH} \mathbf{Y}_{OH} \left( f_d \left( \mathbf{W}_{HI} \mathbf{Y}_{HI}(t_I) \right) \right)^T \right) + \epsilon_{O}^{\text{ext}} \] (7.57)

is the system disturbance consisting of internal modelling disturbance due to imperfect hidden layer weights and external disturbance. Define a positive bound of the norm of \( \epsilon_O(p) \), i.e. \( \|\epsilon_O(p)\| \leq \epsilon_m \forall p \).
Now, denote the \( o^{th} \) row of matrices \( W_{OH} \) and \( Y_{OH} \) by \( w_{oH} \) and \( y_{oH} \) respectively and using mean value theorem on the regression error pertaining to output neuron \( N_o \)

\[
\tilde{e}_o(p) = f\left( w_{oH} y_{oH} \left( f_d\left( W_{HI} Y_{HI}(t_I)\right)^T \right) \right) \\
- f\left( \bar{w}_{oH} y_{oH} \left( f_d\left( W_{HI} Y_{HI}(t_I)\right)^T \right) \right) \\
= \frac{\partial f(u_o)}{\partial u_o} \frac{\partial u_o}{\partial w_{oH}} \left|_{w_{oH} = \bar{w}_{oH}} \right) (w_{oH} - \bar{w}_{oH})^T \\
\text{where } \omega = (1 - c)w_{oH} + c\bar{w}_{oH}, \ c \in (0, 1) \\
= -\frac{y_{oH}(w_{oH} - \bar{w}_{oH})^T}{u_o'(t_o)} = -\frac{u_o(t_o) - \bar{u}_o(t_o)}{u_o'(t_o)} \tag{7.58}
\]

### 7.6.2 Hidden Layer Error System

Consider the error signal

\[
\epsilon_O(p) = \tilde{t}_O - t_O + \epsilon_O^\text{ext} \\
= f_d\left( \overline{W}_{OH} Y_{OH} \left( f_d\left( \overline{W}_{HI} Y_{HI}(t_I)\right)^T \right) \right) \\
- f_d\left( W_{OH} Y_{OH} \left( f_d\left( W_{HI} Y_{HI}(t_I)\right)^T \right) \right) + \epsilon_O^\text{ext} \\
= f_d\left( \overline{W}_{OH} Y_{OH} \left( f_d\left( \overline{W}_{HI} Y_{HI}(t_I)\right)^T \right) \right) \\
- f_d\left( W_{OH} Y_{OH} \left( f_d\left( W_{HI} Y_{HI}(t_I)\right)^T \right) \right) + f_d\left( \overline{W}_{OH} Y_{OH} \left( f_d\left( \overline{W}_{HI} Y_{HI}(t_I)\right)^T \right) \right) \\
- f_d\left( W_{OH} Y_{OH} \left( f_d\left( W_{HI} Y_{HI}(t_I)\right)^T \right) \right) + \epsilon_O^\text{ext} \\
= -\epsilon_O|H|(p) + \epsilon_O|H|(p) \tag{7.59}
\]

where

\[
\epsilon_O|H|(p) = f_d\left( W_{OH} Y_{OH} \left( f_d\left( W_{HI} Y_{HI}(t_I)\right)^T \right) \right) \\
- f_d\left( W_{OH} Y_{OH} \left( f_d\left( W_{HI} Y_{HI}(t_I)\right)^T \right) \right) \tag{7.60}
\]

is the regression error due to hidden layer weight estimation and

\[
\epsilon_O|H|(p) = f_d\left( \overline{W}_{OH} Y_{OH} \left( f_d\left( \overline{W}_{HI} Y_{HI}(t_I)\right)^T \right) \right)
\]
is the system disturbance due to external disturbance and internal disturbance caused by imperfect output layer weights. Define a positive bound of the norm of $\epsilon_O|H(p)$, i.e. $\|\epsilon_O|H(p)\| \leq \epsilon_m \forall p$.

Here, we assume that SpikeProp linearization around firing time is applicable. Then from (7.60),

$$
\tilde{e}_O|H(p) \approx \left( \frac{\partial t_O}{\partial t_h} \right)^T \left( f_d(W_{HI}Y_{HI}(t_i)^T) - f_d(W_{HI}Y_{HI}(t_i)^T) \right) = \sum_h \left( \frac{\partial t_O}{\partial t_h} \right)^T (f(w_{HI}y_{HI}^T) - f(\bar{w}_{HI}y_{HI}^T))
$$

Continuing similarly as in (7.58), we get

$$
\tilde{e}_O|H(p) = -\sum_h \left( \frac{\partial t_O}{\partial t_h} \right)^T \frac{1}{u'_h(t_h)} (u_h(t_h) - \bar{u}_h(t_h))
= \left( \frac{\partial t_O}{\partial t_h} \right)^T \frac{\partial f(u_H)}{\partial u_H} \text{diag} \left( \tilde{W}_{HI}(p)Y_{HI}^T \right)
$$

Where, $\tilde{W}_{HI}(p) = W_{HI}(p) - W_{HI}$.

### 7.7 Weight Convergence Analysis of SpikePropRT

The approach for weight convergence analysis of SpikePropRT uses the Lyapunov function of the same form in Chapter 5.2. The Lyapunov function is defined as the difference between norm of actual and ideal weight vectors in two adjacent iterations. If we show Lyapunov stability of the weight based Lyapunov function, the current weight matrices or vectors always move towards ideal weight, or at least maintain the same distance from ideal weight, at each iteration. The difference is that we will include the external disturbance $\epsilon^{ext}$ in the analysis. We will show weight convergence for output layer and hidden layer separately.
7.7.1 Output Layer Weight Convergence Analysis

define a Lyapunov function

\[ V_O = \left\| \hat{\mathbf{W}}_{OH}(p) \right\|_F^2 \]  

\[ \Delta V_O = \left\| \hat{\mathbf{W}}_{OH}(p+1) \right\|_F^2 - \left\| \hat{\mathbf{W}}_{OH}(p) \right\|_F^2 \]  

**Theorem 7.4.** If the output layer of SNN is trained using SpikePropRT with the output layer learning rate defined in (7.46), the weight \( \mathbf{W}_{OH}(p) \) is convergent in the sense of Lyapunov function \( \Delta V_O \leq 0 \).

**Proof.** Consider the Lyapunov function for an individual output neuron

\[ \Delta V_o = \left\| \mathbf{w}_{oh}(p+1) \right\|^2 - \left\| \mathbf{w}_{oh}(p) \right\|^2 \]

\[ = \left\| \mathbf{w}_{oh}(p+1) - \mathbf{w}_{oh} \right\|^2 - \left\| \mathbf{w}_{oh}(p) - \mathbf{w}_{oh} \right\|^2 \]

\[ = (\mathbf{w}_{oh}(p+1) - \mathbf{w}_{oh})(\mathbf{w}_{oh}(p+1) - \mathbf{w}_{oh})^T \]

\[ - (\mathbf{w}_{oh}(p) - \mathbf{w}_{oh})(\mathbf{w}_{oh}(p) - \mathbf{w}_{oh})^T \]

\[ = \mathbf{w}_{oh}(p+1)\mathbf{w}_{oh}(p+1)^T - 2\mathbf{w}_{oh}(p+1)\mathbf{w}_{oh}^T \]

\[ - \mathbf{w}_{oh}(p)\mathbf{w}_{oh}(p)^T + 2\mathbf{w}_{oh}(p)\mathbf{w}_{oh}^T \]

by using (7.49), we get

\[ \Delta V_o = (\mathbf{w}_{oh}(p) - \eta_O(p)\delta_o\mathbf{y}_{oh})(\mathbf{w}_{oh}(p) - \eta_O(p)\delta_o\mathbf{y}_{oh})^T \]

\[ + 2\eta_O(p)\delta_o\mathbf{y}_{oh}\mathbf{w}_{oh}^T - \mathbf{w}_{oh}(p)\mathbf{w}_{oh}(p)^T \]

\[ = \eta_O(p)^2\delta_o^2\left\| \mathbf{y}_{oh} \right\|^2 - 2\eta_O(p)\delta_o(u_o(t_o) - \bar{u}_o(t_o)) \]

by using the definition of regressor \( \tilde{e}_o(p) \) and \( \delta_o \)

\[ \Delta V_o = \eta_O(p)^2 \frac{\epsilon_o(p)^2}{u'_o(t_o)^2} \left\| \mathbf{y}_{oh} \right\|^2 + 2\eta_O(p)\epsilon_o(p)\tilde{e}_o(p) \]  

(7.65)

replacing \( \tilde{e}_o(p) = \epsilon_o(p) - e_o(p) \) by using equation (7.55)

\[ \Delta V_o = \eta_O(p)^2 \frac{\epsilon_o(p)^2}{u'_o(t_o)^2} \left\| \mathbf{y}_{oh} \right\|^2 + 2\eta_O(p)(\epsilon_o(p)e_o(p) - e_o(p)^2) \]

\[ = \eta_O(p)^2 \frac{\epsilon_o(p)^2}{u'_o(t_o)^2} \left\| \mathbf{y}_{oh} \right\|^2 \]
Now, summing $\Delta V_o$ over all output layer neurons, we get

$$
\Delta V_O \leq -\eta_O(p) \left[ -\sum_o \epsilon_o(p)^2 
\right.
\left. + \sum_o \left( 1 - \frac{\eta_O(p) \|y_oH\|^2}{u'_o(t_o)^2} \right) \epsilon_o(p)^2 \right] 
\leq -\eta_O(p) \left[ -\epsilon_m(p)^2 + \sum_o \left( 1 - \frac{\eta \|y_oH\|^2}{\rho_O(p)u'_o(t_o)^2} \right) \epsilon_o(p)^2 \right] 
$$

(7.67)

The choice of learning rate in (7.46) ensures that for non-zero $\eta_O(p)$, the right hand side in the above expression is less than zero. For zero learning rate, the right hand side is trivially zero. Further, the choice of $\rho_O(p)$ ensures that $\left( 1 - \frac{\eta \|y_oH\|^2}{\rho_O(p)u'_o(t_o)^2} \right) \geq 0$. As a result

$$
\Delta V_O \leq 0
$$

which completes the proof. \qed

### 7.7.2 Hidden Layer Weight Convergence Analysis

Define a Lyapunov function

$$
V_H = \|\hat{W}_{HI}(p)\|_F^2 \Delta V_H = \|\hat{W}_{HI}(p + 1)\|_F^2 - \|\hat{W}_{HI}(p)\|_F^2
$$

(7.68)

**Theorem 7.5.** If the hidden layer of SNN is trained using SpikePropRT with the hidden layer learning rate defined in (7.47), the weight $\mathbf{W}_{HI}(p)$ is convergent in the sense of Lyapunov function $\Delta V_H \leq 0$.

**Proof.** Consider the Lyapunov function for an individual hidden neuron and fol-
lowing similarly as in output layer

\[
\Delta V_h = \|\bar{w}_{hI}(p + 1)\|^2 - \|\bar{w}_{hI}(p)\|^2 \\
= \|w_{hI}(p + 1) - \bar{w}_{hI}\|^2 - \|w_{hI}(p) - \bar{w}_{hI}\|^2 \\
= (w_{hI}(p + 1) - \bar{w}_{hI})(w_{hI}(p + 1) - \bar{w}_{hI})^T \\
- (w_{hI}(p) - \bar{w}_{hI})(w_{hI}(p) - \bar{w}_{hI})^T \\
= w_{hI}(p + 1)w_{hI}(p + 1)^T - 2w_{hI}(p + 1)\bar{w}_{hI}^T \\
- w_{hI}(p)w_{hI}(p)^T + 2w_{hI}(p)\bar{w}_{hI}^T \\
\]

by using (7.49), we get

\[
\Delta V_h = (w_{hI}(p) - \eta H(p)\delta_h y_{hI})(w_{hI}(p) - \eta H(p)\delta_h y_{hI})^T + 2\eta H(p)\delta_h y_{hI}w_{hI}(p) \\
= \eta H(p)^2\delta_h^2\|y_{hI}\|^2 - 2\eta H(p)\delta_h (u_h(t_h) - \bar{u}_h(t_h)) \\
\]

by using the definition of (3.31), we get

\[
\Delta V_h = \eta H(p)^2e_O^T(p) \left( \frac{\partial t_O}{\partial t_h} \right)^T \left( \frac{\partial t_O}{\partial t_h} \right) e_O(p) \left\| y_{hI} \right\|^2 \\
- 2\eta H(p)e_O^T(p) \left( \frac{\partial t_O}{\partial t_h} \right)^T \left( \frac{\partial t_O}{\partial t_h} \right) e_O(p) \left\| y_{hI} \right\|^2 \\
- 2\eta H(p)e_O^T(p) \sum_h \left( \frac{\partial t_O}{\partial t_h} \right)^T \left( u_h(t_h) - \bar{u}_h(t_h) \right) \\
\]

Now, summing \(\Delta V_h\) over all hidden layer neurons, we get

\[
\Delta V_H = \eta H(p)^2 \sum_h e_O^T(p) \left( \frac{\partial t_O}{\partial t_h} \right)^T \left( \frac{\partial t_O}{\partial t_h} \right) e_O(p) \left\| y_{hI} \right\|^2 \\
- 2\eta H(p)e_O^T(p) \sum_h \left( \frac{\partial t_O}{\partial t_h} \right)^T \left( u_h(t_h) - \bar{u}_h(t_h) \right) \\
\]

from (7.62)

\[
\Delta V_H = \eta H(p)^2 \sum_h e_O^T(p) \left( \frac{\partial t_O}{\partial t_h} \right)^T \left( \frac{\partial t_O}{\partial t_h} \right) e_O(p) \left\| y_{hI} \right\|^2 \\
+ 2\eta H(p)e_O^T(p)e_O(p) \\
\]

The matrix \(\left( \frac{\partial t_O}{\partial t_h} \right)^T \left( \frac{\partial t_O}{\partial t_h} \right) \) ∈ \(\mathbb{R}^{|O|×|O|}\) is rank 1 dyad (outer product) and has one
non-zero eigenvalue which is $\| \frac{\partial t}{\partial h} \|^2$. Then

$$\Delta V_H \leq \eta_H(p)^2 \sum_h \| \frac{\partial t}{\partial h} \|^2 \| e_O(p) \| \frac{\partial t}{\partial h} \|^2 \| e_O(p) \|^2 \frac{\| y_h \|^2}{u'_h(t_h)^2}$$

$$+ 2\eta_H(p) e_O^T(p) \tilde{e}_O(p)$$

(7.71)

replacing $\tilde{e}_O(p) = e_{O|H}(p) - e_O(p)$ using equation (7.59)

$$\Delta V_H \leq \eta_H(p)^2 \| e_O(p) \|^2 \sum_h \| \frac{\partial t}{\partial h} \| \frac{\partial t}{\partial h} \|$$

$$+ 2\eta_H(p) e_O^T(p) e_{O|H}(p) - 2\eta_H(p) e_O^T(p) e_O(p)$$

$$\leq \eta_H(p)^2 \| e_O(p) \|^2 \sum_h \| \frac{\partial t}{\partial h} \| \frac{\partial t}{\partial h} \|$$

$$+ \eta_H(p) \| e_{O|H}(p) \|^2 - \eta_H(p) \| e_O(p) \|^2$$

$$\leq -\eta_H(p) \left[ -\epsilon^2_m + \left( 1 - \sum_h \frac{\eta \| y_h \|^2}{\rho_H(p) u'_h(t_h)^2} \| \frac{\partial t}{\partial h} \| \right) \right]$$

(7.72)

The choice of learning rate in (7.47) ensures that for non-zero $\eta_H(p)$, the right hand side in the above expression is less than zero. For zero learning rate, the right hand side is trivially zero. Further, the choice of $\rho_H(p)$ ensures that $\left( 1 - \sum_h \frac{\eta \| y_h \|^2}{\rho_H(p) u'_h(t_h)^2} \| \frac{\partial t}{\partial h} \| \right) \geq 0$. As a result

$$\Delta V_H \leq 0$$

which completes the proof.

7.8 Robust Stability Analysis of SpikePropRT

![Figure 7.2: Closed-loop feedback system for SpikePropRT weight update.](image-url)
The error and disturbance signals $e_j(p) = e_O(p), \dot{e}_j(p) = \dot{e}_O(p)$ or $e_{O|H}(p)$ and $e_j(p) = e_O(p)$ or $e_{O|H}(p)$ form a feedback system for a parameter adaptation algorithm as follows:

$$e_j(p) = e_j(p) - r_j(p) = -H_{2,j} e_j(p) + \epsilon_j(p)$$  \hspace{1cm} (7.73a)

$$\ddot{e}_j(p) = H_{1,j} e_j(p)$$ \hspace{1cm} (7.73b)

$$r_j(p) = H_{2,j} \dot{e}_j(p)$$ \hspace{1cm} (7.73c)

with $H_{1,j}, H_{2,j} : L_{2e} \rightarrow L_{2e}$ and $e_j(p), r_j(p), \dot{e}_j(p) \in L_{2e}$. The operator $H_{1,j}$ represents the SpikePropRT learning algorithm and (7.73a) results from (7.55) and (7.59) with operator $H_{2,j} = 1$, i.e. $r_j(p) = \dot{e}_j(p)$. The feedback operator $H_{2,j} = 1$ because there is no linear controller in the weight update of SpikePropRT. For completeness, denote the dimension of the vectors $e(p), \dot{e}(p)$ and $e(p)$ by $M$. Figure 7.2 shows the closed loop feedback system.

**Theorem 7.6.** If SNN is trained using the SpikePropRT adaptive learning rates defined in (7.46) and (7.47) and the disturbances $\epsilon_o(p), \epsilon_{o|H}(p) \in L_2$, then the normalized signals $\tilde{e}_o^n(p), e_o^n(p), \tilde{e}_{o|H}^n(p), e_o^n(p) \in L_2 \forall o$.

**Proof.** The proof consists of two parts: for output layer learning error system and hidden layer learning error system respectively. In this proof we will temporarily assume nonzero learning rates only because the system is stable whenever learning rate is zero.

From the definition of norm, $\epsilon_o(p) \in L_2 \implies \epsilon_o^n(p) \in L_2$. Further, Theorem 7.4 is true for SpikePropRT. Summing both sides of (7.65) for all output layer neurons

$$\Delta V = \sum_o \left[ \eta_o(p) \epsilon_o(p)^2 \right] = 2\eta_o(p) \sum_o \left[ \frac{1}{2} \frac{\eta \| y_{o|H} \|^2}{\rho_o(p) u_o(l_o)^2} \epsilon_o(p)^2 + \epsilon_o(p) \tilde{e}_o(p) \right]$$

$$\leq 2\eta_o(p) \sum_o \left[ \frac{\eta}{2} \epsilon_o(p)^2 + \epsilon_o(p) \tilde{e}_o(p) \right]$$

$$= 2 \frac{\eta}{\rho_o(p)} \left[ \frac{\eta}{2} e_o(p)^T e_o(p) + e_o(p)^T \tilde{e}_o(p) \right]$$ \hspace{1cm} (7.74)

Using definition of normalized signals (cf. Chapter 4.1.6) and summing from
\[ p = 0 \text{ to } N, \] we get
\[
\sum_{p=0}^{N} \left[ \frac{\eta}{2} e_{o}^T(p) e_{o}(p) + e_{o}^T(p) e_{o}(p) \right] \\
\geq \frac{1}{2\eta} \sum_{p=0}^{N} \left[ \| \tilde{W}_{OH}(p+1) \|_F^2 - \| \tilde{W}_{OH}(p) \|_F^2 \right] \\
\geq -\frac{1}{2\eta} \| \tilde{W}_{OH}(0) \|_F^2 = -\gamma^O 
\]
(7.75)

where
\[
\gamma^O = \frac{1}{2\eta} \left\| \tilde{W}_{OH}(0) \right\|_F^2 \geq 0. 
\]
(7.76)

Therefore, condition (4.10a) of Theorem 4.5 is satisfied. For \( H_2^n = H_2 = 1 \), condition (4.10b) implies
\[
-\frac{2 - \eta}{2} \sum_o \| e_{o}^n(p) \|^2_N \leq -2\omega \sum_o \| e_{o}^n(p) \|^2_N 
\]
(7.77)

which is true for arbitrary \( \omega \leq \frac{(2 - \eta)}{4} \). As a result, from Theorem 4.5 \( e_{o}^n(p) \), \( \hat{e}_{o}^n(p) \in L_2 \).

Similarly for hidden layer learning error signals, \( \epsilon_{oH}(p) \in L_2 \implies e_{oH}^n(p) \in L_2 \).

From (7.71)
\[
\Delta V_H \leq \eta_H(p)^2 e_{O}^T(p) e_{O}(p) \sum_h \left[ \frac{\| y_{H}(h) \|^2}{u_{h}(h)} \| t_{O}(p) \|_F^2 + \frac{\eta_H(p) e_{O}^T(p) \hat{e}_{O|H}(p)}{2} \right] \\
\leq 2 \frac{\eta}{\rho_H(p)} \left[ \frac{\eta_H(p) e_{O}^T(p) e_{O}(p) + e_{O}^T(p) \hat{e}_{O|H}(p)}{2} \right] 
\]
(7.78)

Using definition of normalized signals (cf. Chapter 4.1.6), summing from \( p = 0 \) to \( N \), and proceeding similarly as for output layer, we obtain
\[
\sum_{p=0}^{N} \left[ \frac{\eta}{2} e_{o}^T(p) e_{o}(p) + e_{o}^T(p) e_{o}(p) \right] \geq -\gamma^H 
\]
(7.79)

where
\[
\Delta V_H = \frac{1}{2\eta} \left\| \tilde{W}_{H}(0) \right\|_F^2 \geq 0. 
\]
(7.80)

This satisfies condition (4.10a) of Theorem 4.5. For \( H_2^n = H_2 = 1 \), condition
(4.10b) implies
\[-\frac{2 - \eta}{2} \sum_o \|\hat{e}_o^n(p)\|_N^2 \leq -2\omega \sum_o \|\hat{e}_o^n(p)\|_N^2.\] (7.81)

We can have arbitrary $\omega \leq \frac{(2 - \eta)}{4}$ that satisfies the condition above. Therefore, from Theorem 4.5, $\hat{e}_o^n(p), \hat{e}_o|H(p) \in L_2$. This completes the proof.

Now we will relax the stability conditions by extending $L_2$ stability result in $L_\infty$ space.

**Theorem 7.7.** If the SNN is trained using the SpikePropRT adaptive learning rates defined in (7.46) and (7.47) and the disturbances $\epsilon_o(p), \epsilon_o|H(p) \in L_\infty$, then the signals $\hat{e}_o(p), e_o(p), \hat{e}_o|H(p), e_o(p) \in L_\infty \forall o$.

**Proof.** For output layer learning error signals, consider the normalized exponentially weighted signals $e_o^n(p)^\beta, \hat{e}_o^n(p)^\beta$ and $\epsilon_o^n(p)^\beta$. Following similarly as in Theorem 7.6 and transforming the error signals into exponentially weighted signals, we can show the $L_2$ stability of the normalized exponentially weighted signals as well i.e. $(H_{1,o}^n)^\beta + \sigma/2$, where $(H_{1,o}^n)^\beta = \beta^p H_{1,o}^n \beta^{-p}$, is passive (condition (4.2a)) and $(H_{2,o}^n)^\beta = \beta^p H_{2,o}^n \beta^{-p} = 1$ is strictly inside $\text{Cone}(\eta^{-1}, \eta^{-1})$ (condition (4.2b)) for adaptive learning rates defined in (7.46) and (7.47). Also note that $\|\epsilon_o^n(p)\|_\infty \leq \|\epsilon_o(p)\|_\infty / \sqrt{p}$ from (7.44).

Now, from Theorem 4.7, $L_\infty$ stability of $\hat{e}_o(p)$ is guaranteed with $\beta^2 = \mu^{-1}$. With $\epsilon_o(p) \in L_\infty$, it follows from the feedback system that $e_o(p) \in L_\infty$. This proves the stability for output layer.

Now following similar procedure for hidden layer error system, and using Theorem 4.7, the result follows for hidden layer learning errors.

The only requirement for $L_\infty$ boundedness is the boundedness of the disturbance $\epsilon_j(p)$. Therefore, this $L_\infty$ extension of stability result is very useful in practice.

### 7.9 Performance Comparison

Now, we will compare the performance of SpikePropR, SpikePropRT and other relevant learning methods for SNN viz. the adaptive delay learning modification to SpikeProp, SpikePropAd and RProp. We will compare these methods based
on results of simulations on four benchmark datasets: XOR, Fisher’s Iris data, Wisconsin Breast Cancer data [110] and Statlog (Landsat Satellite) data [110]. As discussed in Chapter 3.7, we will compare the simulation results based on their convergence rate (cf. Chapter 3.7.2), average number of epochs required to reach a tolerance value (cf. Chapter 3.7.3), training and testing accuracy (cf. Chapter 3.7.4) and average learning curve (cf. Chapter 3.7.5).

The SNN architecture used is a three layer architecture as described in Chapter 3.1. The weights are initialized randomly based on normal distribution using the method described in Chapter 3.3.2. We will now present the simulation results on each of the datasets and discuss the performance results.

### 7.9.1 XOR problem

The simulation parameters for XOR classification are listed below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Architecture</td>
<td>$</td>
</tr>
<tr>
<td>Synaptic Delays</td>
<td>1 : 1 : 16</td>
</tr>
<tr>
<td>SRM time constant</td>
<td>$\tau_s = 7$</td>
</tr>
<tr>
<td>Activation Threshold</td>
<td>$\vartheta = 1$</td>
</tr>
<tr>
<td>Simulation Interval</td>
<td>0 : 0.01 : 25 units</td>
</tr>
<tr>
<td>Weight Initialization</td>
<td>$t_{\text{min}} = 0.1$ units &amp; $t_{\text{max}} = 10$ units</td>
</tr>
<tr>
<td>Dead Neuron Boosting</td>
<td>boost by 0.05</td>
</tr>
</tbody>
</table>

The input output spike encoding is same as in Chapter 5.3.1.

The results are based on 1000 different simulations using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT with a maximum cap of 5000 epochs to label non-converging instances. The plot of convergence rate in figure 7.3 shows that SpikePropR has excellent convergence rate along with SpikePropAd. SpikePropRT has good convergence rate, however, it’s performance is not as good as SpikePropR and SpikePropAd. This is possibly because XOR is a synthetic dataset, therefore, there is no external disturbance. However, the SpikePropRT algorithm is conservative in its approach. The average number of epochs required to reach different minimum mean square error values is plotted in figure 7.4. The lower limit of error bar indicates the minimum number of epochs required and the upper error bar indicates the standard deviation of the epochs required. We can see that SpikePropAd is the fastest with SpikePropR closely behind. SpikePropRT on the other hand is the slowest among the lot. For SpikePropR, the smaller value
of the parameter $\epsilon_m$ yields better results. Note that for smaller $\epsilon_m$ and $\lambda$, SpikePropR and SpikePropAd yield similar learning rate. On the other hand, there is not much substantial difference in the number of epochs required among the two parameter set for SpikePropRT, even for smaller value of disturbance bound $\epsilon_m$.

We have plotted the average cost from 1000 different trails for all the different methods in figure 7.5. Although, the mean square error of SpikePropRT does not drop as fast as SpikePropAd and SpikePropR, the mean square error slowly, but steadily keeps on decreasing and eventually catches up with SpikePropAd and reaches the lowest value. The average learning performance is also similar between SpikePropRT with different parameter sets. Figure 7.6 shows a typical learning curve for XOR training. It can be seen that the occurrence of surge is far less

---

**Figure 7.3:** Convergence Rate plot for learning XOR pattern using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

**Figure 7.4:** Average number of epochs required to learn XOR pattern using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.
Figure 7.5: Average learning curves for XOR problem using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

Figure 7.6: Typical learning curves for XOR problem using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

in SpikePropR, SpikePropRT and SpikePropAd compared to RProp. A typical

Figure 7.7: Trace of output and hidden layer learning rate in a typical XOR learning using SpikePropR ($\epsilon_m = 0.001$).
trace of learning rate in each iteration for output layer and hidden layer during
training with SpikePropRT ($\epsilon_m = 0.001$) for XOR problem is shown in figure 7.7
showing the continuous variation of the learning rate. The periodic variation of
learning rate for every 4 iteration is visible. A typical trace of learning rate in

![Figure 7.8: Trace of output and hidden layer learning rate in a typical XOR
learning using SpikePropRT ($\eta = 1, \bar{\rho} = 1, \epsilon_m = 0.01, \mu = 0.5$).](image)

each iteration for output layer and hidden layer during training with SpikePropRT ($\eta = 1, \bar{\rho} = 1, \epsilon_m = 0.01, \mu = 0.5$) for XOR problem is shown in figure 7.8
showing the dead zone rule and variable learning rate mechanism in effect.

### 7.9.2 Fisher’s Iris classification

The simulation parameters for Fisher’s Iris classification are listed below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Architecture</td>
<td>$</td>
</tr>
<tr>
<td>Synaptic Delays</td>
<td>1 : 1 : 16</td>
</tr>
<tr>
<td>SRM time constant</td>
<td>$\tau_s = 7$</td>
</tr>
<tr>
<td>Activation Threshold</td>
<td>$\vartheta = 1$</td>
</tr>
<tr>
<td>Simulation Interval</td>
<td>0 : 0.01 : 25 units</td>
</tr>
<tr>
<td>Weight Initialization</td>
<td>$t_{\text{min}} = 0.1$ units &amp; $t_{\text{max}} = 10$ units</td>
</tr>
<tr>
<td>Dead Neuron Boosting</td>
<td>boost by 0.05</td>
</tr>
</tbody>
</table>

The input and output spikes are encoded similarly as in Chapter 5.3.2.

100 independent simulations were performed using using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT and maximum epochs of 1500 was considered to label non-converging instances. Figure 7.9 compares the convergence rate of all the methods. We can see SpikePropRT (both parameter options) show very
good convergence rate with only two out of hundred instances failing to reach tolerance of 0.01. SpikePropRT is close in terms of convergence rate with $\epsilon_m = 0.001$ performing better. On the other hand, the performance of SpikePropAd is midway between the two parameters of SpikePropR, but not as good as SpikePropRT.

Figure 7.9: Convergence Rate plot for learning Fisher’s Iris data using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

In figure 7.10, mean number of epochs required is plotted. The lower point in error bar denotes the minimum number of epoch required and the upper length of the error bar indicates the standard deviation of the epochs. We can see that SpikePropRT ($\eta = 1$, $\bar{\rho} = 0.1$, $\epsilon_m = 0.001$, $\mu = 0.01$) is the fastest among the lot with other SpikePropRT instances quite close to it. SpikePropR and SpikePropAd also achieve learning in less number of epochs, similar number of epochs...
among them. This performance is better than basic SpikeProp, however it is not as good as RProp, which in turn is not as good as SpikePropRT. Figure 7.11 and

![Figure 7.11: Training accuracy in Fisher’s Iris classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.](image1)

![Figure 7.12: Testing accuracy in Fisher’s Iris classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.](image2)

figure 7.12 show the plot of training accuracy and testing accuracy respectively corresponding to different final mean square values for all the methods. The upper error bar position indicates the maximum accuracy achieved and the length of lower error bar indicates the standard deviation in accuracy for both training and testing accuracy. SpikePropAd, SpikePropR and SpikePropRT show similar accuracy performance, which is better than the rest. RProp on the other hand shows big dip in both training and testing accuracy with decrease in final mean square error, which is strange! Average learning plots over 100 instances of Fisher’s Iris
Learning is shown in figure 7.13. SpikePropRT (\( \eta = 1, \bar{\rho} = 0.1, \epsilon_m = 0.001, \mu = 0.01 \)) has the best performance among all the methods and SpikePropRT in general shows better performance than SpikeProp, RProp, SpikePropAd and SpikePropR. The performance of SpikePropR and SpikePropAd is similar. From the typical learning curve in figure 7.14, we can see that SpikePropRT, SpikePropR and SpikePropAd have lot less and smaller surges. The trace of learning rate in SpikePropR during Fisher’s Iris learning is shown in figure 7.15 for output layer and hidden layer. The trace of learning rate in SpikePropRT during Fisher’s Iris learning is shown in figure 7.16 for output layer and hidden layer.
7.9. Performance Comparison

Figure 7.15: Trace of output and hidden layer learning rate in a typical Fisher’s Iris learning using SpikePropRT ($\epsilon_m = 0.001$) for first 1000 iterations.

Figure 7.16: Trace of output and hidden layer learning rate in a typical Fisher’s Iris learning using SpikePropRT ($\eta = 1$, $\bar{\rho} = 0.1$, $\epsilon_m = 0.001$, $\mu = 0.01$) for first 2000 iterations.

7.9.3 Wisconsin Breast Cancer classification

The simulation parameters for Wisconsin Breast Cancer classification problem are listed below.

- **Network Architecture**: $|\mathcal{I}| = 64$, $|\mathcal{H}| = 15$, $|\mathcal{O}| = 2$ & $|\mathcal{K}| = 16$
- **Synaptic Delays**: $1 : 1 : 16$
- **SRM time constant**: $\tau_s = 7$
- **Activation Threshold**: $\vartheta = 1$
- **Simulation Interval**: $0 : 0.01 : 25$ units
- **Weight Initialization**: $t_{\text{min}} = 0.1$ units & $t_{\text{max}} = 10$ units
- **Dead Neuron Boosting**: boost by 0.05

The input and output spikes are encoded similarly as in Chapter 5.3.2.
100 independent simulations using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT were performed and 1000 epochs were considered before labeling the simulation trial as non-converging. Comparison of convergence rate of all the methods is shown in figure 7.17 Wisconsin Breast Cancer learning. All the instances of SpikePropRT show convergence rate of 1, which is excellent. SpikePropR on the other hand shows good convergence rate of 0.84, which is good and slightly better than SpikePropAd. The plot of number of average epochs required

Figure 7.17: Convergence Rate plot for learning Wisconsin Breast Cancer data using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

Figure 7.18: Average number of epochs required to learn Wisconsin Breast Cancer data using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

in figure 7.18 show that SpikePropRT requires very few epoch to learn compared to SpikeProp, RProp and SpikePropAd. SpikePropR on the other hand has comparable speed as SpikePropAd. In figure 7.18, the lower error bar corresponds to
the minimum number of epoch and the length of upper error bar indicates the standard deviation of epochs. The plot of training and testing accuracy obtained

![Training Accuracy Graph](image1)

**Figure 7.19:** Training accuracy in Wisconsin Breast Cancer classification using using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

![Testing Accuracy Graph](image2)

**Figure 7.20:** Testing accuracy in Wisconsin Breast Cancer classification using using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT is shown in figure 7.19 and figure 7.20 respectively. SpikeProp, SpikePropAd, SpikePropR and SpikePropRT have good learning accuracy performance, which is in similar range and better than RProp. The average cost function over 100 different instances is plotted in figure 7.21 and typical learning curve is presented in figure 7.22 for Wisconsin Breast cancer data training. We can see that SpikePropRT shows significantly better average learning rate and very few surges as well. SpikePropR has similar average learning curve as SpikePropAd with minimal surges as well,
Figure 7.21: Average learning curves during Wisconsin Breast Cancer classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

Figure 7.22: Typical learning curves during Wisconsin Breast Cancer classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

Figure 7.23: Trace of output and hidden layer learning rate in a typical Wisconsin Breast Cancer classification using SpikePropR ($\epsilon_m = 0.001$) for first 5000 iterations.
but not as good as SpikePropRT. The trace of learning rate for an output neuron and a hidden neuron for first 10000 iterations using SpikePropRT during Wisconsin Breast Cancer classification learning is plotted in figure 7.23. The trace of

\[ \eta_O(p) \]

\[ \eta_H(p) \]

Figure 7.24: Trace of output and hidden layer learning rate in a typical Wisconsin Breast Cancer classification using SpikePropRT \((\eta = 1, \bar{\rho} = 0.1, \epsilon_m = 0.01, \mu = 0.05)\) for first 10000 iterations.

learning rate for an output neuron and a hidden neuron for first 10000 iterations using SpikePropRT during Wisconsin Breast Cancer classification learning is plotted in figure 7.24. It is interesting to note that the learning rate goes as high as 9, however, the dead zone rule keeps the learning stable in SpikePropRT.

### 7.9.4 Statlog (Landsat Satellite) data classification

The simulation parameters for Statlog (Landsat Satellite) data classification problem are listed below.

| Network Architecture | \(|I| = 101, |H| = 25, |O| = 6 & |K| = 16\) |
|----------------------|---------------------------------|
| Synaptic Delays      | \(1 : 1 : 16\) |
| SRM time constant    | \(\tau_s = 7\) |
| Activation Threshold | \(\vartheta = 1\) |
| Simulation Interval  | \(0 : 0.01 : 25\) units |
| Weight Initialization| \(t_{\text{min}} = 0.1\) units & \(t_{\text{max}} = 10\) units |
| Dead Neuron Boosting | boost by 0.05 |

The input and output spikes are encoded similarly as in Chapter 5.3.2.

10 independent simulations were performed using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT and a maximum of 1000 epochs was used before
labeling the instance as non-converging. The convergence rate is plotted in figure 7.25. SpikePropR, SpikePropRT and SpikePropAd demonstrate the best convergence rate. The average number of epochs required to reach different mean square values is plotted in figure 7.26. The lower error bar corresponds to the minimum number of epoch and the length of upper error bar indicates the standard deviation of epochs. We can see that SpikePropRT required lot less number of epochs compared to other methods. SpikePropR and SpikePropAd required similar number of epochs. The training accuracy and testing accuracy achieved for different MSE tolerance values is depicted in figure 7.27 and 7.28 respectively. It can be seen that SpikePropR, SpikePropRT, SpikePropAd and SpikeProp show comparable accuracy performance as well.

Figure 7.25: Convergence Rate plot for learning Statlog (Landsat Satellite) data using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

Figure 7.26: Average number of epochs required to learn Statlog (Landsat Satellite) data using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.
Ch. 7. Robust Learning in SpikeProp

7.9. Performance Comparison

Figure 7.27: Training accuracy in Statlog (LandSat Satellite) classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

Figure 7.28: Testing accuracy in Statlog (LandSat Satellite) classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

The average learning cost for each of the methods is plotted in figure 7.29 and a typical learning cost for each of the methods is plotted in figure 7.30. We can see that SpikePropRT has better performance in terms of average learning cost. There is not much difference between different parameters of SpikePropRT as well. SpikePropR shows similar average learning curves as SpikePropAd. Also, in terms of surges, SpikePropAd, SpikePropR and SpikePropRT show superior performance. Figure 7.31 shows the trace of learning rate for an output neuron and a hidden neuron for first 1000 iterations using SpikePropRT (η = 1, \(\bar{\rho} = 0.1\), \(\epsilon_m = 0.01\), \(\mu = 0.5\)) during Statlog (LandSat Satellite) data classification learning process. Figure 7.32 shows the trace of learning rate for an output neuron and a hidden neuron for first 1000 iterations using SpikePropRT (η = 1, \(\bar{\rho} = 0.1\), \(\epsilon_m = 0.01\), \(\mu = 0.5\)) during Statlog (LandSat Satellite) data classification learning process.
Figure 7.29: Average learning curves during Statlog (Landsat Satellite) classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

Figure 7.30: Typical learning curves during Statlog (Landsat Satellite) classification using SpikeProp, RProp, SpikePropAd, SpikePropR and SpikePropRT.

Figure 7.31: Trace of learning rate of a neuron in a typical Statlog (Landsat Satellite) data classification using SpikePropR ($\epsilon_m = 0.0001$) for first 5,000 iterations.
7.10 Choice of SpikePropR Learning Parameter

The learning performance of SpikePropR for different values of $\epsilon_m$ is tabulated in Table 7.1. As the value of $\epsilon_m$ is increased, the learning becomes more conservative. If the value of 0.1 or higher is chosen, the inequality condition in SpikePropR learning becomes stringent; therefore, resulting in zero learning rate i.e., no learning at all.

7.11 Choice of SpikePropRT Learning Parameter

There are four tunable parameters in SpikePropRT learning: $\eta$, $\bar{\rho}$, $\mu$ and $\epsilon_m$. The results for different variation of these parameters is listed in Table 7.2. The results confirm that $\eta/\bar{\rho}$ effect the max learning rate and hence the speed of learning. $\mu$ controls the robustness vs activeness of learning and $\epsilon_m$ controls the allowable error in the system.
## Table 7.1: Variation of $\epsilon_m$ in SpikePropR.

<table>
<thead>
<tr>
<th>SpikePropR</th>
<th>$\epsilon_m$</th>
<th>Tol = 0.01</th>
<th>Conv</th>
<th>Epoch</th>
<th>Conv</th>
<th>Epoch</th>
<th>Tol = 0.001</th>
<th>Conv</th>
<th>Epoch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Conv</td>
<td></td>
<td>Epoch</td>
<td></td>
<td></td>
<td>Conv</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>0.1</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0.01</td>
<td>0.996</td>
<td>140.27</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>0.999</td>
<td>76.67</td>
<td>0.997</td>
<td>198.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0001</td>
<td>0.998</td>
<td>72.76</td>
<td>0.997</td>
<td>103.85</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tol = 0.1</td>
<td>Tol = 0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fisher</td>
<td>0.1</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0.01</td>
<td>1</td>
<td>78.42</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>1</td>
<td>78.01</td>
<td>0.97</td>
<td>466.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0001</td>
<td>1</td>
<td>90.36</td>
<td>0.92</td>
<td>507.22</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tol = 0.1</td>
<td>Tol = 0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wisconsin</td>
<td>0.1</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0.01</td>
<td>0.99</td>
<td>264.95</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>0.94</td>
<td>320.53</td>
<td>0.54</td>
<td>524.91</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0001</td>
<td>1</td>
<td>191.4</td>
<td>0.76</td>
<td>508.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tol = 6</td>
<td>Tol = 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statlog</td>
<td>0.1</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0.01</td>
<td>7.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>1</td>
<td>10.3</td>
<td>1</td>
<td>385.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0001</td>
<td>1</td>
<td>15.9</td>
<td>1</td>
<td>323.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 7.11. Choice of SpikePropRT Learning Parameter

<table>
<thead>
<tr>
<th></th>
<th>SpikePropRT</th>
<th>Tol = 0.01</th>
<th>Tol = 0.0001</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>ρ</td>
<td>εₘ</td>
</tr>
<tr>
<td>XOR</td>
<td>1</td>
<td>1</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>1</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.1</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Table 7.2: Variation of learning parameters in SpikePropRT.

- XOR
- Fisher
- Wisconsin
- Statlog
Chapter 8

Event based SpikeProp for Learning Spike Train

Until now, the methods presented in this thesis for learning in an SNN with hidden layer consisted of various stable extensions to SpikeProp. These methods use the advantage of hidden layer, however, they are limited to the first spike of the neuron. There are methods that can learn a series of spikes, known as spike train (cf. Chapter 3). However, these methods are limited to training a single neuron only. Nevertheless, there are Multi-ReSuMe [38] and SpikeProp-Multi [95] that use the power of hidden layer dynamics and efficient information transfer capability of spike train, albeit with their own sets of drawbacks.

Multi-ReSuMe method is based on the assumption that the input spike train is linearly transformed into output spike train through linear interaction with weights. In reality, there is a complex non-linear dynamic relation between input spike train to a spiking neuron and the output spike train it is emitting. Therefore this linear interaction of spike train is a rather loose assumption. Further, it needs the interaction between all the recorded spikes which makes it slower as the number of spikes increases. SpikeProp-Multi on the other hand considers the non-linear internal dynamics of a spiking neuron. The issue, however, is that it assumes that the number of desired spikes and number of actual spikes that occurred in a simulation interval must be same. This condition cannot always be guaranteed in practice.

In this Chapter, we will introduce event based weight update strategy which updates weight for each spike of a neuron in a continuous fashion allowing us to
learn a spike train pattern as the spikes come and go in a manner similar to online learning. We will call this method EvSpikeProp. Similar to Chapter 5 and 7, we will perform weight convergence analysis of this event based weight update method and show its robustness against learning disturbance, be it in the form of disturbance in desired spike sample or internal disturbance due to modeling imperfection, and hence stability of the learning process in $L_2$ and $L_\infty$ space. The robust extension of EvSpikeProp will be denoted by EvSpikePropR.

We will first introduce event based weight update strategy and formulate the weight update rule. Next, we will propose adaptive learning rate rule similar to SpikePropRT (cf. Chapter 7.5) and then proceed to its weight convergence analysis and robust stability analysis. Finally, we will compare the performance based on standard simulation results.

### 8.1 The EvSpikeProp Algorithm

The EvSpikeProp also follows the principle of backpropagation for weight update. It is similar to the MultiSpikeProp learning rule by Ghosh et al. [17] (cf. Chapter 3.4.4), however, there is slight difference in the weight update rule and the major distinction is in the manner in which weight update is scheduled. We will first formalize the SNN model and then formulate the EvSpikeProp learning rule.

#### 8.1.1 Spiking Neuron Model and Network Architecture

The SNN architecture is the same three-layer-fully-connected feedforward architecture for SpikeProp and its extensions as depicted in figure 3.1 (cf. Chapter 3.1). The spiking neuron model, on the other hand is the short term memory neuron (cf. Chapter 2.2.4). We will reiterate the model equation below.

$$u_j(t) = \nu(t - t_j^{(f-1)}) + \mathbf{w}_j^T \mathbf{y}_j(t, t_f)$$

$$t_j^{(f)} : \quad u_j(t_j^{(f)}) = \vartheta, \quad u'_j(t_j^{(f)}) > 0, \quad t_j^{(f)} > t_j^{(f-1)}$$

For completeness, denote the function that maps membrane potential of neuron to spike time as

$$t_j^{(f)} = f(u_j)$$
denote a function that maps membrane potential into a vector consisting of all the multiple spikes.

At all spike event, we must satisfy \( u_j(t_j^{(f)}) = \partial \forall t_j^{(f)} \), taking differential with respect to a weight on both sides, we get

\[
\frac{du_j(t_j^{(f)})}{dw_{ji}^{(k)}} = 0
\]

\[
\Rightarrow \frac{\partial u_j(t_j^{(f)})}{\partial w_{ji}^{(k)}} + \frac{\partial u_j(t_j^{(f)})}{\partial t_j^{(f)}} \frac{\partial t_j^{(f)}}{\partial w_{ji}^{(k)}} = 0
\]

\[
\Rightarrow \frac{\partial u_j(t_j^{(f)})}{\partial w_{ji}^{(k)}} + u_j'(t_j^{(f)}) \frac{\partial u_j(t_j^{(f)})}{\partial u_j(t_j^{(f)})} \frac{\partial u_j(t_j^{(f)})}{\partial w_{ji}^{(k)}} = 0
\]

\[
\Rightarrow \frac{\partial t_j^{(f)}}{\partial u_j(t_j^{(f)})} = \frac{\partial f(u_j)}{\partial u_j(t_j^{(f)})} = -\frac{1}{u_j'(t_j^{(f)})}
\]

which is the SpikeProp rule that relates output spike as a function of membrane potential, extended for the case of multiple spike events.

### 8.1.2 Event Based Weight Update

One of the major issues with SpikeProp-Multi [95] is that the number of actual firing at the output layer must match the number of desired firing time. This condition is not always met in practical situations. The issue arises because we need to wait for the complete simulation interval to update weight. In this section, we propose weight update for each spike event at output layer to overcome this problem. The term event, henceforth, refers to the spike event of a spiking neuron. This enables us to update weight in an online fashion, which means that we can keep on learning indefinitely. In addition, there is always a desired output firing time, be it quite a time beyond, to pair with output firing time as well. In backpropagation, the error needs to be distributed backward from output layer. As a result, for hidden layer weight update, we need to wait for error value at the output layer. Therefore, we wait for spike event at output layer and update weights for all spike events that occur between the recent spike and penultimate spike at the output layer. This strategy is illustrated in figure 8.1.
8.1. The EvSpikeProp Algorithm

Formally, an iteration \( p \) consists of time interval \( (t^{(f-1)}_a, t^{(f)}_a) \). In every iteration, we update weights of each neuron for all of its firing event. The weight update is dependent on the spike event of output neuron. The hidden layer weight update rule is also linked to hidden layer neuron spike event as well. Because hidden layer weight is linked to output layer error, we need to wait for output firing event to update hidden layer weight. One should also note that for multiple neurons in output layer, the output spiking event occur independently. Therefore, this rule is applicable in that case also, although the time of weight update is different depending on which output neuron spikes first.

8.1.3 The Recurrent Factor

The refractory response means that there is a self recurrence in a spiking neuron, meaning previous spike have an effect on the upcoming spike. To simplify representation, we introduce a recurrent factor similar to Real Time Recurrent Learning (RTRL) \([4]\), defined as

\[
\lambda^{(f)}_{ji} = \frac{\partial t^{(f)}_j}{\partial w_{ji}} \in \mathbb{R}^{\mid\mathcal{I}\mid\mid\mathcal{K}\mid}
\]  \hspace{1cm} \text{(8.5)}

Now evaluating recurrent factor and by using (8.4)

\[
\lambda^{(f)}_{ji} = \frac{\partial u_j(t^{(f)}_j)}{\partial w_{ji}} \frac{\partial t^{(f)}_j}{\partial u_j(t^{(f)}_j)} + \frac{\partial t^{(f-1)}_j}{\partial w_{ji}} \frac{\partial t^{(f)}_j}{\partial t^{(f-1)}_j}
\]
8.1.4 Error Backpropagation

Using event based weight update strategy (cf. Section 8.1.2), without loss of
generality, we will consider a single output neuron in this derivation. Same scheme
is applicable for multiple output neurons as well. Consider the time frame after
output neuron spike at $t_{o}^{(f-1)}$. Assume that the next output neuron firing time is
at $\bar{t}_{o}^{(f)}$ and we desire it to be at $\tilde{t}_{o}^{(f)}$. We will denote the optimal parameters by
$(\bar{\cdot})$. Then the optimal membrane potential is given by

$$\bar{u}_{j}(t) = \nu(t - \tilde{t}_{j}^{(f-1)}) + \bar{w}_{j}^{T}y_{j}(t, t_{i})$$

and the optimal output firing times is given by

$$\bar{t}_{o}^{(f)} = f(\bar{w}_{oH}^{T}y_{oH}(f_{m}(W_{HI}Y_{HI}(t_{I}^{T}))))$$

In practical system, the desired learning targets are inevitably corrupted by ex-
ternal disturbances. If we denote the external disturbance in desired firing time
by $\epsilon_{o}^{ext} \in L_{2}, L_{\infty}$, the error in the output layer is given by

$$e_{o}^{(f)} = \bar{t}_{o}^{(f)} - t_{o}^{(f)} + \epsilon_{o}^{ext}$$

and the cost function is

$$E^{(f)} = \frac{1}{2} \{e_{o}^{(f)}\}^2$$

For Output layer, we get

$$\frac{\partial E^{(f)}}{\partial w_{oH}} = \frac{\partial t_{o}^{(f)}}{\partial w_{oH}} \frac{\partial E^{(f)}}{\partial t_{o}^{(f)}} = -e_{o}^{(f)} \lambda_{oH}^{(f)}$$

(8.11)
and for hidden layer, we get

$$\frac{\partial E^{(f)}}{\partial w_{hl}} = \sum_{t_h^{(f)} \in F_h} \frac{\partial t_h^{(f)}}{\partial w_{hl}} \frac{\partial E^{(f)}}{\partial t_h^{(f)}} = -\sum_{t_h^{(f)} \in F_h} \lambda_h^{(f)} \frac{\partial t_o^{(f)}}{\partial t_h^{(f)}} e_o^{(f)}.$$  

Define $e_h^{(f)} = \frac{\partial t_o^{(f)}}{\partial t_h^{(f)}} e_o^{(f)}$ (8.12)

Therefore

$$\frac{\partial E^{(f)}}{\partial w_{hl}} = -\sum_{t_h^{(f)} \in F_h} e_h^{(f)} \lambda_h^{(f)}$$ (8.13)

And the weight update rule is

$$\Delta w_{oH} = \eta e_o^{(f)} \lambda_{oH}^{(f)}$$ (8.14)

$$\Delta w_{hl} = \eta \sum_{t_h^{(f)} \in F_h} e_h^{(f)} \lambda_h^{(f)}$$ (8.15)

The hidden layer weight update has components for each firing event of hidden layer neuron. If we follow strategy of weight update at each firing event, then we can view the hidden layer update rule above as the cumulative effect of individual update for each firing event.

In general we can write the update rule as

$$\Delta w_{jH} = \eta e_j^{(f)} \lambda_j^{(f)}$$ (8.16)

with weight update enforced at each $t_j^{(f)}$ event.

Like SpikeProp, the neurons can go silent due to weight update. In case a neuron goes silent for a long period of time, we use the heuristic solution of selectively increasing weights associated with the dead neuron (boosting) [64]. Hair trigger situation, the condition when change is weight is large due to neuron barely reaching threshold, is also a real issue. We will next propose a robust adaptive learning rate in next subsection that will tackle hair trigger situation and ensure stability during learning as well.
8.2 The EvSpikePropR Algorithm

The event based learning method for learning spike train, using fixed learning rate $\eta$ throughout the learning process, also inherits the stability and convergence issue of SpikeProp as well as hair trigger problem. In this section, we will propose a variable learning rate based on normalization and dead zone on-off criterion which will curb aforementioned issues.

The robust adaptive learning rates for output and hidden layer are defined as

$$
\eta_o(p) = \begin{cases} 
\frac{\eta}{\rho_o(p)} & \text{if } \left( 1 - \frac{\eta}{\rho_o(p)} \right) e_o(p)^2 \geq \epsilon_m^2 \\
0 & \text{otherwise}
\end{cases} \quad (8.17)
$$

where

$$
\rho_o(p) = \mu \rho_o(p - 1) + \max \left( \left\| \lambda_{oH}^{(f)} \right\|^2, \bar{\rho} \right) \quad (8.18)
$$

is the normalization factor for output layer and

$$
\eta_H(p) = \begin{cases} 
\frac{\eta}{\rho_H(p)} & \text{if } \left( 1 - \sum_h \sum_{t_h^{(f)}} \frac{\eta}{\rho_H(p)} \left\| \frac{\partial t_o^{(f)}}{\partial t_h^{(f)}} \lambda_{H}^{(f)} \right\|^2 \right) e_o(p)^2 \geq \epsilon_m^2 \\
0 & \text{otherwise}
\end{cases} \quad (8.19)
$$

where

$$
\rho_H(p) = \mu \rho_H(p - 1) + \max \left( \sum_h \sum_{t_h^{(f)}} \left\| \frac{\partial t_o^{(f)}}{\partial t_h^{(f)}} \lambda_{H}^{(f)} \right\|^2, \bar{\rho} \right) \quad (8.20)
$$

is the normalization factor for hidden layer. Here $\bar{\rho} > 0$, $\mu \in (0, 1)$ & $\eta \in (0, 1)$. The normalization factors are of the same form as used in [105].

Note that from the definition of normalization factors $\rho_o(p)$ and $\rho_{oH}(p)$, the left hand term in dead zone switching condition is always greater than zero, i.e.

$$
\left( 1 - \eta \left\| \lambda_{oH}^{(f)} \right\|^2 / \rho_o(p) \right) > (1 - \eta) > 0 \quad \text{and}
$$
8.3 Weight Convergence Analysis of EvSpikePropR

The dead zone on-off condition in (8.17) and (8.19) ensures that the learning converges in the sense of weight convergence. The normalization factor on the other hand ensures that the normalized system is $L_2$ stable and the denormalized system is $L_\infty$ stable. We will elaborate on the weight convergence and robust stability in $L_2$ and $L_\infty$ space in next sections (cf. Section 8.3 and 8.4). The parameter $\bar{\rho}$ sets the minimum value of normalization factor, which means the upper bound on the learning rate is $\eta/\bar{\rho}$. $\mu$ on the other hand is a compromise between alertness and robustness of learning [105]. The term $\epsilon_m$ is the upper bound on overall disturbance to the system. Furthermore, the choice of normalization factors, (8.18) and (8.20), ensure that large change in weight due to $u'_j(t^{(f)}_j) \to 0$ is balanced by the resulting large normalization factor as well.

8.3 Weight Convergence Analysis of EvSpikePropR

The approach for weight convergence analysis of SpikePropR uses the same approach as in Chapter 7.7. We will present the error system formulation in the weight convergence analysis itself.

8.3.1 Output layer Weight Convergence Analysis

Consider

$$
e^{(f)}_o(p) = \tilde{t}^{(f)}_o - t^{(f)}_o + \epsilon^{\text{ext}}_o
$$

$$
= f \left( \tilde{w}^{T}_{oH} y_{oH}(\tilde{t}_H) \right) - f \left( w^{T}_{oH} y_{oH}(t_H) \right) + \epsilon^{\text{ext}}_o
$$

$$
= f \left( \tilde{w}^{T}_{oH} y_{oH}(\tilde{t}_H) \right) + f \left( \tilde{w}^{T}_{oH} y_{oH}(t_H) \right) - f \left( w^{T}_{oH} y_{oH}(t_H) \right) + \epsilon^{\text{ext}}_o
$$

$$
= -\tilde{e}^{(f)}_o(p) + \epsilon^{(f)}_o(p)
$$

(8.21)

Here

$$
\tilde{e}^{(f)}_o(p) = f \left( \tilde{w}^{T}_{oH} y_{oH}(t_H) \right) - f \left( \tilde{w}^{T}_{oH} y_{oH}(t_H) \right)
$$

(8.22)
is regression error due to improper output layer weight and

\[
\epsilon_o^{(f)}(p) = f(\bar{w}_{oH}^Ty_{oH}(t_H)) - f(\bar{w}_{oH}^Ty_{oH}(t_H)) + \epsilon_{\text{ext}}^{o}
\]

is the total disturbance to the system, consisting of internal modelling error in hidden layer weights and external disturbance to the system. Define a bound \(\epsilon_m > \|\epsilon_o^{(f)}(p)\|_\infty\) as maximum disturbance to a system.

Now, consider

\[
\dot{\epsilon}_o^{(f)}(p) = f(\bar{w}_{oH}^Ty_{oH}(t_H)) - f(\bar{w}_{oH}^Ty_{oH}(t_H))
= (w_{oH} - \bar{w}_{oH})^T \frac{\partial t_o^{(f)}}{\partial u_o(t_o)} \left[ \frac{\partial u_o(t_o)}{\partial w_{oH}} + \frac{\partial t_o^{(f)}}{\partial w_{oH}} \right] w_{oH} = \omega
\]

where \(\omega = (1 - c)w_{oH} + c \bar{w}_{oH}, \ c \in (0, 1)\)

\[
= (w_{oH} - \bar{w}_{oH})^T \left( -\frac{1}{u'_o(t_o)} \right) \left[ y_{oH} - \lambda_{oH}^{(f-1)} y'_o(t_o) - \lambda_{oH}^{(f-1)} \right] w_{oH} = \omega
\]

\[
= (w_{oH} - \bar{w}_{oH})^T \lambda_{oH}^{(f)}
\]

Define the difference between current and ideal weight in iteration \(p\) by \(\tilde{w}_{oH}(p) = w_{oH}(p) - \bar{w}_{oH}\) and a Lyapunov function as

\[
\Delta V_o = \|\tilde{w}_{oH}(p + 1)\|^2 - \|\tilde{w}_{oH}(p)\|^2
\]

**Theorem 8.1.** If the output layer of the SNN is trained using EvSpikePropR with the output layer learning rate defined in (8.17), the weight \(w_{oH}(p)\) is convergent in the sense of Lyapunov function \(\Delta V_o \leq 0\).

**Proof.** Consider the Lyapunov function for an individual output neuron

\[
\Delta V_o = \|\tilde{w}_{oH}(p + 1)\|^2 - \|\tilde{w}_{oH}(p)\|^2
= \|w_{oH}(p + 1) - \bar{w}_{oH}\|^2 - \|w_{oH}(p) - \bar{w}_{oH}\|^2
= (w_{oH}(p + 1) - \bar{w}_{oH})^T (w_{oH}(p + 1) - \bar{w}_{oH})
- (w_{oH}(p) - \bar{w}_{oH})^T (w_{oH}(p) - \bar{w}_{oH})
= w_{oH}(p + 1)^T w_{oH}(p + 1) - 2\bar{w}_{oH}^T w_{oH}(p + 1)
- w_{oH}(p)^T w_{oH}(p) + 2\bar{w}_{oH}^T w_{oH}(p)
\]
by using (8.16), we get

\[
\Delta V_o = (\mathbf{w}_{oH}(p) + \eta_o(p) e_o^{(f)}(p) \lambda_{oH}^{(f)})^T \mathbf{w}_{oH}(p) + \eta_o(p) e_o^{(f)}(p) \lambda_{oH}^{(f)}
\]

\[
- 2\eta_o(p) e_o^{(f)}(p) \tilde{\mathbf{w}}_{oH}^T \lambda_{oH}^{(f)} - \mathbf{w}_{oH}(p)^T \mathbf{w}_{oH}(p)
\]

\[
= \eta_o(p)^2 e_o^{(f)}(p)^2 \left\| \lambda_{oH}^{(f)} \right\|^2 + 2\eta_o(p) e_o^{(f)}(p) (\mathbf{w}_{oH}(p) - \tilde{\mathbf{w}}_{oH})^T \lambda_{oH}^{(f)}
\]

by using the definition of regressor \( \tilde{e}_o^{(f)}(p) \)

\[
\Delta V_o = \eta_o(p)^2 e_o^{(f)}(p)^2 \left\| \lambda_{oH}^{(f)} \right\|^2 + 2\eta_o(p) e_o^{(f)}(p) \tilde{e}_o^{(f)}(p)
\]

(8.26)

replacing \( \tilde{e}_o^{(f)}(p) = e_o^{(f)}(p) - e_o^{(f)}(p) \) using equation (8.21)

\[
\Delta V_o = \eta_o(p)^2 e_o^{(f)}(p)^2 \left\| \lambda_{oH}^{(f)} \right\|^2 + 2\eta_o(p) (e_o^{(f)}(p) e_o^{(f)}(p) - e_o^{(f)}(p)^2)
\]

\[
= \eta_o(p)^2 e_o^{(f)}(p)^2 \left\| \lambda_{oH}^{(f)} \right\|^2 + 2\eta_o(p) e_o^{(f)}(p) e_o^{(f)}(p) - 2\eta_o(p) e_o^{(f)}(p)^2
\]

\[
\leq \eta_o(p)^2 e_o^{(f)}(p)^2 \left\| \lambda_{oH}^{(f)} \right\|^2 + \eta_o(p) e_o^{(f)}(p)^2 + \eta_o(p) e_o^{(f)}(p)^2 - 2\eta_o(p) e_o^{(f)}(p)^2
\]

\[
= -\eta_o(p) \left[ -e_o^{(f)}(p)^2 + \left( 1 - \frac{\eta}{\rho_o(p)} \left\| \lambda_{oH}^{(f)} \right\|^2 \right) e_o^{(f)}(p)^2 \right]
\]

\[
\leq -\eta_o(p) \left[ -e_o^{(f)}(p)^2 + \left( 1 - \frac{\eta}{\rho_o(p)} \left\| \lambda_{oH}^{(f)} \right\|^2 \right) e_o^{(f)}(p)^2 \right] \leq 0
\]

(8.27)

The choice of learning rate in (8.17) ensures that for non-zero \( \eta_o(p) \), the right hand side in the above expression is less than zero. For zero learning rate, the right hand side is trivially zero. Further, the choice of \( \rho_o(p) \) ensures that

\[
\left( 1 - \frac{\eta}{\rho_o(p)} \left\| \lambda_{oH}^{(f)} \right\|^2 \right) \geq 0.
\]

This completes the proof. \( \square \)

8.3.2 Hidden layer Weight Convergence Analysis

Consider the system error again

\[
e_o^{(f)}(p) = \tilde{e}_o^{(f)}(p) - t_o^{(f)} + e_o^{ext}
\]

\[
= f(\tilde{\mathbf{w}}_{oH}^T \mathbf{y}_{oH}(\mathbf{f}_m(\mathbf{W}_{HI} \mathbf{Y}_{HI}(\mathbf{t}_I)^T)))
\]

\[
- f(\mathbf{w}_{oH}^T \mathbf{y}_{oH}(\mathbf{f}_m(\mathbf{W}_{HI} \mathbf{Y}_{HI}(\mathbf{t}_I)^T))) + e_o^{ext}
\]

\[
= f(\mathbf{w}_{oH}^T \mathbf{y}_{oH}(\mathbf{f}_m(\mathbf{W}_{HI} \mathbf{Y}_{HI}(\mathbf{t}_I)^T)))
\]
Here, we suppose linearization around firing time. Then

\[
- f \left( w_{oH}^T y_{oH} (f_m(W_{HI} Y_{HI}(t_I))^T) \right)
+ f \left( \bar{w}_{oH}^T y_{oH} (f_m(W_{HI} Y_{HI}(t_I))^T) \right)
- f \left( w_{oH}^T y_{oH} (f_m(W_{HI} Y_{HI}(t_I))^T) \right) + \epsilon^{ext}
= -\epsilon^{(f)}_{oH}(p) + \epsilon^{(f)}_{oH}(p) \tag{8.28}
\]

where

\[
\epsilon^{(f)}_{oH}(p) = f \left( w_{oH}^T y_{oH} (f_m(W_{HI} Y_{HI}(t_I))^T) \right)
- f \left( w_{oH}^T y_{oH} (f_m(W_{HI} Y_{HI}(t_I))^T) \right) \tag{8.29}
\]

is regression error due to improper hidden layer weight estimation and

\[
\epsilon^{(f)}_{oH}(p) = f \left( w_{oH}^T y_{oH} (f_m(W_{HI} Y_{HI}(t_I))^T) \right)
- f \left( w_{oH}^T y_{oH} (f_m(W_{HI} Y_{HI}(t_I))^T) \right) + \epsilon^{ext} \tag{8.30}
\]

is system disturbance due to improper output layer weights and external disturbance as well. Define a bound \( \epsilon_m > \| \epsilon^{(f)}_{oH}(p) \|_\infty \) as maximum disturbance to a system.

Here, we suppose linearization around firing time. Then

\[
\epsilon^{(f)}_{oH}(p) \approx \left( \frac{\partial t_{o}^{(f)}}{\partial t_{H}} \right)^T \left( f_m(W_{HI} Y_{HI}(t_I))^T) - f_m(W_{HI} Y_{HI}(t_I))^T) \right)
= \sum_h \sum_{t_h^{(f)} \in F_h} \frac{\partial t_{o}^{(f)}}{\partial t_{H}} (f(w_{hi}^T y_{hi}(t_h^{(f)})) - f(\bar{w}_{hi}^T y_{hi}(t_h^{(f)})))
= \sum_h \sum_{t_h^{(f)} \in F_h} \frac{\partial t_{o}^{(f)}}{\partial t_{H}} (w_{hi} - \bar{w}_{hi})^T \frac{\partial t_{h}^{(f)}}{\partial t_{h}(t_h^{(f)})}
\left[ \frac{\partial u_{hi}(t_h^{(f)})}{\partial w_{hi}} + \frac{\partial t_{h}^{(f-1)}}{\partial w_{hi}} \frac{\partial u_{hi}(t_h^{(f)})}{\partial t_{h}(t_h^{(f-1)})} \right]_{w_{hi}=\omega}
\]

where \( (1-c)w_{hi} + cw_{hi}, \ c \in (0, 1) \)

\[
= \sum_h \sum_{t_h^{(f)} \in F_h} \frac{\partial t_{o}^{(f)}}{\partial t_{H}} (w_{hi} - \bar{w}_{hi})^T \left( -\frac{1}{u_{hi}(t_h^{(f)})} \right)
\left[ y_{hi} - \lambda_{hi}^{(f-1)} \nu^{(f)}(t_h^{(f)} - t_h^{(f-1)}) \right]_{w_{hi}=\omega}
\]
8.3. Weight Convergence Analysis of EvSpikePropR

\[
\Delta V_H = \sum_h \sum_{t'(n) \in F_h} (f_{t'(n)}(p) - \bar{w}_{hI}(p))_T \lambda_{hI}^{(f)} \tag{8.31}
\]

Define the difference between current and ideal weight in iteration \( p \) by \( \tilde{w}_{hI}(p) = w_{hI}(p) - \bar{w}_{hI} \) and a Lyapunov function as

\[
\Delta V_H = \sum_h \sum_{t'(n) \in F_h} \|	ilde{w}_{hI}(p+1)\|^2 - \|	ilde{w}_{hI}(p)\|^2 \tag{8.32}
\]

**Theorem 8.2.** If the hidden layer of the SNN is trained using EvSpikePropR with the hidden layer learning rate defined in (8.19), the weight \( W_{HI}(p) \) is convergent in the sense of Lyapunov function \( \Delta V_H \leq 0 \).

**Proof.** Consider the Lyapunov function for an individual hidden neuron

\[
\Delta V_h = \|	ilde{w}_{hI}(p+1)\|^2 - \|	ilde{w}_{hI}(p)\|^2 \\
= \|w_{hI}(p+1) - \tilde{w}_{hI}\|^2 - \|w_{hI}(p) - \tilde{w}_{hI}\|^2 \\
= (w_{hI}(p+1) - \tilde{w}_{hI})^T (w_{hI}(p+1) - \tilde{w}_{hI}) \\
= (\tilde{w}_{hI}(p) - \bar{w}_{hI})^T (w_{hI}(p) - \bar{w}_{hI}) \\
= w_{hI}(p+1)^T w_{hI}(p+1) - 2w_{hI}(p+1)^T \bar{w}_{hI} \\
- w_{hI}(p)^T w_{hI}(p) + 2w_{hI}(p)^T \bar{w}_{hI}
\]

by using (8.16), we get

\[
\Delta V_h = (w_{hI}(p) + \eta_H(p) e^{(f)}(p) \lambda_{hI}^{(f)}(p) \lambda_{hI}(p))^T (w_{hI}(p) + \eta_H(p) e^{(f)}(p) \lambda_{hI}(p)) \\
- 2\eta_H(p) e^{(f)}(p) \bar{w}_{hI}^T \lambda_{hI}^{(f)} - w_{hI}(p)^T w_{hI}(p) \\
= \eta_H(p)^2 e^{(f)}(p)^2 \| \lambda_{hI}^{(f)} \|^2 + 2\eta_H(p) e^{(f)}(p) (w_{hI} - \bar{w}_{hI})^T \lambda_{hI}^{(f)}
\]

by using the definition of (8.12), we get

\[
\Delta V_h = \eta_H(p)^2 \left\{ \frac{\partial t^{(f)}_o}{\partial t^{(f)}_h} \right\}^2 e^{(f)}(p)^2 \| \lambda_{hI}^{(f)} \|^2 \\
+ 2\eta_H(p) e^{(f)}(p) \frac{\partial t^{(f)}_o}{\partial t^{(f)}_h} (w_{hI} - \bar{w}_{hI})^T \lambda_{hI}^{(f)} \tag{8.33}
\]

Now, summing \( \Delta V_h \) over all hidden layer neurons and their corresponding firing
times, we get

$$\Delta V_H = \sum_h \sum_{i_h^{(f)} \in F_h} \Delta V_h$$

$$= \eta_H(p)^2 e_o^{(f)}(p)^2 \sum_h \sum_{i_h^{(f)} \in F_h} \left( \frac{\partial t_o^{(f)}}{\partial t_h^{(f)}} \right)^2 \| \lambda_{hI}^{(f)} \|^2$$

$$+ 2\eta_H(p)e_o^{(f)}(p) \sum_h \sum_{i_h^{(f)} \in F_h} \frac{\partial t_o^{(f)}}{\partial t_h^{(f)}} (w_{hI} - w_{hI})^T \lambda_{hI}^{(f)}$$

from (8.31)

$$\Delta V_H = \eta_H(p)^2 e_o^{(f)}(p)^2 \sum_h \sum_{i_h^{(f)} \in F_h} \left( \frac{\partial t_o^{(f)}}{\partial t_h^{(f)}} \right)^2 \| \lambda_{hI}^{(f)} \|^2 + 2\eta_H(p)e_o^{(f)}(p)e_{oIH}^{(f)}(p)$$

(8.34)

replacing $e_{o|H}^{(f)}(p) = e^{(f)}_{o|H}(p) - e^{(f)}_O(p)$ using equation (8.28)

$$\Delta V_H = \eta_H(p)^2 e_o^{(f)}(p)^2 \sum_h \sum_{i_h^{(f)} \in F_h} \left( \frac{\partial t_o^{(f)}}{\partial t_h^{(f)}} \right)^2 \| \lambda_{hI}^{(f)} \|^2$$

$$+ 2\eta_H(p)e_o^{(f)}(p)e_{o|H}^{(f)}(p) - 2\eta_H(p)e_o^{(f)}(p)^2$$

$$\leq \eta_H(p)^2 e_o^{(f)}(p)^2 \sum_h \sum_{i_h^{(f)} \in F_h} \left( \frac{\partial t_o^{(f)}}{\partial t_h^{(f)}} \right)^2 \| \lambda_{hI}^{(f)} \|^2$$

$$+ \eta_H(p)e_{o|H}^{(f)}(p)^2 - \eta_H(p)e_o^{(f)}(p)^2$$

$$\leq -\eta_H(p) \left[ -\epsilon^2_m + \left( 1 - \sum_h \sum_{i_h^{(f)} \in F_h} \frac{\eta}{\rho_H(p)} \left\| \frac{\partial t_o^{(f)}}{\partial t_h^{(f)}} \lambda_{hI}^{(f)} \right\|^2 \right] e_o^{(f)}(p)^2 \right] \leq 0$$

(8.35)

The choice of learning rate in (8.19) ensures that for non-zero $\eta_H(p)$, the right hand side in the above expression is less than zero. For zero learning rate, the right hand side is trivially zero. Further, the choice of $\rho_H(p)$ ensures that

$$\left( 1 - \sum_h \sum_{i_h^{(f)} \in F_h} \frac{\eta}{\rho_H(p)} \left\| \frac{\partial t_o^{(f)}}{\partial t_h^{(f)}} \lambda_{hI}^{(f)} \right\|^2 \right) \geq 0.$$ 

This completes the proof. \hfill \square
8.4 Robust Stability Analysis of EvSpikePropR

Now we will use conic sector theorem from input-output theory (cf. Chapter 4) for stability analysis. The functional analysis based input-output stability theory makes minimal assumptions about the process under consideration is suitable to investigate the robust stability of this algorithm.

The error and disturbance signals $e_m(p) = e_o^{(f)}(p)$, $\dot{e}_m(p) = \dot{e}_o^{(f)}(p)$ or $\dot{e}_{o|H}^{(f)}(p)$ and $\epsilon_m(p) = e_o^{(f)}(p)$ or $e_{o|H}^{(f)}(p)$ form a feedback system for a parameter adaptation algorithm as follows

$$e_m(p) = e_m(p) - r_m(p) = -H_{2,m} \dot{e}_m(p) + e_m(p) \quad (8.36a)$$

$$\dot{e}_m(p) = H_{1,m} e_m(p) \quad (8.36b)$$

$$r_m(p) = H_{2,m} \dot{e}_m(p) \quad (8.36c)$$

with $H_{1,m}$, $H_{2,m} : L_{2e} \to L_{2e}$ and $e_m(p)$, $\dot{e}_m(p)$, $r_m(p)$, $\epsilon_m(p) \in L_{2e}$. The operator $H_{1,m}$ represents the EvSpikePropR learning algorithm and (8.36a) results from (8.21) and (8.28) with operator $H_{2,m} = 1$, i.e. $r_m(p) = \dot{e}_m(p)$. The feedback operator $H_{2,m} = 1$ because there is no linear controller in the weight update of EvSpikePropR. Figure 8.2 shows the closed loop feedback system.

**Theorem 8.3.** If the SNN is trained using EvSpikePropR adaptive learning rates defined in (8.17) and (8.19) and the disturbances $\epsilon_o(p)$, $\epsilon_{o|H}(p) \in L_2$, then the normalized signals $\dot{e}_o^{(f)}(p)$, $e_o^{(f)}(p)$, $\dot{e}_{o|H}^{(f)}(p)$, $e_o^{n(f)}(p) \in L_2$.

**Proof.** The proof consists of two parts: for output layer learning error system and hidden layer learning error system respectively. We temporarily assume nonzero learning rates only as the system is stable whenever learning rate is zero.
From the definition of norm, \( \epsilon_o^{(f)}(p) \in L_2 \implies \epsilon_n^{(f)}(p) \in L_2 \). From (8.26)

\[
\Delta V_o = \eta_o(p)^2 \epsilon_o^{(f)}(p)^2 \left\| \lambda_{oH}^{(f)} \right\|^2 + 2 \eta_o(p) \epsilon_o^{(f)}(p) \dot{\epsilon}_o^{(f)}(p)
\]

\[
= 2 \eta_o(p) \left[ \frac{\eta}{2} \left\| \lambda_{oH}^{(f)} \right\|^2 \epsilon_o^{(f)}(p)^2 + \epsilon_o^{(f)}(p) \dot{\epsilon}_o^{(f)}(p) \right]
\]

\[
\leq 2 \frac{\eta}{\rho_o(p)} \left[ \eta \epsilon_o^{(f)}(p)^2 + \epsilon_o^{(f)}(p) \dot{\epsilon}_o^{(f)}(p) \right]
\]

Using definition of normalized signals (cf Chapter 4.1.6) and summing from \( p = 0 \) to \( N \), we get

\[
\sum_{p=0}^{N} \left[ \frac{\eta}{2} \epsilon_o^{n(f)}(p)^2 + \epsilon_o^{n(f)}(p) \dot{\epsilon}_o^{n(f)}(p) \right]
\]

\[
\geq \frac{1}{2\eta} \sum_{p=0}^{N} \left[ \left\| \tilde{w}_{oH}(p+1) \right\|^2 - \left\| \tilde{w}_{oH}(p) \right\|^2 \right]
\]

\[
\geq -\frac{1}{2\eta} \left\| \tilde{w}_{oH}(0) \right\|^2_F = -\gamma^o
\]

where

\[
\gamma^o = \frac{1}{2\eta} \left\| \tilde{w}_{oH}(0) \right\|^2_F \geq 0.
\]

Therefore, condition (4.2a) of Theorem 4.3 is satisfied. For \( H_2^n = H_2 = 1 \), condition (4.2b) implies

\[
-\frac{2 - \eta}{2} \left\| \epsilon_n^{(f)}(p) \right\|^2_N \leq -2\omega \left\| \epsilon_n^{(f)}(p) \right\|^2_N
\]

which is true for arbitrary \( \omega \leq \frac{(2-\eta)}{4} \). Therefore, from Theorem 4.3, \( \epsilon_n^{(f)}(p) \in L_2 \) and from the feedback system, \( \epsilon_o^{(f)}(p) \in L_2 \).

Similarly for hidden layer learning error signals, \( \epsilon_{oH}^{(f)}(p) \in L_2 \implies \epsilon_{oH}^{n(f)}(p) \in L_2 \). From (8.34)

\[
\Delta V_H = \eta_H(p)^2 \epsilon_o^{(f)}(p)^2 \sum_h \sum_{t_h^{(f)} \in \mathcal{F}_h} \left\| \frac{\partial t_h^{(f)}}{\partial h^{(f)}} \right\|^2 \left\| \lambda_{H_h}^{(f)} \right\|^2
\]

\[
+ 2 \eta_H(p) \epsilon_o^{(f)}(p) \dot{\epsilon}_{oH}^{(f)}(p)
\]

\[
\leq 2 \frac{\eta}{\rho_H(p)} \left[ \frac{\eta}{2} \epsilon_o^{(f)}(p)^2 + \epsilon_o^{(f)}(p) \dot{\epsilon}_{oH}^{(f)}(p) \right]
\]

(8.41)
Using definition of normalized signals (cf Chapter 4.1.6), summing from \( p = 0 \) to \( N \), and proceeding similarly as for output layer, we obtain

\[
\sum_{p=0}^{N} \left[ \frac{\eta}{2} e_{o}^{n(f)}(p)^2 + e_{o}^{n(f)}(p) \bar{e}_{o|H}^{n}(p) \right] \geq -\gamma^H
\]  

(8.42)

where

\[
\gamma^H = \frac{1}{2\eta} \sum_h \| \bar{w}_{hI}(0) \|^2 \geq 0.
\]  

(8.43)

This satisfies condition (4.2a) of Theorem 4.3. For \( H_2^n = H_2 = 1 \), condition (4.2b) implies

\[
-\frac{2 - \eta}{2} \| \bar{e}_{o}^{n(f)}(p) \|_N^2 \leq -2\omega \| \bar{e}_{o}^{n(f)}(p) \|_N^2.
\]  

(8.44)

We can have arbitrary \( \omega \leq \frac{(2-\eta)}{4} \) that satisfies the condition above. Therefore, from Theorem 4.3, \( \bar{e}_{o}^{n(f)}(p) \in L_2 \) and consequently from feedback system, \( e_{o}^{n(f)}(p) \in L_2 \). This completes the proof.

Now we will relax the stability conditions by extending \( L_2 \) stability result in \( L_\infty \) space.

**Theorem 8.4.** If the SNN is trained using EvSpikePropR adaptive learning rates defined in (8.17) and (8.19) and the disturbances \( e_{o}^{n(f)}(p),\ \bar{e}_{o|H}^{n}(p) \in L_\infty \), then, the signals \( \bar{e}_{o}^{n(f)}(p), \ e_{o}^{n(f)}(p), \ e_{o}^{n(f)}(p) \in L_\infty \).

**Proof.** For output layer learning error signals, consider the normalized exponentially weighted signals \( e_{o}^{n(f)}(p)^\beta,\ \bar{e}_{o|H}^{n(f)}(p)^\beta \) and \( e_{o}^{n(f)}(p)^\beta \). Following similarly as in Theorem 8.3 and transforming the error signals into exponentially weighted signals, we can show the \( L_2 \) stability of the normalized exponentially weighted signals as well i.e. \( (H_{1,o}^n)^\beta + \sigma/2 \), where \( (H_{1,o}^n)^\beta = \beta^p H_{1,o}^n \beta^{-p} \), is passive (condition (4.2a)) and \( (H_{2,o}^n)^\beta = \beta^p H_{2,o}^n \beta^{-p} = 1 \) is strictly inside \( \text{Cone}(\eta^{-1}, \eta^{-1}) \) (condition (4.2b)) for adaptive learning rates defined in (8.17) and (8.19). Also note that \( \| \bar{e}_{o}^{n(f)}(p) \|_\infty \leq \| e_{o}^{n(f)}(p) \|_\infty / \sqrt{p} \) from (8.18).

Now, from Theorem 4.7, \( L_\infty \) stability of \( \bar{e}_{o}^{f}(p) \) is guaranteed with \( \beta^2 = \mu^{-1} \). With \( e_{o}^{f}(p) \in L_\infty \), it follows from the feedback system that \( e_{o}^{f}(p) \in L_\infty \). This proves the stability for output layer.
Now following similar procedure for hidden layer error system, and using Theorem 4.7, the result follows for hidden layer learning errors.

The only requirement for $L_\infty$ boundedness is the boundedness of the disturbance $\epsilon_i(p)$. From (8.23) and (8.30), we can see that $\epsilon_o$ and $\epsilon_{oj}^H$ consist of internal modelling disturbance and external disturbance. It might be possible to completely eliminate internal modelling disturbance after some iterations, however, it is not always possible to have zero external disturbance in practical situation. Therefore, this $L_\infty$ extension of stability result is very useful.

8.5 Performance Comparison of EvSpikePropR

Now, we will compare the performance of MultiReSuMe, EvSpikePropR and EvSpikePropR. SpikePropMulti has similar formulation as EvSpikePropR except how the weight update is scheduled and how multiple target spikes are handled, therefore, it is not included in the simulation. We will compare these methods based on results of simulations on four benchmark datasets: XOR, Fisher’s Iris data, Wisconsin Breast Cancer data [110] and a synthetic Poisson spike data (c.f. 8.5.4). As discussed in Chapter 3.7, we will compare the simulation results based on their convergence rate (cf. Chapter 3.7.2), average number of epochs required to reach a tolerance value (cf. Chapter 3.7.3), training and testing accuracy (cf. Chapter 3.7.4) and average learning curve (cf. Chapter 3.7.5).

The SNN architecture used is a three layer architecture as described in Chapter 3.1. The weights are initialized randomly based on normal distribution using the method described in Chapter 3.3.2. We will now present the simulation results on each of the datasets and discuss the performance results.

8.5.1 XOR problem

The simulation parameters for XOR classification are listed below.
Network Architecture \(|I| = 3, |H| = 5, |O| = 1 \& |K| = 16^1\)
Synaptic Delays \(1 : 1 : 16\)
SRM time constant \(\tau = 30\)
Activation Threshold \(\vartheta = 1\)
Simulation Interval \(0 : 0.01 : 25\) units
Weight Initialization \(t_{\min} = 0.1\) units \& \(t_{\max} = 10\) units
Dead Neuron Boosting boost by 0.05

The input values are encoded as firing times at \(t = 6\) units and 0 units after the starting spike to represent ‘0’ and ‘1’ respectively. The desired firing times are represented by spike after 16 units and 10 units after the starting spike to represent ‘0’ and ‘1’ respectively. These spikes were arranged in a continuous spike train for each training sample and fed to the training system in continuous manner.

A total of 1000 different instances of simulations with up to 5000 epochs were evaluated for comparison. The convergence rate is plotted in figure 8.3. We can see that EvSpikePropR clearly shows better convergence rate. In figure 8.4,

![Figure 8.3: Convergence Rate plot for learning XOR pattern using MultiReSuMe, EvSpikePropR and EvSpikePropR.](image)

the average number of epochs required to reach different maximum network error values is plotted. The lower limit of error bar indicates the minimum number of epochs required and the upper error bar indicates the standard deviation of the epochs required. It is clear that EvSpikePropR is the fastest. The average learning error from 1000 different trials is plotted in figure 8.5. We can see that EvSpikeProp clearly achieves smaller network error values. A typical spike raster plot depicting the network learning XOR spike pattern after each epoch is plotted.

---

1Results using \(|K| = 1\) are also presented and are mentioned when presented.
in figure 8.6. The green lines indicate the desired spike times. We can see that EvSpikePropR quickly locks on the target spike values and progressively refines on it. A typical trace of learning rate in each iteration for output layer and hidden layer during training with EvSpikePropR ($\eta = 0.1$, $\bar{\rho} = 0.5$, $\epsilon_m = 0.001$, $\mu = 0.1$) for XOR problem is shown in figure 8.7. The discontinuity in the plot indicates the dead zone rule in effect and the variable learning rate is clear from the plot.

We were also able to learn XOR pattern using single delayed synaptic connection, $|K| = 1$, instead of 16. 850 out of 1000 instances learned the pattern successfully using EvSpikePropR whereas 106 out of 1000 instances learned the pattern successfully using EvSpikeProp. In comparison, 552 out of 1000 instances learned the
8.5. Performance Comparison of EvSpikePropR

Figure 8.6: Spike Raster plot after each epoch during XOR learning using Multi-ReSuMe, EvSpikePropR and EvSpikePropR.

Figure 8.7: Trace of output and hidden layer learning rate in a typical XOR learning using EvSpikePropR ($\eta = 0.1$, $\bar{\rho} = 0.5$, $\epsilon_m = 0.001$, $\mu = 0.1$).

8.5.2 Fisher’s Iris classification

The simulation parameters for Fisher’s Iris classification are listed below.
Network Architecture $|I| = 16$, $|H| = 7$, $|O| = 1$ & $|K| = 16$

Synaptic Delays $1 : 1 : 16$

SRM time constant $\tau_s = 30$

Activation Threshold $\vartheta = 1$

Simulation Interval $0 : 0.01 : 25$ units

Weight Initialization $t_{\text{min}} = 0.1$ units & $t_{\text{max}} = 10$ units

Dead Neuron Boosting boost by 0.05

For input encoding, we used population encoding (cf. Chapter 2.3.1) with 4 Gaussian receptive fields and arranged the input spikes in spike train corresponding to each training sample. The output spikes were encoded to fire after $t = 5$ units, 10 units & 15 units after first input spike for each of three different classes.

We performed 100 independent simulations with up to 1500 epochs. The plot of convergence rate for different methods during Fisher’s Iris learning is shown in figure 8.8. It is clear that EvSpikePropR demonstrates the best convergence rate.

The mean number of epochs required to reach different network error values is plotted in figure 8.9. The lower point in error bar denotes the minimum number of epoch required and the upper length of the error bar indicates the standard deviation of the epochs. We can see that EvSpikePropR is the fastest among all methods. One must not be confused by less number of epochs required by EvSpikeProp to reach tolerance value of 0.032 because only 2 out of 100 instances reached the tolerance value within 1500 epochs. Average learning plots over 100 instances of Fisher’s Iris Learning is shown in figure 8.10. EvSpikeProp achieves lowest average learning network error among the bunch. A typical spike raster
8.5. Performance Comparison of EvSpikePropR

![Bar chart](image)

Figure 8.9: Average number of epochs required to learn Fisher’s Iris data using MultiReSuMe, EvSpikePropR and EvSpikePropR.

![Line chart](image)

Figure 8.10: Average learning curves during Fisher’s Iris classification using MultiReSuMe, EvSpikePropR and EvSpikePropR.

plot for a learning sample for Fisher spike pattern after each epoch is plotted in figure 8.11. The green line indicates the desired spike time. We can see that EvSpikePropR quickly locks on the target spike values and progressively refines on it. On the other hand, Multi-ReSuMe locks on the target initially but switches between spike at mid value for two classes. Figure 8.12 and figure 8.13 show the plot of training accuracy and testing accuracy respectively corresponding to different final network error values for all the methods. The upper error bar position indicates the maximum accuracy achieved and the length of lower error bar indicates the standard deviation in accuracy for both training and testing accuracy. EvSpikeProp demonstrates best and consistent training and testing accuracy as well. The trace of learning rate during EvSpikePropR training of
Ch. 8. Event based SpikeProp for Learning Spike Train

8.5. Performance Comparison of EvSpikePropR

Figure 8.11: Spike Raster plot after each epoch during Fisher’s Iris Learning using MultiReSuMe, EvSpikePropR and EvSpikePropR.

Figure 8.12: Training accuracy in Fisher’s Iris classification using MultiReSuMe, EvSpikePropR and EvSpikePropR.

Fisher’s Iris data is shown in figure 8.14 for output layer and hidden layer.

8.5.3 Wisconsin Breast Cancer classification

The simulation parameters for Wisconsin Breast Cancer classification problem are listed below.
8.5. Performance Comparison of EvSpikePropR

![Graph showing testing accuracy in Fisher's Iris classification using MultiReSuMe, EvSpikePropR and EvSpikePropR.](image)

Figure 8.13: Testing accuracy in Fisher’s Iris classification using MultiReSuMe, EvSpikePropR and EvSpikePropR.

![Graph showing trace of output and hidden layer learning rate in a typical Fisher’s Iris learning using EvSpikePropR (η = 0.1, ¯ρ = 0.1, ϵm = 0.1, µ = 0.5) for first 1000 iterations.](image)

Figure 8.14: Trace of output and hidden layer learning rate in a typical Fisher’s Iris learning using EvSpikePropR (η = 0.1, ¯ρ = 0.1, ϵm = 0.1, µ = 0.5) for first 1000 iterations.

**Network Architecture**

<table>
<thead>
<tr>
<th>I</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>15</td>
</tr>
<tr>
<td>O</td>
<td>2</td>
</tr>
<tr>
<td>K</td>
<td>16</td>
</tr>
</tbody>
</table>

**Synaptic Delays**

1 : 1 : 16

**SRM time constant**

τs = 30

**Activation Threshold**

ϑ = 1

**Simulation Interval**

0 : 0.01 : 25 units

**Weight Initialization**

t_{min} = 0.1 units & t_{max} = 10 units

**Dead Neuron Boosting**

boost by 0.05

The nine inputs were encoded between time t = 0 to 4 using Population encoding with 7 Gaussian receptive fields and the input spikes were arranged as spike train. The outputs were interpreted based on winner-takes-all scheme.
100 independent simulations were performed with up to 1000 epochs of learning. Comparison of convergence rate of all the methods is shown in figure 8.15 for Wisconsin Breast Cancer learning. We can see that EvSpikePropR show excellent convergence rate. The average number of epochs required to reach different minimum network error values is plotted in figure 8.16. The lower error bar corresponds to the minimum number of epoch and the length of upper error bar indicates the standard deviation of epochs. We can see that EvSpikePropR ($\eta = 0.1$, $\bar{\rho} = 0.1$, $\epsilon_m = 0.001$, $\mu = 0.05$) requires the least number of epoch. EvSpikePropR ($\eta = 0.032$, $\bar{\rho} = 0.1$, $\epsilon_m = 0.001$, $\mu = 0.05$) is quite close though.

The learning plots averaged over 100 different learning instances is plotted in figure 8.17. We can clearly see that EvSpikePropR ($\eta = 0.032$, $\bar{\rho} = 0.1$, $\epsilon_m = 0.001$, $\mu = 0.05$)
8.5. Performance Comparison of EvSpikePropR

The average learning curves during Wisconsin Breast Cancer classification using MultiReSuMe, EvSpikePropR and EvSpikePropR are depicted in Figure 8.17.

![Average learning curves](image)

Figure 8.17: Average learning curves during Wisconsin Breast Cancer classification using MultiReSuMe, EvSpikePropR and EvSpikePropR.

The training and testing accuracy achieved using different learning methods is plotted in Figure 8.19 and Figure 8.20 respectively. We can see that the learning curves of EvSpikeProp and ReSuMe are significantly better compared to the learning curves of EvSpikePropR. However, EvSpikePropR demonstrates better learning performance through intelligent use of learning rate. In figure 8.18, the spike raster plot of a learning sample class is plotted for each epoch during learning. The green line indicates the desired spike time. We can clearly see that EvSpikePropR quickly corrects the spike time towards the target without any oscillation.

![Spike Raster plot](image)

Figure 8.18: Spike Raster plot after each epoch during Wisconsin Breast Cancer Learning using MultiReSuMe, EvSpikePropR and EvSpikePropR.

μ = 0.05) shows the best performance with the average learning curve of EvSpikePropR (η = 0.1, ̄ρ = 0.1, ɛm = 0.001, μ = 0.05) also good. Compared to the learning curves of EvSpikeProp and ReSuMe, EvSpikePropR demonstrates significantly better learning performance through intelligent use of learning rate. In figure 8.18, the spike raster plot of a learning sample class is plotted for each epoch during learning. The green line indicates the desired spike time. We can clearly see that EvSpikePropR quickly corrects the spike time towards the target without any oscillation. The training and testing accuracy achieved using different learning methods is plotted in figure 8.19 and figure 8.20 respectively. We can see
8.5. Performance Comparison of EvSpikePropR

![Training Accuracy Diagram](image1)

**Figure 8.19:** Training accuracy in Wisconsin Breast Cancer classification using MultiReSuMe, EvSpikePropR and EvSpikePropR.

![Testing Accuracy Diagram](image2)

**Figure 8.20:** Testing accuracy in Wisconsin Breast Cancer classification using MultiReSuMe, EvSpikePropR and EvSpikePropR.

that the accuracy performance of EvSpikePropR and EvSpikeProp is very good and that of ReSuMe lagging behind. The trace of learning rate for an output neuron and a hidden neuron for first 1000 iterations using EvSpikePropR during Wisconsin Breast Cancer classification learning is plotted in figure 8.21. We can see the modulation of learning rate by the robust stability and dead zone rule.

### 8.5.4 Poisson Spike data classification

Poisson Spike data is a synthetic data in which three spike trains are generated using Poisson spike generation and the task is to distinguish between these spike train types. The first class consists of Poisson spike train with firing rate of 1.5, 0.3
8.5. Performance Comparison of EvSpikePropR

Figure 8.21: Trace of output and hidden layer learning rate in a typical Wisconsin Breast Cancer classification using EvSpikePropR ($\eta = 0.032$, $\bar{\rho} = 0.1$, $\epsilon_m = 0.001$, $\mu = 0.05$) for first 1000 iterations.

The second class consists of Poisson spike train with firing rate of 0.6, 0.1 & 1.5. The output is represented using spike trains of two distinct spike pattern with four and three spike for each of the classes. The simulation parameters for Poisson Spike data classification problem are listed below.

- **Network Architecture**: $|\mathcal{I}| = 3$, $|\mathcal{H}| = 30$, $|\mathcal{O}| = 1 \& |\mathcal{K}| = 2$
- **Synaptic Delays**: 1 : 1 : 2
- **SRM time constant**: $\tau_s = 30$
- **Activation Threshold**: $\vartheta = 1$
- **Simulation Interval**: 0 : 0.01 : 25 units
- **Weight Initialization**: $t_{\text{min}} = 0.1$ units & $t_{\text{max}} = 10$ units
- **Dead Neuron Boosting**: boost by 0.05

10 independent simulations were performed with up to 1000 epochs of learning. Comparison of convergence rate of all the methods is shown in figure 8.22 for Poisson Spike Data learning. We can see that EvSpikePropR show excellent convergence rate. In figure 8.23, the plot of mean number of epoch required to reach a tolerance value is plotted. The lower error bar corresponds to the minimum number of epoch and the length of upper error bar indicates the standard deviation of epochs. We can see that EvSpikePropR ($\eta = 0.032$, $\bar{\rho} = 1$, $\epsilon_m = 0.1$, $\mu = 0.5$) requires the least number of epoch. EvSpikePropR ($\eta = 0.01$, $\bar{\rho} = 1$, $\epsilon_m = 0.1$, $\mu = 0.5$) also has decent performance. Figure 8.24 shows the average learning curve during Poisson Spike Data learning averaged for 10 different trials. We can clearly see that EvSpikePropR ($\eta = 0.032$, $\bar{\rho} = 1$, $\epsilon_m = 0.1$, $\mu = 0.5$) shows...
EvSpikeProp, $\eta = 0.001$
Multi-ReSuMe, $A_+ = 1.1$, $A_- = 0.25$
EvSpikePropR, $\eta = 0.032$, $\bar{\rho} = 1$, $\epsilon_m = 0.1$, $\mu = 0.5$
EvSpikePropR, $\eta = 0.01$, $\bar{\rho} = 1$, $\epsilon_m = 0.1$, $\mu = 0.5$

**Figure 8.22:** Convergence Rate plot for learning Poisson spike data classification using MultiReSuMe, EvSpikePropR and EvSpikePropR.

EvSpikeProp learns the first three target spikes quickly, but struggles to correctly produce the fourth target spike. The solution diverges while trying to correct the fourth spike. EvSpikePropR on the other hand quickly latches to the first three target spikes and slowly adjusts for the fourth target spike. The spike raster plot for multi-ReSuMe shows erratic spiking behaviour with more spikes concentrated...
8.5. Performance Comparison of EvSpikePropR

![Figure 8.24: Average learning curves during Poisson spike data classification using MultiReSuMe, EvSpikePropR and EvSpikePropR.](image)

![Figure 8.25: Spike Raster plot after each epoch during Poisson spike data classification Learning using MultiReSuMe, EvSpikePropR and EvSpikePropR.](image)

around the target spike times. The training and testing accuracy achieved using different learning methods is plotted in figure 8.26 and figure 8.27 respectively. We can see that the accuracy performance of EvSpikePropR and EvSpikeProp good.

The trace of learning rate for an output neuron and a hidden neuron for first 1000 iterations using EvSpikePropR during Wisconsin Breast Cancer classification learning is plotted in figure 8.28. We can see the modulation of learning rate by the robust stability and dead zone rule. Note the discontinuities in the learning rate for zero learning rate.
8.6 Choice of EvSpikePropR Learning Parameter

There are four tunable parameters in EvSpikePropR learning: $\eta$, $\bar{\rho}$, $\mu$ and $\epsilon_m$. The results for different variation of these parameters is listed in Table 8.1. The results confirm that $\eta/\bar{\rho}$ effect the max learning rate and hence the speed of learning. $\mu$ controls the robustness vs activeness of learning and $\epsilon_m$ controls the allowable error in the system.
### Ch. 8. Event based SpikeProp for Learning Spike Train

#### 8.6. Choice of EvSpikePropR Learning Parameter

Table 8.1: Variation of learning parameters in EvSpikePropR.

<table>
<thead>
<tr>
<th></th>
<th>EvSpikePropR</th>
<th>Tol = 0.01</th>
<th>Tol = 0.0001</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>ρ</td>
<td>εm</td>
</tr>
<tr>
<td>XOR</td>
<td>0.1</td>
<td>0.5</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>0.056</td>
<td>0.5</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.01</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.5</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fisher</td>
<td>0.1</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.1</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>0.032</td>
<td>0.1</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>1</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.1</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>0.032</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>1</td>
<td>0.1</td>
</tr>
</tbody>
</table>
8.6. Choice of EvSpikePropR Learning Parameter

Figure 8.28: Trace of output and hidden layer learning rate in a typical Poisson spike data classification using EvSpikePropR ($\eta = 0.032$, $\bar{\rho} = 1$, $\epsilon_m = 0.1$, $\mu = 0.5$) for first 1000 iterations.
Chapter 9

Conclusion and Future Work

9.1 Conclusion

The non-linear and discontinuous nature of spiking neurons makes it vulnerable to stability and convergence issues. Firstly, there are not much learning algorithms for SNN, especially in a network with hidden layer. Secondly, the algorithms that exist face stability and convergence concerns. The goal in this thesis is to develop fast, stable and convergent methods to learn in an SNN with hidden layer i.e. multilayer SNN. In this thesis, several algorithms were proposed. The algorithms for SNN presented here focus on the convergence issues of the algorithms. In addition, robustness to external disturbances and stability of the learning method is a key concern in this thesis. The convergence study is presented here for single spike learning of weight and delay as well as spike train learning of weight parameter. The stability and robustness study is done for learning weight parameter for both single spike and multiple spike learning. The main results of this thesis are summarized below.

1. Weight convergence analysis of SpikeProp algorithm was performed in detail. Proof of convergence of weight parameter is shown and the implication is that the weights learned will move closer to, at least not move farther from, the ideal weight parameter. In Chapter 5, an adaptive learning rate based on weight convergence condition – SpikePropAd – is presented. The result of convergence condition is that the learning is much faster and the training converges successfully a lot of time compared to the existing methods. The learning accuracy results were also superior compared to the peers. The
weight convergence also curbed the issue of surges, resulting in none or minimal surges during learning and avoided hair trigger issue as well, both of which are a major concern in SpikeProp learning.

2. Delay convergence analysis of delay learning extension of SpikeProp was performed in detail in Chapter 6. An adaptive learning rate for delay adaptation based on delay convergence condition led to SpikePropAdDel extension of SpikeProp. In the simulations, better convergence and faster learning was observed compared to plain SpikeProp delay learning method. We were also able to learn the same benchmark problem with smaller network size, although, it took longer time to learn using SpikePropAdDel. The results, however, were not superior to SpikePropAd. Perhaps, it is because in SpikePropAdDel, there are double the tunable parameters because both weights and delays are being learned, as a result, there is additional degree of freedom for the learning algorithm to manage. Further, we were not able to achieve learning results with delay learning alone. Therefore, delay adaptation is always used in conjugation with weight adaptation in this thesis.

3. Along with weight convergence analysis, robustness analysis to external disturbance was performed in Chapter 7. The robustness analysis was performed with two different approach: the individual error analysis leading to SpikePropR algorithm and the total error analysis leading to SpikePropRT algorithm. Nevertheless, the stability of learning algorithm in $L_2$ space was shown and extended for stability in $L_\infty$ space. As we have mentioned numerous times, the extension of stability results in $L_\infty$ space is more practical because it requires the disturbance signal to be bounded, rather than the requirement of disturbance to die out to zero in $L_2$ stability. Both SpikePropR and SpikePropRT have minimal assumption of bounded disturbance signal in the learning system.

4. In simulation results, SpikePropR showed better convergence rate compared to SpikePropAd, SpikeProp and RProp method. However, sometimes it was slightly slower than SpikePropAd. Nevertheless with slight loss in speed, better convergence was achieved using SpikePropR. Furthermore, SpikePropRT showed even better convergence rate than SpikePropR method, especially for bigger problem set. The trade-off between convergence and speed of learning was also evident in SpikePropRT learning as well. However, it was faster and showed better convergence than the rest, barring XOR problem. This indicates that the total error approach to robust stability is more effective.
5. We proposed a new Event based weight update rule – EvSpikeProp – that can learn a spike train in a continuous manner in Chapter 8. This allows us to exploit the hidden layer dynamics as well as efficient information exchange using spike train, unlike the previous methods which were limited to single spike of the neuron. With the knowledge of convergence and robustness analysis of SpikeProp, convergence and robustness results were shown for EvSpikeProp leading to EvSpikePropR algorithm. Total error system analysis was chosen because it performed better for single spiking case. Indeed the simulation results show that EvSpikePropR has better performance compared to EvSpikeProp and MultiReSuMe learning. EvSpikeProp and EvSpikePropR also required less computation time as MultiReSuMe became more slower and slower with bigger network because of the inherent necessity to compare between each and every spike in a spike train. Furthermore, EvSpikePropR algorithm, being equipped with convergence and stability results, showed much better convergence results as well as much faster learning speed. The accuracy results were also good.

6. For XOR data, we were able to learn the pattern using single synaptic connection with no delay much consistently (85% of the time) with EvSpikePropR. It is very very difficult to learn even the XOR pattern in such a network using single spike learning, even with the efficient modifications to SpikeProp presented in this thesis. This shows the advantage of multiple spike in SNN learning scenario.

The technique of weight convergence used in this thesis has proven its usefulness to determine stable learning steps in SpikeProp based learning methods. Along with increasing the chances of successful learning, it also speeds up the learning process, achieving stable result much quickly. This is demonstrated via all the adaptive learning-rate based learning methods introduced in this thesis. It shows that weight convergence is a useful method to achieve not only convergent learning, but faster learning as well. In this thesis, the benefits of weight convergence method is seen in the case of supervised learning in multilayer SNNs. Nevertheless, it is a useful enhancement that can be applied to other similar backpropagation based learning methods in general. In addition, the robust stability analysis based on conic sector theory is a useful tool in analyzing robust stability of a learning system as well, as demonstrated in SpikePropR, SpikePropRT and EvSpikePropR learning methods in this thesis. The robust stability is useful in establishing stability of a learning method even in presence of external disturbance and internal disturbance.
These rigorous theoretical results also translate well in practical scenario as well.

For better generalization, we can easily add regularization term in the loss function and modify the learning rule accordingly. The modification is trivial. Early stopping using cross validation data is independent of the learning rule. It can easily be implemented for the learning methods in this thesis for better generalization performance.

Most of the methods dealt in this thesis are for learning the first spike of a SNN. It limits their application for learning in static (vector based) data only. The event based learning method, introduced in Chapter 8, overcomes this limitation of SpikeProp based methods and expands its horizon for learning in temporal data sequences. The work presented in Chapter 8 has been derived for a fully connected feedforward network. Nevertheless, it is straightforward to modify the learning rule of EvSpikeProp and EvSpikePropR for a spatially restricted interconnections between the layers for learning spatio-temporal patterns. In addition, these methods can be cascaded in front of a spiking neural network reservoir (similar to the one in reservoir computing [85–87]) or a structured interconnection between spiking neurons (similar to NeuCube [93]). The advantage is that we can use the flexibility and power of hidden layer dynamics, instead of a single neuron, to learn spike patterns as well.

Also, there is nothing to stop us from using the results for output layer neuron in EvSpikePropR for learning weights for a single neuron as well. It can, therefore, serve as a robust and convergent learning rule to learn weights for a single neuron. The event based learning in EvSpikeProp and EvSpikePropR allow us to handle learning for a continuous stream of input spikes and desired spikes indefinitely in a seamless manner. It is a distinct advantage from most of the batch mode spike train learning methods.

For all the methods introduced in this thesis, the neuron model is some variant of SRM. Modeling a SRM is more taxing than simpler LIF models of a spiking neuron. Therefore, it is less computationally intensive to use LIF models in bigger sized networks with large number of neuron and synapses. This is a limitation to SpikeProp and its variants like the ones discussed in this thesis that makes using them for bigger networks difficult to implement compared to the ones with LIF neurons. Although, a form of speed up for SRM neuron is presented in Algorithm 1 & 2, it is an issue in implementing our methods for bigger networks with thousands of neurons.
Also, since the learning algorithms presented in this thesis all involve hidden layer neurons, it is not always possible to interpret the weight parameters that are learned via these learning algorithms. Similar is the case for MLP and deep learning methods for second generation ANNs, especially for non image datasets. We just know that the knowledge is represented by the weight and delay parameter. The manner in which the parameters represent knowledge is unknown for these methods. These methods simply output desired outputs given the input spikes.

9.2 Future Work

In this section, we will discuss about the possible future research directions leading from the works of this thesis.

9.2.1 Robustness Analysis of Delay Learning Extension of SpikeProp

Delay learning is a crucial aspect of knowledge representation in Spiking Neural Network. Delay adaptation allows us to use smaller network and get rid of redundant connections. In addition it is more biologically plausible way of knowledge representation. As we discussed in Chapter 6, it is also important in the sense that it gives more computational power to SNN. Nevertheless, delay learning using SpikeProp faces the same stability and convergence issues faced by SpikeProp with weight adaptation only. The issues are in fact doubled because the tunable parameters also increases in delay learning scenario. In this thesis, we performed delay convergence analysis of delay learning extension of SpikeProp to resolve convergence issues. Robust stability of SpikePropDel is also a useful. We would like to pursue in this direction to propose more advanced robust delay learning extensions of SpikePropAd similar to SpikePropR and SpikePropRT for weight adaptation.

9.2.2 Delay Adaptation for Learning Spike Train

As far as we know, there is no delay learning algorithm for SpikeProp that can learn SpikeTrain in a multilayer setting. For single spiking neuron though, there is DI-ReSuMe [111]. Since delay learning is more biologically plausible, it is an attractive area of further research. We would like to extend EvSpikeProp and
EvSpikePropR to learn delay parameters in a multilayer SNN to learn spike train. Such a method would be a first of its kind.

9.2.3 Implementation in Neuromorphic Chips

Neuromorphic chips are the future of on-chip neural network implementation. They are very useful because such a neural network using spiking neuron are very energy efficient (cf. Chapter 1.1.3). Further, simulating a SNN in a CPU or GPU based system is very slow and time consuming. A neuromorphic implementation would allow us to train an SNN in large scale to solve useful tasks. Further, it would test the scalability of SNN learning algorithms and test their usefulness.

9.2.4 Spike Encoding and Decoding Principle

Spiking Neural Network have huge interest and potential as a computational system. However, real world signals are numeric in nature and spiking neurons receive and emit spike patterns. There have been significant efforts to provide a method of translating numeric inputs into spikes and back such as LNP, GLM etc. (cf. Chapter 2.3). Nevertheless, there is no single robust method that serves as the key to spike encoding and decoding. A well settled method to encode and decode spikes would be very useful to use SNN as a computational unit, especially for regression problems.
Appendix A

Matrix Calculus

In this Appendix, we will present some useful formulae of matrix calculus. We will follow denominator layout convention here. In denominator layout, the result of differential operator follows the dimension of denominator variable. It is also called Hessian formulation. Similar results are possible using numerator layout convention as well.

A.1 Derivatives of Vector Functions

Suppose $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ i.e.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad (A.1)$$

Further, $y = y(x) : \mathbb{R}^n \to \mathbb{R}^m$ is a function of $x$. 
A.1.1 Derivative of Vector with Respect to Vector

The derivative of $y$ with respect to $x$ is defined as

$$\frac{\partial y}{\partial x} \triangleq \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \frac{\partial y_2}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{n \times m} \quad (A.2)$$

This formulation is the transpose of Jacobian matrix.

A.1.2 Derivative of Scalar with Respect to Vector

The derivative of $y$ with respect to $x$ is defined as

$$\frac{\partial y}{\partial x} \triangleq \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n \quad (A.3)$$

A.1.3 Derivative of Vector with Respect to Scalar

The derivative of $y$ with respect to $x$ is defined as

$$\frac{\partial y}{\partial x} \triangleq \begin{bmatrix} \frac{\partial y_1}{\partial x} & \frac{\partial y_2}{\partial x} & \cdots & \frac{\partial y_m}{\partial x} \end{bmatrix} \in \mathbb{R}^{1 \times m} \quad (A.4)$$

A.1.4 Chain Rule for Vector Functions

Suppose $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ and $z \in \mathbb{R}^r$ i.e.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad \& \quad z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_r \end{bmatrix} \quad (A.5)$$
Further \( z = z(y) : \mathbb{R}^m \to \mathbb{R}^r \) and \( y = y(x) : \mathbb{R}^n \to \mathbb{R}^m \). For the elements \( z_i \) and \( x_j \), the derivative is

\[
\frac{\partial z_i}{\partial x_j} = \sum_{k=1}^{m} \frac{\partial y_k}{\partial x_j} \frac{\partial z_i}{\partial y_k} \quad \forall i = 1, 2, \ldots, r \text{ & } j = 1, 2, \ldots, n \tag{A.6}
\]

Then,

\[
\frac{\partial z}{\partial x} = \begin{bmatrix}
\frac{\partial z_1}{\partial x_1} & \frac{\partial z_2}{\partial x_1} & \cdots & \frac{\partial z_r}{\partial x_1} \\
\frac{\partial z_1}{\partial x_2} & \frac{\partial z_2}{\partial x_2} & \cdots & \frac{\partial z_r}{\partial x_2} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial z_1}{\partial x_n} & \frac{\partial z_2}{\partial x_n} & \cdots & \frac{\partial z_r}{\partial x_n}
\end{bmatrix} \in \mathbb{R}^{n \times r}
\]

\[
= \begin{bmatrix}
\sum_{k=1}^{m} \frac{\partial y_k}{\partial x_1} \frac{\partial z_1}{\partial y_k} & \sum_{k=1}^{m} \frac{\partial y_k}{\partial x_1} \frac{\partial z_2}{\partial y_k} & \cdots & \sum_{k=1}^{m} \frac{\partial y_k}{\partial x_1} \frac{\partial z_r}{\partial y_k} \\
\sum_{k=1}^{m} \frac{\partial y_k}{\partial x_2} \frac{\partial z_1}{\partial y_k} & \sum_{k=1}^{m} \frac{\partial y_k}{\partial x_2} \frac{\partial z_2}{\partial y_k} & \cdots & \sum_{k=1}^{m} \frac{\partial y_k}{\partial x_2} \frac{\partial z_r}{\partial y_k} \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{k=1}^{m} \frac{\partial y_k}{\partial x_n} \frac{\partial z_1}{\partial y_k} & \sum_{k=1}^{m} \frac{\partial y_k}{\partial x_n} \frac{\partial z_2}{\partial y_k} & \cdots & \sum_{k=1}^{m} \frac{\partial y_k}{\partial x_n} \frac{\partial z_r}{\partial y_k}
\end{bmatrix}
\]

\[
= \frac{\partial y}{\partial x} \frac{\partial z}{\partial y}
\]

(A.7)

Further, if \( w(z) : \mathbb{R}^r \to \mathbb{R}^p \), then

\[
\frac{\partial w}{\partial x} = \frac{\partial y}{\partial x} \frac{\partial z}{\partial y} \frac{\partial w}{\partial z} \quad \tag{A.8}
\]

Note the order of chain rule expansion is from right to left.

### A.1.5 Vector Derivative Formulae

The following table presents a collection of vector derivative formulae

<table>
<thead>
<tr>
<th>( y )</th>
<th>( \frac{\partial y}{\partial x} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Ax )</td>
<td>( A^\top )</td>
</tr>
<tr>
<td>( x^\top A )</td>
<td>( A )</td>
</tr>
<tr>
<td>( x^\top x )</td>
<td>( 2x )</td>
</tr>
<tr>
<td>( x^\top Ax )</td>
<td>( Ax + A^\top x )</td>
</tr>
</tbody>
</table>
A.2 Derivative of Matrix functions

We will present the definition of derivative of a matrix by a scalar and derivative of scalar by a matrix. For derivative of matrix by a vector, derivative of vector by a matrix and derivative of matrix by a matrix, there is no general consensus.

A.2.1 Derivative of Scalar by a Matrix

The derivative of $y$ with respect to $X \in \mathbb{R}^{m \times n}$ is defined as

$$\frac{\partial y}{\partial X} \triangleq \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{12}} & \cdots & \frac{\partial y}{\partial x_{1n}} \\ \frac{\partial y}{\partial x_{21}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{m1}} & \frac{\partial y}{\partial x_{m2}} & \cdots & \frac{\partial y}{\partial x_{mn}} \end{bmatrix} \in \mathbb{R}^{m \times n} \quad (A.9)$$

A.2.2 Derivative of Matrix by a Scalar

The derivative of $Y \in \mathbb{R}^{m \times n}$ with respect to $x$ is defined as

$$\frac{\partial Y}{\partial x} \triangleq \begin{bmatrix} \frac{\partial y_{11}}{\partial x} & \frac{\partial y_{21}}{\partial x} & \cdots & \frac{\partial y_{m1}}{\partial x} \\ \frac{\partial y_{12}}{\partial x} & \frac{\partial y_{22}}{\partial x} & \cdots & \frac{\partial y_{m2}}{\partial x} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{1n}}{\partial x} & \frac{\partial y_{2n}}{\partial x} & \cdots & \frac{\partial y_{mn}}{\partial x} \end{bmatrix} \in \mathbb{R}^{n \times m} \quad (A.10)$$

A.3 Derivative of Function of Matrix Product

The derivative of a scalar functional of diagonal of product of two matrices yields a special form. It is presented below.

§ A.1. Define $y = f(\text{diag}(AB)) = f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$.

where $A \in \mathbb{R}^{n \times m}$, $B \in \mathbb{R}^{m \times n}$ & $x = \text{diag}(AB) \in \mathbb{R}^n$

Then $\frac{\partial y}{\partial A} = \text{diag} \left( \frac{\partial y}{\partial x} \right) B^T \quad (A.11)$

Proof. Consider $i^{\text{th}}$ row and $i^{\text{th}}$ column of matrices $A$ and $B$ respectively and
denote them by \( a_i \) and \( b_i \). Then the \( i^{th} \) element of \( x \) is \( x_i = a_i \cdot b_i \). Now consider

\[
\frac{\partial y}{\partial a_i} = \frac{\partial f(x)}{\partial a_i} = \frac{\partial a_i \cdot b_i \cdot \partial f(x)}{\partial x_i} = b_i^T \frac{\partial y}{\partial x_i}
\]

Concatenating the individual components row wise, we get

\[
\frac{\partial y}{\partial A} = \begin{bmatrix}
\frac{\partial y}{\partial x_1} b_1^T \\
\vdots \\
\frac{\partial y}{\partial x_n} b_n^T
\end{bmatrix} = \begin{bmatrix}
\frac{\partial y}{\partial x_1} & 0 & 0 \\
0 & \ddots & 0 \\
0 & 0 & \frac{\partial y}{\partial x_n}
\end{bmatrix} \begin{bmatrix}
b_1^T \\
\vdots \\
b_n^T
\end{bmatrix} = \text{diag} \left( \frac{\partial y}{\partial x} \right) B^T
\]

\[\square\]
Appendix B

Discrete Bellman-Gronwall Lemma

§ B.1. For nonnegative sequences \( \langle x(n) \rangle, \langle f(n) \rangle \) and \( \langle g(n) \rangle \), if

\[
x(n) = f(n) + \sum_{k=0}^{n-1} g(k) x(k)
\]

then

\[
x(n) = f(n) + \sum_{k=0}^{n-1} f(k) g(k) \prod_{j=k+1}^{n-1} (1 + g(j))
\]

Proof. Define

\[
G(n) = \prod_{j=0}^{n-1} (1 + g(j))
\]

and

\[
\chi(n) = f(n) + \sum_{k=0}^{n-1} f(k) g(k) \prod_{j=k+1}^{n-1} (1 + g(j))
= f(n) + \sum_{k=0}^{n-1} f(k) g(k) \frac{G(n)}{G(k+1)}
\]

First we prove the following identity

\[
G(n) = G(k) + \sum_{j=k}^{n-1} g(j) G(j)
\]
It is straightforward to see that

\[
G(0) = 1 \\
G(1) = 1 + g(0) = G(0) + g(0)G(0)
\]

satisfy \(B.5\). If \(B.5\) is true for \(n = m\), then

\[
G(m + 1) = (1 + g(m))G(m) \\
= G(m) + g(m)G(m) \\
= G(k) + \sum_{j=k}^{m-1} g(j) G(j) + g(m)G(m) \\
= G(k) + \sum_{j=k}^{m} g(j) G(j)
\]

and the proof of \(B.5\) follows by induction.

Now, to prove \(B.2\), note that

\[
x(0) = x(0) = f(0)
\]

Now assume \(x(n) = \chi(n)\) for \(0 \leq n < m\). Then consider

\[
x(m) = f(m) + \sum_{k=0}^{m-1} g(k)x(k) = f(m) + \sum_{k=0}^{m-1} g(k) \chi(k)
\]

\[
= f(m) + \sum_{k=0}^{m-1} g(k) \left\{ f(k) + \sum_{j=0}^{k-1} f(j) g(j) \frac{G(k)}{G(j + 1)} \right\}
\]

\[
= f(m) + \sum_{k=0}^{m-1} g(k) f(k) + \sum_{k=0}^{m-1} \sum_{j=0}^{k-1} g(k) f(j) g(j) \frac{G(k)}{G(j + 1)}
\]

\[
= f(m) + \sum_{k=0}^{m-1} g(k) f(k) + \sum_{j=0}^{m-2} \sum_{k=j+1}^{m-1} g(k) f(k) g(k) \frac{G(j)}{G(k + 1)}
\]

\[
= f(m) + g(m-1)f(m-1) + \sum_{k=0}^{m-2} g(k) f(k) \left\{ 1 + \sum_{j=k+1}^{m-1} g(j) \frac{G(j)}{G(k + 1)} \right\}
\]

by using \(B.5\), we get
\[= f(m) + g(m-1)f(m-1) \frac{G(m)}{G(m-1)+1} + \sum_{k=0}^{m-2} g(k) f(k) \frac{G(m)}{G(k+1)}\]

\[= f(m) + \sum_{k=0}^{m-1} f(k) g(k) \frac{G(m)}{G(k+1)} = \chi(m)\]

And the conclusion follows by induction.
Author’s Publications

Journal Papers


Conference Papers


Bibliography


works Applications in Engineering Selected Papers from the 12th {EANN} International Conference, 2011.


discrete-time adaptive control systems using input-output stability theory:
141, Mar 1988.

systems part one: Conditions derived using concepts of loop gain, conicity,
Apr 1966.


[107] Michael George Safonov. *Stability and robustness of multivariable feedback

[108] Qing Song, Jizhong Xiao, and Yeng-Chai Soh. Robust backpropagation
training algorithm for multilayered neural tracking controller. *Neural Net-

[109] Qing Song, Yilei Wu, and Yeng-Chai Soh. Robust adaptive gradient-descent
training algorithm for recurrent neural networks in discrete time domain.

[110] K. Bache and M. Lichman. UCI machine learning repository. *University of
California, Irvine, School of Information and Computer Sciences.*, 2013.

[111] Aboozar Taherkhani, Ammar Belatreche, Yuhua Li, and Liam P Maguire.
DL-ReSuMe: a delay learning-based remote supervised method for spik-
ing neurons. *IEEE transactions on neural networks and learning systems*,