Learning spatio-temporal spike train encodings with ReSuMe, DelReSuMe, and Reward-modulated Spike-timing Dependent Plasticity in Spiking Neural Networks

Doctor of Philosophy

IBRAHIM OZTURK

University of York
Electronic Engineering
April, 2017

Abstract

SNNs are referred to as the third generation of ANNs. Inspired from biological observations and recent advances in neuroscience, proposed methods increase the power of SNNs. Today, the main challenge is to discover efficient plasticity rules for SNNs. Our research aims are to explore/extend computational models of plasticity. We make various achievements using ReSuMe, DelReSuMe, and R-STDP based on the fundamental plasticity of STDP.

The information in SNNs is encoded in the patterns of firing activities. For biological plausibility, it is necessary to use multi-spike learning instead of single-spike. Therefore, we focus on encoding inputs/outputs using multiple spikes. ReSuMe is capable of generating desired patterns with multiple spikes. The trained neuron in ReSuMe can fire at desired times in response to spatio-temporal inputs. We propose alternative architecture for ReSuMe dealing with heterogeneous synapses. It is demonstrated that the proposed topology exactly mimic the ReSuMe. A novel extension of ReSuMe, called DelReSuMe, has better accuracy using less iteration by using multi-delay plasticity in addition to weight learning under noiseless and noisy conditions. The proposed heterogeneous topology is also used for DelReSuMe.

Another plasticity extension based on STDP takes into account reward to modulate synaptic strength named R-STDP. We use dopamine-inspired STDP in SNNs to demonstrate improvements in mapping spatio-temporal patterns of spike trains with the multi-delay mechanism versus single connection. From the viewpoint of Machine Learning, Reinforcement Learning is outlined through a maze task in order to investigate the mechanisms of reward and eligibility trace which are the fundamental in R-STDP. To develop the approach we implement Temporal-Difference learning and novel knowledge-based RL techniques on the maze task. We develop rule extractions which are combined with RL and wall follower algorithms. We demonstrate the improvements on the exploration efficiency of TD learning for maze navigation tasks.

Contents

Abstract **3**

Contents **5**

List of Tables **13**

List of Figures **15**

Acknowledgments **23**

Declaration **25**

1 Introduction **27**
1.1 Motivation 27
1.2 Research Questions 30
1.3 Contributions 31
1.4 Structure of the Thesis 33

2 Biological Neuron, Spiking Neuron Models and Learning **37**
2.1 Introduction 37
2.2 Biological Neuron 38
2.2.1 Structure of Neuron 39
2.2.2 The Membrane Potential 40
2.2.3 Ion Channels 41
2.2.4 The Action Potential 42
2.3 Relation between Artificial and Biological Neurons 43
2.4 History from Artificial Neural Network to Spiking Neural Network 45
2.5 Spiking Neuron Models 47
2.5.1 Compartmental Models 48
2.5.2 Hodgkin-Huxley Model 49
2.5.3 Integrate-and-Fire Model 55
2.5.4 Leaky-Integrate-and-Fire Model 57
2.5.5 Spike Response Model 61

2.5.6 Izhikevich model . . . 62
2.5.7 Discussion . . . 64
2.6 Supervised, Unsupervised, and Reinforcement Learning . . . 66
2.7 Simulators . . . 68
2.7.1 Brian . . . 69
2.8 Summary . . . 71

3 Learning Mechanisms in Spiking Neural Networks 73
3.1 Introduction . . . 73
3.2 Synapse . . . 75
3.2.1 Synaptic Transmission . . . 76
3.2.2 Synapse Model . . . 77
3.2.3 Synaptic Plasticity . . . 77
3.3 Hebbian Plasticity . . . 78
3.3.1 Long Term Potentiation (LTP) . . . 79
3.3.2 Long Term Depression (LTD) . . . 79
3.4 Spike-timing Dependent Plasticity (STDP) . . . 80
3.4.1 Modeling of STDP . . . 82
3.4.2 Weight Dependence . . . 86
3.4.3 Implementation of STDP . . . 87
3.4.4 Weight Bounds . . . 89
3.5 Homeostatic Plasticity . . . 89
3.5.1 Synaptic Scaling . . . 90
3.6 Dopamine Modulated Plasticity . . . 91
3.7 Summary . . . 93

4 Reinforcement Learning 95
4.1 Introduction to Reinforcement Learning . . . 95
4.1.1 Reinforcement Learning in Spiking Neurons . . . 97
4.2 The Agent-Environment Interface . . . 98
4.3 The Markov Property . . . 100
4.4 An Example Markov Chain . . . 101
4.5 Markov Decision Process . . . 104
4.6 Value Function . . . 105
4.7 Bellman Equations . . . 107
4.8 Optimisation . . . 108
4.9 Algorithms for Reinforcement Learning . . . 109
4.9.1 Temporal-Difference Learning . . . 110
4.9.2 Eligibility Traces . . . 110
4.9.3 Q-Learning . . . 111

4.9.4 Replacement Rules for Maze Navigation . . . 114
4.10 Summary . . . 116

5 Extension of Replacement Rules with Reinforcement Learning for Maze Navigation 119
5.1 Introduction . . . 119
5.2 Experimental Setup . . . 120
5.2.1 Task Description . . . 121
5.2.2 Maze Generation . . . 123
5.3 Extension of Replacement Rules . . . 123
5.3.1 Algorithmic Generation of Replacement Rules . . . 127
5.3.2 Comparison to Older Replacement Rules . . . 129
5.4 Replacement Rules with Temporal-Difference Learning . . . 133
5.4.1 Applying to Temporal-Difference Learning . . . 133
5.4.2 Comparison of Proposed Algorithms . . . 134
5.5 Summary . . . 136

6 Spiking Neural Network: Background 139
6.1 Introduction . . . 139
6.2 Spiking Neural Network Topologies . . . 140
6.2.1 Feed-forward Architectures . . . 141
6.2.2 Actor-Critic Networks . . . 142
6.2.3 Recurrent Networks . . . 143
6.3 Spike Generation . . . 143
6.3.1 Spike Train . . . 144
6.3.2 Poisson Processes . . . 144
6.3.3 Homogeneous Poisson Processes . . . 144
6.4 Neural Coding . . . 145
6.4.1 Rate Coding . . . 145
6.4.2 Population Coding . . . 146
6.4.3 Temporal Coding . . . 147
6.4.4 Summary . . . 148
6.5 Measures of Spike Train Difference and Spike Train Synchrony . . . 149
6.5.1 Victor & Purpura Distance (VP) . . . 149
6.5.2 Coincidence Factor (CF) . . . 150
6.5.3 Schreiber Distance (ScD) . . . 150
6.5.4 van Rossum Distance (vRD) . . . 151
6.5.5 Discussion . . . 151
6.6 Summary . . . 152

7 Spiking Neural Network: Implementation 153

7.1 Introduction . . . 154
7.2 Neuron Models . . . 154
7.3 Noise . . . 155
7.4 Delay Mechanisms . . . 158
7.5 Network Architectures . . . 160
7.5.1 Training . . . 160
7.5.2 Testing . . . 162
7.5.3 Bias Neuron . . . 162
7.6 Neural Coding . . . 164
7.6.1 Spike Generation . . . 164
7.6.2 Encoding . . . 165
7.7 Error Analysis . . . 168
7.7.1 Measures of Spike Train Difference . . . 168
7.7.2 Network Classification . . . 169
7.8 Training and Testing Mechanisms . . . 171
7.8.1 Training Mechanism . . . 171
7.8.2 Testing Mechanism . . . 173
7.9 Benchmark Descriptions . . . 174
7.9.1 Mapping . . . 174
7.9.2 Logical Operations . . . 175
7.10 Summary . . . 175

8 ReSuMe and DelReSuMe **177**
8.1 Introduction . . . 177
8.2 SpikeProp . . . 179
8.3 Remote Supervised Method . . . 181
8.4 Architecture for Proposed Topology . . . 184
8.5 Error Analysis . . . 189
8.6 ReSuMe Overall Setup Experiments . . . 190
8.7 Experiments: Mapping for Remote Supervised Method . . . 191
8.7.1 Results of Noiseless Simulations . . . 191
8.7.2 Results of Noisy Simulations . . . 195
8.7.3 Discussion of Mapping . . . 198
8.8 Experiments: Logical Operations for Remote Supervised Method . . . 199
8.8.1 Results of Noiseless Simulations . . . 200
8.8.2 Discussion . . . 202
8.9 Delayed Remote Supervised Method . . . 206
8.10 Experiments: Mapping for Delayed Remote Supervised Method . . . 209
8.10.1 Results of Noiseless Simulations . . . 210

8.10.2 Results of Noisy Simulations . . . 214
8.10.3 Discussion of Mapping . . . 215
8.11 Summary . . . 217

9 Reward-modulated STDP 219
9.1 Introduction . . . 219
9.2 Reward Mechanism . . . 222
9.3 Eligibility Traces . . . 224
9.4 Comparison between R-STDP and R-max . . . 226
9.5 Overall Setup . . . 228
9.6 Experiments: Mapping . . . 230
9.6.1 Results of Noiseless Simulations (Single-Connection) . . . 231
9.6.2 Results of Noiseless Simulations (Multi-Constant-Delay) . . . 234
9.6.3 Results of Noisy Simulations (Single-Connection) . . . 234
9.6.4 Results of Noisy Simulations (Multi-Constant-Delay) . . . 241
9.6.5 Discussion . . . 242
9.7 Summary . . . 248

10 Conclusion - Contributions and Future Research 251
10.1 Conclusion and Contributions . . . 251
10.2 Future Research . . . 257

Appendices 261

A Functions Used 261
A.1 Dirac Delta Function . . . 261
A.2 Alpha Function . . . 261
A.3 Signum Function . . . 262
A.4 Heaviside Step Function . . . 262
A.5 Truth Tables for Used Operations . . . 262
A.6 LCE Calculation Formula . . . 263

B Simulators 265
B.1 NEURON . . . 265
B.2 NEST . . . 266
B.3 GENESIS . . . 266
B.4 Nengo . . . 267

C Simulation Framework Documentation 269
C.1 Introduction . . . 270
C.2 Motivation : The Memory Problem . . . 270

C.3 Software Specification . . . 271
C.4 Description of Simulation Sections . . . 271
C.5 Simulation Folder Mechanism . . . 272
C.5.1 Session Mechanism . . . 272
C.5.2 Initialization Mechanism . . . 272
C.5.3 Spike Set Mechanism . . . 273
C.6 Modes . . . 273
C.6.1 Run Mode . . . 273
C.6.2 Analyse Mode . . . 274
C.7 Stored Dynamics . . . 274
C.8 Learning Algorithms . . . 275
C.9 Parameter Settings . . . 275
C.9.1 Neuron Types and Dynamics . . . 276
C.9.2 Number of Neurons . . . 276
C.9.3 The Distribution of Input Types . . . 276
C.9.4 Learning Rate . . . 277
C.9.5 Min/Max Learning Weights . . . 278
C.9.6 Min/Max Initial Weights . . . 278
C.9.7 Scaling Status . . . 278
C.9.8 Scaling Factor . . . 278
C.9.9 Initialization Methods . . . 278
C.9.10 Timings . . . 279
C.10 Generation of Spike Sets . . . 280
C.10.1 File System of Spike Sets . . . 282
C.11 File Content of Session for Training and Testing . . . 283
C.12 The Generations of Dynamic Initializations (File System) . . . 283
C.13 Used File Formats . . . 284
C.14 Further Work . . . 284
C.15 Discussions . . . 285

D Further Results **287**
D.1 Further Results for Mapping Experiments . . . 287
D.1.1 ReSuMe Experiments . . . 287
D.1.2 DelReSuMe Experiments . . . 287
D.2 Maze Platform on Python . . . 287
D.3 Maze Generation . . . 290
D.4 Maze Solving Algorithms . . . 291
D.4.1 Left Wall Follower (LSR Rule) . . . 292
D.4.2 Right Wall Follower (RSL Rule) . . . 293

D.5 Example Screenshot from Workspace 294
D.6 Prerequisites 294
D.7 Software Contributions 295

E Credits **297**

Glossary **299**

Bibliography **303**

List of Tables

2.1 Parameters for the neuron models and synapses 52

2.2 The parameters of the Hodgkin-Huxley equations 53

2.3 Empirical functions of α and β by Hodgkin and Huxley 54

2.4 Model parameters for the Leaky-Integrate-and-Fire simulation 60

2.5 Model parameters for the Izhikevich model simulation 64

3.1 Model parameters used for the computer simulations through STDP 86

4.1 Basic Replacement Rules for LSR (BscRepLSR): Replacement Rules from Venkata et al. (2011) 115

5.1 Extended Replacement Rules for LSR (ExtRepLSR): Replacement Rules from proposed list through LSR 125

5.2 Extended Replacement Rules for RSL (ExtRepRSL): Replacement Rules from proposed list through RSL 125

5.3 Efficiency comparison of Replacement Rules through four different groups of maze sizes 132

7.1 Parameters of the network architecture for mapping and logical operation benchmarks 160

7.2 Encoding parameters for mapping and logical operation benchmarks based on ReSuMe, DelReSuMe and R-STDP 166

8.1 Model parameters used for ReSuMe used in Figure 8.1 183

8.2 Model parameters used for error analysis 190

8.3 Length parameters used for the computer simulations 190

8.4 Model parameters used for the computer simulations through ReSuMe 191

8.5 Summary of all operations through ReSuMe 205

9.1 Length parameters used for the computer simulations through R-STDP . . . 228

9.2 Model parameters used for the computer simulations during R-STDP 229

9.3 Scaling parameters for R-STDP 229

9.4 Parameters of the network architecture for mapping benchmark with a single-connection and multi-constant-delay mechanism 230

A.1 Operation P1 262

A.2 Operation TRUE 263

A.3 Operation AND 263

A.4 Operation OR 263

A.5 Operation Exclusive Or (XOR) 263

C.1 Format for initializations : Weights (in the top row) and delays (in the bottom row) 273

C.2 Typical durations of simulation parameters for training and testing 280

C.3 Format for firing times to write into file and program memory 282

C.4 The contents of session configuration file 283

List of Figures

2.1 The morphology of three neurons 39

2.2 A schematic diagram of a section of the lipid bilayer 41

2.3 The action potential and axon-dendrite junction 42

2.4 Typical form of an action potential based on Hodgkin-Huxley model 43

2.5 The artificial neuron model from human neuron 45

2.6 Compartmental model for pyramidal cell 48

2.7 The equivalent circuit for a generic neural compartment 49

2.8 The proposed equivalent electrical circuit with three channels by Hodgkin-Huxley 51

2.9 Simulation of a Hodgkin-Huxley neuron in response to an external input current 54

2.10 Equivalent circuit diagram for the IF neuron model 56

2.11 Simulation of a single Non-Leaky IF neuron by an external input current . . 57

2.12 Equivalent circuit diagram for the LIF neuron model 58

2.13 Simulation of a single LIF neuron by an external input current 59

2.14 Simulation of a single Izhikevich model neuron by an external input current . 63

3.1 Illustration of the main parts of a synapse 76

3.2 Schematic representation of a synapse from the source neuron to the target neuron 76

3.3 Sample experiments illustrating LTP and LTD 80

3.4 Symmetric-STDP 82

3.5 Asymmetric-STDP 82

3.6 The schematic of STDP window function 83

3.7 Illustration of the implementation of the STDP process over simulation time steps, $dt = 1ms$ 85

3.8 Overview of reward structures in the human brain 92

4.1 The Reinforcement Learning framework 99

4.2 The Reinforcement Learning state transition 99

4.3 Possible states from a four-by-four maze platform 102

4.4 Discrete Time Markov Chain Graphic Model based on given maze 103

4.5 Maze task with all possible actions and their action names 113

4.5 Maze task with all possible actions and their action names 114

5.1 4x4 maze platform from drawn and the simulation environment 121

5.2 An example path from starting room to goal 122

5.3 Maze platform with all possible directions and an example of possible initial possibilities on a specific room 122

5.4 An example for deadlock state scenario 123

5.5 Replacement Rule LUR to U 124

5.6 First and second run screenshots (Experiment 3) 126

5.7 First and second run screenshots (Experiment 1) 126

5.8 First and second run screenshots (Experiment 2) 129

5.9 The performance of Replacement Rules through LSR rule exploration by 05x05 mazes 130

5.10 The performance of Replacement Rules through LSR rule exploration by 10x10 mazes 130

5.11 The performance of Replacement Rules through LSR rule exploration by 15x15 mazes 131

5.12 The performance of Replacement Rules through LSR rule exploration by 20x20 mazes 131

5.13 Replacement Rules with Q-Learning 134

5.14 The comparison of revisit rates 135

5.15 The comparison of total taken discounted reward 136

6.1 Illustration of a typical feed-forward SNN structure 141

6.2 Illustration of a typical actor-critic architecture 142

6.3 Illustration of a recurrent SNN structure 143

6.4 A typical spatiotemporal spike pattern 147

7.1 The characteristics of the low noise and how it affects neuronal activity . . . 156

7.2 The characteristics of the high noise and how it affects neuronal activity . . . 157

7.3 Proposed delay mechanism 159

7.4 Illustration of an actor-critic network architecture 161

7.5 Proposed network architecture for ReSuMe training of Spiking Neural Network 162

7.6 The network architecture during training for Reinforcement Learning in spiking neural network 163

7.7 The network architecture during testing for all proposed learning techniques in Spiking Neural Network 164

7.8 Prepared spike trains for the output 167

7.9 Prepared spike trains for the input 168

7.10 Illustration of van Rossum Distance measure 169

7.11 Block diagram of mapping benchmarks with spike trains 175

7.12 Block diagram of logical operation benchmarks with spike trains 176

8.1 Illustration of concepts underlying the ReSuMe learning method from Ponulak (2008) 183

8.2 The comparison for pre-post neuron connection for the current and proposed structure 185

8.3 Minimalist network scenario for the proposed topology 187

8.4 Simulation scenario with different number of spikes in desired pattern 188

8.4 Simulation scenario with different number of spikes in desired pattern 189

8.5 Reconstruction of the transformation from input patterns to output spike timings (noiseless condition) 193

8.6 Evolution of synaptic weights for the two mapping experiments (noiseless condition) . . . 194

8.7 Histogram of weights is prepared from the lower experiment in Figure 8.6 (noiseless condition) . . . 194

8.8 Averaged-vRE for mapping experiments . . . 195

8.9 Reconstruction of the transformation from input patterns to output spike timings (low noise) . . . 196

8.10 Evolution of synaptic weights for the two mapping experiments (low noise) . 197

8.11 Histogram of weights through noisy conditions (low noise) (prepared from the lower experiment in Figure 8.6 . . . 197

8.12 Reconstruction of the transformation from input patterns to output spike timings (high noise) . . . 198

8.13 Evolution of synaptic weights for the two mapping experiments (high noise) . 198

8.14 Histogram of weights through noisy conditions (high noise) (prepared from the lower experiment in Figure 8.13 . . . 199

8.15 Averaged-vRE for operation TRUE . . . 200

8.16 Averaged-vRE for operation P1 . . . 200

8.17 Averaged-vRE for operation AND . . . 201

8.18 Averaged-vRE for operation OR . . . 201

8.19 Averaged-vRE for operation XOR . . . 202

8.20 Logic Classification Error (LCE) for operation TRUE . . . 202

8.21 Logic Classification Error (LCE) for operation P1 . . . 203

8.22 Logic Classification Error (LCE) for operation AND . . . 203

8.23 Logic Classification Error (LCE) for operation OR . . . 204

8.24 Logic Classification Error (LCE) for operation XOR . . . 204

8.25 An exponential trace for synapse k with its internal delay in DelReSuMe . . . 207

8.26 An exponential trace for each synaptic delay in DelReSuMe . . . 208

8.27 Reconstruction of the transformation from input patterns to output spike timings (noiseless condition) . . . 210

8.28 Reconstruction of the transformation from input patterns to output spike timings (noiseless condition). Zoomed version of Figure 8.5 and Figure 8.27 . 211

8.29 Evolution of synaptic weights for the two mapping experiments (noiseless condition) 212

8.30 Histogram of weights is prepared from the experiment in Figure 8.29 (noiseless condition) 212

8.31 The comparison of Remote Supervised Method and Delayed Remote Supervised Method through mapping tasks 213

8.32 Zoomed version of the comparison of Remote Supervised Method and Delayed Remote Supervised Method through mapping tasks 213

8.33 Reconstruction of the transformation from input patterns to output spike timings (low noise) 215

8.34 Evolution of synaptic weights for the mapping experiment (low noise) 215

8.35 Histogram of weights is prepared from the experiment in Figure 8.34 (low noise) 216

8.36 Reconstruction of the transformation from input patterns to output spike timings (high noise) 216

8.37 Evolution of synaptic weights for the mapping experiment (high noise) 217

8.38 Histogram of weights is prepared from the experiment in Figure 8.34 (high noise) 217

9.1 Schematic of Reward-modulated Spike-timing Dependent Plasticity learning rule 222

9.2 An example of eligibility trace and weight update during a single learning cycle 226

9.3 Reconstruction of the transformation from input patterns to output spike timings (noiseless, without delay) 232

9.4 Evolution of synaptic weights for the mapping experiment (noiseless, without delay) 232

9.5 Histogram of weights is prepared from the experiment in Figure 9.4 (noiseless, without delay) 233

9.6 The trajectory of current and averaged reward versus van Rossum Distance (vRD) randomly selected simulations (noiseless, without delay) 233

9.7 Reconstruction of the transformation from input patterns to output spike timings (noiseless, multi-delay) 235

9.8 Evolution of synaptic weights for the mapping experiment (noiseless, multi-delay) 235

9.9 Histogram of weights is prepared from the experiment in Figure 9.8 (noiseless, multi-delay) 236

9.10 The trajectory of current and averaged reward versus van Rossum Distance (vRD) randomly selected simulations (noiseless, multi-delay) 236

9.11 Reconstruction of the transformation from input patterns to output spike timings (low noise, without delay) 237

9.12 Evolution of synaptic weights for the mapping experiment (low noise, without delay) 237

9.13 Histogram of weights is prepared from the experiment in Figure 9.12 (low noise, without delay) 238

9.14 The trajectory of current and averaged reward versus van Rossum Distance (vRD randomly selected simulations) (low noise, without delay) 238

9.15 Reconstruction of the transformation from input patterns to output spike timings (high noise, without delay) 239

9.16 Evolution of synaptic weights for the mapping experiment (high noise, without delay) 239

9.17 Histogram of weights is prepared from the experiment in Figure 9.16 (high noise, without delay) 240

9.18 The trajectory of current and averaged reward versus van Rossum Distance (high noise, without delay) 240

9.19 Reconstruction of the transformation from input patterns to output spike timings (low noise, multi-delay) 241

9.20 Evolution of synaptic weights for the mapping experiment (low noise, multi-delay) 242

9.21 Histogram of weights is prepared from the experiment in Figure 9.20 (low noise, multi-delay) 242

9.22 The trajectory of current and averaged reward versus van Rossum Distance (vRD randomly selected simulations) (low noise, multi-delay) 243

9.23 Reconstruction of the transformation from input patterns to output spike timings (high noise, multi-delay) 243

9.24 Evolution of synaptic weights for the mapping experiment (high noise, multi-delay) 244

9.25 Histogram of weights is prepared from the experiment in Figure 9.24 (high noise, multi-delay) 244

9.26 The trajectory of current and averaged reward versus van Rossum Distance (vRD randomly selected simulations) (high noise, multi-delay) 245

9.27 The trajectory of averaged reward versus van Rossum Distance (low noise, single connection and the multi-delay comparison) 246

9.28 The trajectory of averaged reward versus van Rossum Distance (high noise, single connection and the multi-delay comparison) 247

D.1 Reconstruction of the transformation from input patterns to output spike timings randomly selected experiment (Experiment 1 - noiseless) 288

D.2 Reconstruction of the transformation from input patterns to output spike timings randomly selected experiment (Experiment 2 - noiseless) 288

D.3 Reconstruction of the transformation from input patterns to output spike timings randomly selected experiment (Experiment 3 - noiseless) 289

D.4 Reconstruction of the transformation from input patterns to output spike timings randomly selected experiment (Experiment 4 - noiseless) 289

D.5 Reconstruction of the transformation from input patterns to output spike timings randomly selected experiment (Experiment 5 - noiseless) 290

D.6 Reconstruction of the transformation from input patterns to output spike timings (Zoomed-Experiment 1) 290

D.7 Reconstruction of the transformation from input patterns to output spike timings (low noise). Zoomed version of Figure 8.9 291

D.8 Python maze simulation menu 292

D.9 Python maze simulation environment 293

D.10 Maze simulation environment with the size of 40 by 40 on Python 294

D.11 Maze generation steps based on Backtrack Recursion algorithm 295

D.12 Example screenshot from workspace 296

Acknowledgments

In the Name of Allah, Most Gracious, Most Merciful.

Alhamdulillah, all praises to Allah for the strengths and His blessing in completing this thesis in the time given. I would like to thank my supervisor, Dr. David Halliday, for his assistance, encouragement and guidance which he has provided to me. Also I would like to thank my thesis advisor Prof. Jon Timmis for his advice, guidance and comments on this work.

I have made lots of friends/colleagues during my study in the University of York and I thank them for their full support and encouragement. Their views are extremely useful for me. Unfortunately, it is not possible to list all of them in this limited space.

I would like to thank the Republic of Turkey Ministry of National Education for its financial support during my entire education.

The Brian Users' Group has provided an excellent help in early stages of developments. I would especially like to thank Marcel Stemberg and Dan Goodman for answering all the questions I had about the Brian simulator environment.

Lastly, and most importantly, I would like to thank my amazing family for the love, support, and constant encouragement I have gotten over my entire study. In particular, I would like to thank my wife for her continued support.

To my beloved parents, wife, and daughter,
who taught me the power of positive thinking.

Declaration

Selected aspects of the research described in this thesis, as well as some earlier work have been documented or published elsewhere:

Ibrahim Ozturk and David M. Halliday. *Mapping Spatio-temporally Encoded Patterns by Reward-Modulated STDP in Spiking Neurons.* In Proceedings of International Conference on Computational Intelligence (IEEE SSCI 2016), Athens, Greece, December, 2016

Ibrahim Ozturk and David M. Halliday. *Knowledge based Reinforcement Learning in Maze Navigation Task.* Poster for YDS 2014. (Poster Competition Fourth Prize).

Ibrahim Ozturk and David M. Halliday. *Novel Replacement Rules on Reinforcement Learning for the Maze Task : Tested on Q-Learning.* To be submitted: International Conference on Computational Intelligence (IEEE SSCI 2018).

Ibrahim Ozturk and David M. Halliday. *Delayed Remote Supervised Method (DelReSuMe).* To be submitted: International Conference on Computational Intelligence (IEEE SSCI 2018).

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Chapter 1

Introduction

Contents

1.1 Motivation **27**
1.2 Research Questions **30**
1.3 Contributions **31**
1.4 Structure of the Thesis **33**

This chapter gives a brief overview of the research done in the thesis. Section 1.1 gives a short recap of the project background and motivation. Section 1.2 presents the research questions with our goal. Then, in section 1.3, intended contributions are presented. Finally, the structure of the thesis is outlined in section 1.4.

1.1 Motivation

Artificial Neural Networks (ANNs) have three generations based on their computational units (Maass, 1996; Gerstner & Kistler, 2002; Bohte et al., 2002b, a). Third generation of ANNs, based on spiking neurons, are more biologically plausible and more efficient computationally for various problems (Maass, 1996; Gerstner & Kistler, 2002). Spiking Neural Networks (SNNs) process the information encoded in the timing between neuron firings. As biological neurons communicate by receiving and transmitting pulses or "spikes", then SNNs also carry information across synapses between other neurons in the network using spikes.

Individual computational neurons in each layer of SNNs can be mathematically modelled in a number of different ways. The model of Hodgkin-Huxley (HH) describes biological neurons closely with increased computational expense. However, the Leaky-Integrate-and-Fire (LIF)

models the spiking dynamics with a reduced computational cost. Although simplified neurons have different advantages and drawbacks, we use the LIF model.

The complexity and the computational power of artificial SNNs are restricted compared with the abilities of biological neural systems (Kasinski & Ponulak, 2006). However, developing efficient learning techniques and recognizing information codes in SNNs are still open problems from a computational point of view. Various characteristics of SNNs such as learning algorithms and theoretical models can be considered to understand the ability of SNNs. Therefore, we focus on various learning mechanisms for SNNs, especially the biologically plausible plasticity rules. In addition, we consider how can we improve these in a biologically plausible sense. For this reason, we mainly use multi-spike coding rather than single-spike coding schemes in Spiking Neural Networks.

One extension of synaptic plasticity rules can be achieved by taking into account reward/punishment as reinforcements to modulate synaptic strength. As a fundamental plasticity rule, the mechanism of Spike-timing Dependent Plasticity (STDP) strengthens or weakens synapses based on the pre- and post-synaptic firing times. Recently, several plasticity experiments, including STDP, demonstrate that neuromodulators, particularly Dopamine (DA) related to novelty and reward prediction, have global mechanisms for synaptic modification (Pawlak & Kerr, 2008; Wickens, 2009). This process is referred to as Reinforcement Learning (RL). In order to consolidate changes of synaptic strength in response to pre- and post-synaptic neuronal activity such a signal could be used as a reward (Bailey et al., 2000). In order to investigate further potential plasticity mechanisms considering with biological plausibility, the details of Reinforcement Learning from the viewpoint of Machine Learning is outlined.

Dopamine (DA) plays a key role in Reinforcement Learning in the human body (Schultz, 1998, 2007). In the Reinforcement Learning literature, an agent tries to maximize the cumulative reward by taking actions in an environment. Modulation of STDP with Dopamine can be linked to Reinforcement Learning in Spiking Neural Networks. We introduce an RL environment through a maze task. To develop the approach, we implement Q-Learning and novel knowledge based Reinforcement Learning techniques on the maze task. This achievement helps us during Reward-modulated Spike-timing Dependent Plasticity (R-STDP). Through R-STDP experiments, our aim is to demonstrate Reinforcement Learning in mapping spatio-

temporal patterns of spike trains.

Spiking Neural Networks consist of nodes and the connections between them. The class of nodes includes spiking neurons and objects which enable stimulating or recording from neurons. Synaptic plasticity is handled by the modifications of synaptic weight in algorithms such as SpikeProp (Bohte et al., 2002b) and Remote Supervised Method (Ponulak & Kasinski, 2010). Synaptic weights determine the impact of the pre-synaptic signal on the post-synaptic potential. However, synaptic delays determine how long it takes for the signal to travel from the pre-synaptic to the post-synaptic neuron. Hence, the dynamics of synaptic delays can also be taken into account in order to have better performance. For this purpose, we develop a novel algorithm by extending existing ReSuMe and we call it Delayed Remote Supervised Method (DelReSuMe). The DelReSuMe rule modifies both synaptic weights and delays to have faster convergence compared with ReSuMe.

In order to implement Remote Supervised Method, although there are existing SNN simulators, they do not support ReSuMe because of its heterogeneous dynamics. In this case, researchers have to implement ReSuMe in a custom environment in order to duplicate ReSuMe based results, or try to extend the algorithm. At this stage, we propose a novel and alternative connection structure for ReSuMe compatible with existing SNN simulators. Experimental evidences on the Brian simulator show that proposed heterosynaptic architecture works exactly the same as existing ReSuMe with its design benefits.

The management of training and/or testing in a Spiking Neural Network is more complicated then in a traditional ANN. As the size of network or the duration of simulation grows, the performance of experiments for training/testing and the management of the implementation become much more complex in SNN simulations. However, as far as we know, there is no public tool available for the Brian package or other SNN simulators to ease the organization of training/testing sessions. Thus, we develop our own framework for the Brian SNN simulation environment. The framework is tested with four training algorithms that apply weight and/or delay modification by capturing the detailed firing of individual neurons. One algorithm modifies weights and delays. The other algorithms apply learning rules to only weights. We test the framework with the listed algorithms:

- Spike-timing Dependent Plasticity (STDP) (detailed in section 3.4)
- Remote Supervised Method (ReSuMe) (detailed in section 8.3)
- Delayed Remote Supervised Method (DelReSuMe) (detailed in section 8.9)
- Reward-modulated Spike-timing Dependent Plasticity (R-STDP) (detailed in chapter 9)

In addition to those listed algorithms tested through the framework, using existing SNN simulators can be set up using this approach.

1.2 Research Questions

Although we start to ask the main research question of "Can we extend plasticity mechanisms or topologies to incorporate multi-spike coding in Spiking Neural Networks?", there are a number of research questions addressed in this thesis. Let us review them:

- **(a)** : Can we extend the exploration performance of Reinforcement Learning, focusing on existing Temporal-Difference Learning, for maze tasks?

  This is achieved in chapter 4. We introduce various domain-based rules for different maze solving algorithms in order to reduce the exploration period.

- **(b)** : Can we develop Remote Supervised Method (ReSuMe) or similar techniques using multi-spike coding schemes in SNNs through available simulators, specifically Brian, which does not support heterogeneous synapses? Can we demonstrate the results of the proposed implementation through different tasks under noiseless and realistically noisy conditions?

  This question is answered through various experiments including mapping tasks and logical operations in chapter 8 and chapter 9. Furthermore, in order to ease the management of simulations and the modularity of initialization/training/testing sessions, we propose a framework for SNNs which is tested and described in appendix C.

- **(c)** : Is it be possible to extend the ReSuMe algorithm in order to have more efficient performance with faster convergence in SNNs? Can we compare the performance of the ReSuMe and proposed method under noiseless, and realistically noisy conditions?

  This question is answered in section 8.9. Slightly better accuracy using many fewer learning iterations is achieved by using delay plasticities in addition to weight learning. This novel plasticity is named the Delayed Remote Supervised Method, DelReSuMe.

- **(d)** : Can we demonstrate the mapping of spatio-temporal patterns with multi-spike coding using Dopamine modulated Spike-timing Dependent Plasticity in SNNs?

  This is achieved in chapter 9. The existing technique of Reward-modulated Spike-timing Dependent Plasticity is implemented with different parameters for multi-spike coding. The algorithm is tested using the proposed framework for SNNs.

1.3 Contributions

In this thesis we make the various contributions which are summarized below. More details of contributions can be found in section 10.1.

- Rule extractions for the maze task. We introduce a set of novel Replacement Rules for maze environments named as Extended Replacement Rules for LSR and Extended Replacement Rules for RSL with a remarkable performance compared to Left-Straight-Right (LSR) rule or Right-Straight-Left (RSL) rule itself, also to previously offered set of rules from Venkata et al. (2011) (named as Basic Replacement Rules for LSR). Replacement Rules are also combined with a Reinforcement Learning algorithm and wall follower algorithms (LSR and RSL rules). We experimentally demonstrate the improvements on the exploration efficiency of the Reinforcement Learning algorithm for a maze navigation task via the proposed set of rules. Details can be found in chapter 4.

- Remote Supervised Method (ReSuMe) implementation for multi-spike coding. This is tested with various benchmarks in the face of noiseless, relatively low noise and relatively

high noise conditions: mapping experiments and logical operations. Details can be found in chapter 8.

- Extended version of Remote Supervised Method as Delayed Remote Supervised Method is proposed adding delay learning into existing weight learning in ReSuMe. The learning efficiency of the modified algorithm DelReSuMe is compared to ReSuMe on a series of mapping tasks in the face of noiseless, relatively low noise and relatively high noise conditions. Faster learning and convergence with slightly better accuracy have been achieved in section 8.9.

- We introduce an alternative synaptic connection for Remote Supervised Method and Delayed Remote Supervised Method dealing with learning through heterogeneous synapses. The novel connection scheme using the proposed bias neuron is introduced for ReSuMe with its implementation and mathematical/topological descriptions. It is experimentally demonstrated that the proposed heterosynaptic topology can exactly mimic the ReSuMe weight change through two synapses from input to actual output neuron and desired output neuron. Details can be found in chapter 8.

- We combine Spike-timing Dependent Plasticity with the reward modulation as Reward-modulated Spike-timing Dependent Plasticity for multi-spike coding based on the multi-delay mechanism. We compare single connection without synaptic delays and with the multi-delay mechanism in the face of noiseless, relatively low noise and relatively high noise conditions. We show that the proposed architecture for Reward-modulated Spike-timing Dependent Plasticity have better convergence speed under noiseless, and realistically noisy conditions than the single connection without delay structure for the same mapping tasks. Details can be found in chapter 9.

- We develop a new framework for our SNN simulator in order to ease the management of initialization/training/testing sessions. It is tested with Brian simulator. The entire simulation scenarios based on SNNs are implemented using this framework. The documentation of the framework can be found in appendix C.

1.4 Structure of the Thesis

The structure of this thesis is organized as follows:

Chapter 2, *Biological Neuron, Spiking Neuron Models and Learning*, begins with an overview of the physiology of the neuron and its mathematical model's which are fundamentals for the following chapters. It describes details of action potential generation. This is followed with a review and example implementations of various approaches in order to model the neuron transfer function such as neuron models of Hodgkin-Huxley, Integrate-and-Fire, Spike Response Model and Izhikevich model. It also gives an overview of some common simulators, focusing on the Brian simulator, for Spiking Neural Networks. The chapter is concluded with the summary and the reasons to choose Brian package for the experiments throughout the thesis.

Chapter 3, *Learning Mechanisms in Spiking Neural Networks*, discusses learning mechanisms for spiking neurons, especially biologically plausible plasticity rules. Learning in neuroscientific observations is modelled by modifications in the strength of connections between neurons. The connection between neurons is called a synapse, and the ability of its strength to change over time is called synaptic plasticity. This chapter defines the biological background of the synapse, and then reviews experimental protocols that can induce synaptic plasticity.

The goal of solving Reinforcement Learning (RL) tasks is to learn how to perform actions in order to maximise the reward in the long term. Chapter 4, *Reinforcement Learning*, examines the background to RL beginning with formalization as a Markov Decision Process. A system in RL learns through feedback without explicit teaching. We review some standard RL algorithms such as Q-Learning on the maze task. We introduce a series of novel Replacement Rules for the maze task which can be used alone or can be applied in any RL technique on a maze task in order to shorten the exploration period. Improvements of the proposed rules through learning more quickly are also demonstrated in practice.

Chapter 5, *Extension of Replacement Rules with Reinforcement Learning for Maze Navigation*, demonstrates a set of novel Replacement Rules as Extended Replacement Rules for LSR and Extended Replacement Rules for RSL in section 5.3 for maze environments. Replacement

Rules are also combined with a Reinforcement Learning algorithm and wall follower algorithms (Left-Straight-Right and Right-Straight-Left rules). Experimental results for improvements on the exploration efficiency of the Reinforcement Learning algorithm are demonstrated for a maze navigation task via the proposed set of rules.

Chapter 6, *Spiking Neural Network: Background*. Network topologies, focusing on Spiking Neural Networks, are summarized. Spike train notation for generation of artificial spike trains for experiments is described here. Some important concepts related to the simulation of Spiking Neural Networks, such as spike encoding methods and the measurement of spike train synchrony, are also discussed here. The proposed encoding mechanism for spatio-temporal patterns is presented.

Chapter 7, *Spiking Neural Network: Implementation*, presents the common mechanisms and techniques during the experiments. The types of neuron models used in proposed SNNs are summarized considering the scenario of the presence of noise. In addition, the proposed mechanism of multiple delay connections between input and output neurons is described. The network architectures of spiking neurons during training and testing are demonstrated. The adoption of van Rossum Distance (vRD) in order to measure spike-train similarity and a (mis)classification error metric in order to evaluate task performance of the network are detailed. Also, the pseudocodes of training and testing mechanisms throughout the experiments are outlined. Finally, benchmarks used during performed experiments are introduced in section 7.9.

Chapter 8, *ReSuMe and DelReSuMe*. This chapter focuses on the description of the implemented learning mechanisms for the spiking neural model. It begins with a review of Spike Propagation (SpikeProp), which is one of the baseline learning algorithms for SNNs. Then, Remote Supervised Method (ReSuMe) with various improvements to SpikeProp are discussed for the Leaky-Integrate-and-Fire model. Two benchmarks of mapping and logical operations are addressed. In the implementation with Brian package, several issues are observed, which are closely analysed. For that reason, a novel connection scheme is introduced for ReSuMe with its implementation and its mathematical description. The learning efficiency of the modified algorithm DelReSuMe is compared to ReSuMe on a series of mapping tasks. We have also developed a framework for Brian to handle training and testing. Then new proposed simulation framework is detailed in appendix C. The framework enables us to perform all

experiments on SNNs in chapters 8 and 9.

Chapter 9, *Reward-modulated STDP*, presents the implementation of the Reward-modulated Spike-timing Dependent Plasticity. We apply the plasticity for the mapping of multi-spike coding with spatio-temporal patterns. The convergence results of learning are presented with spike distance metrics. The same framework in appendix C for the simulation management is used during the experiments in this chapter.

The overall achievements of this thesis and their implications are discussed in Chapter 10, *Conclusion - Contributions and Future Research*, closing with limitations and future research directions to extend the proposed ideas.

Chapter 2

Biological Neuron, Spiking Neuron Models and Learning

Contents

2.1 Introduction . . . 37
2.2 Biological Neuron . . . 38
2.2.1 Structure of Neuron . . . 39
2.2.2 The Membrane Potential . . . 40
2.2.3 Ion Channels . . . 41
2.2.4 The Action Potential . . . 42
2.3 Relation between Artificial and Biological Neurons . . . 43
2.4 History from Artificial Neural Network to Spiking Neural Network . . . 45
2.5 Spiking Neuron Models . . . 47
2.5.1 Compartmental Models . . . 48
2.5.2 Hodgkin-Huxley Model . . . 49
2.5.3 Integrate-and-Fire Model . . . 55
2.5.4 Leaky-Integrate-and-Fire Model . . . 57
2.5.5 Spike Response Model . . . 61
2.5.6 Izhikevich model . . . 62
2.5.7 Discussion . . . 64
2.6 Supervised, Unsupervised, and Reinforcement Learning . . . 66
2.7 Simulators . . . 68
2.7.1 Brian . . . 69
2.8 Summary . . . 71

2.1 Introduction

The human body has trillions of cells. The basic building block of the nervous system is specialized cells called neurons. Neurons, also known as nerve cells, provide the processing

that occurs in the central nervous system (CNS). The neurons are able to propagate signals over large distances through chemical and electrical signals to other neurons as well as receive information from other connected neurons. This remarkable transmission can be done via generating and transmitting characteristic electrical pulses called action potentials described in subsection 2.2.4. Neurons carry messages electrochemically as movements of ions controlled by chemical processes. The physiology of the neuron including the action potential as a carrier of information between neurons is discussed in section 2.2.

In the history of neuroscience, the first mathematical model of action potential generation is proposed by Alan Lloyd Hodgkin and Andrew Fielding Huxley in 1952. This classical work is named the Hodgkin-Huxley model detailed in subsection 2.5.2. Following a description of this various modelling approaches for the neuron transfer function are developed including compartmental approach in subsection 2.5.1, Hodgkin-Huxley model in subsection 2.5.2, Integrate-and-Fire (IF) neuron model in subsection 2.5.3, Leaky-Integrate-and-Fire (LIF) model in subsection 2.5.4, Spike Response Model (SRM) in subsection 2.5.5 and Izhikevich model (IM) in subsection 2.5.6. Furthermore, to simulate SNNs, Brian simulator is compared existing tools detailed in section 2.7.

2.2 Biological Neuron

In 1891, Waldeyer suggested that neurons are the functional and structural units of the nervous system, this is known as the “Neuron doctrine” or “Neuron Theory”. This basis about the neuron is stated as “If ... we accept the existence of nerve networks our interpretation is somewhat altered, but nevertheless we can retain the nerve-units. Then the boundary between two nerve-units would always lie in a nerve network ...” (Jacobson, 1993).

Neurons are responsible for communicating and processing all the information in the brain. That information covers motor information between muscles; sensory information associated with vision, hearing, touch, taste, smell and balance-movement; cognitive information needed to reason, think and learn. Since our research is about the plasticity between neurons, it is important to detail the structure and function of biological neurons.

2.2.1 Structure of Neuron

Before we go into details on how neural networks function, it is essential to give a brief overview of the neuron anatomy. Although there are various types of nerve cells, in general they all share common parts: soma (cell body), axon (a long slender) and dendrites (tree-like structures).

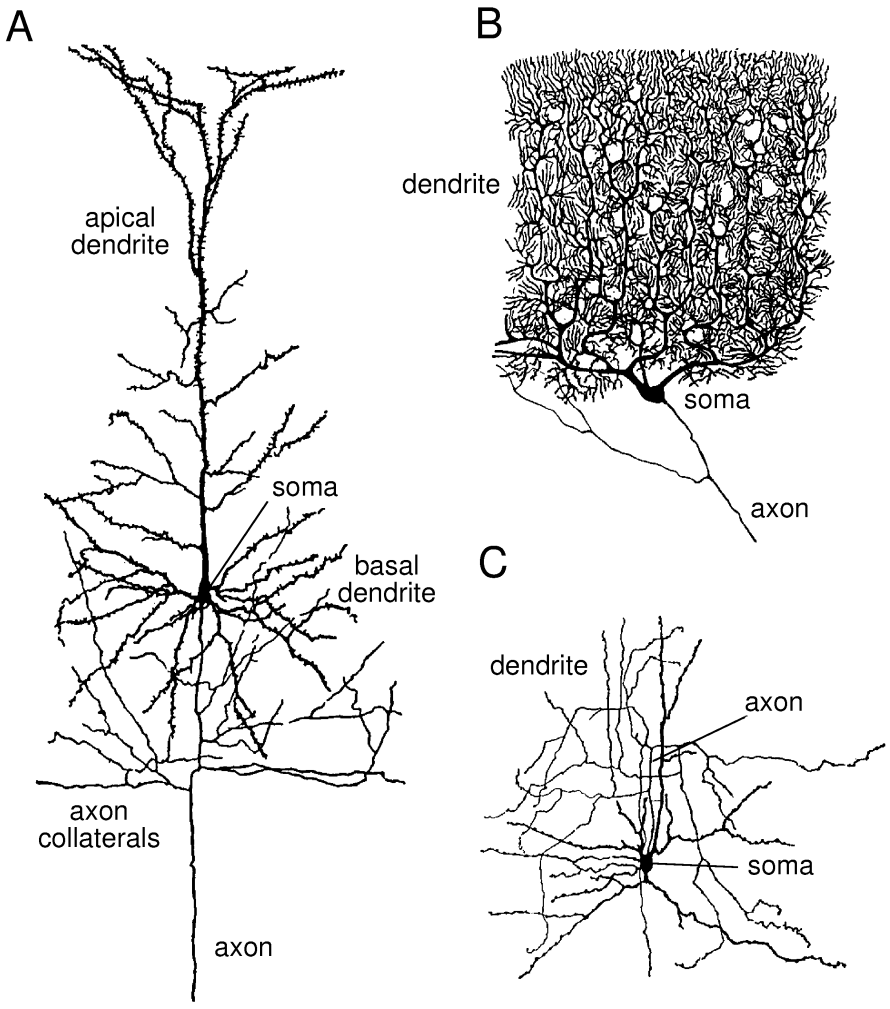


Figure 2.1: The morphology of three neurons. A) A cortical pyramidal cell. B) A Purkinje cell of the cerebellum. C) A stellate cell of the cerebral cortex. Some important morphological specializations of neurons such as dendrites, soma, and axon are illustrated. See text for a detailed explanation of the figure. Used with permission from Dayan & Abbott (2005).

Figure 2.1 illustrates important morphological specializations of neuron. A neuron has a cell body called the **soma** that processes the incoming signals and converts them into output neural activation signals. The soma has a **nucleus** which contains the genetic material in the

form of DNA as in most types of cell. Receiving signals can be either excitatory or inhibitory. Excitatory signals make the neuron generate a spike and inhibitory ones tend to keep the neuron from firing. A type of extension of nerve cell called **dendrites** handles incoming signals generated by other neurons. Generated action potentials (spikes) are transmitted to other neurons through an extension of the cell body called the **axon** which is typically longer than the dendrites. The axon, also called the nerve fibre, is a special cellular extension that take electrical signals away from the cell body (Kandel et al., 2000; Dayan & Abbott, 2005; Trappenberg, 2010).

Synapses are specialized junctions between neurons that allow signal transmission from the axon terminals of the pre-synaptic neuron to the dendrites of the post-synaptic neuron. This transmission across the synaptic cleft is handled by diffusion of chemicals called neurotransmitters illustrated in Figure 2.3 (Dayan & Abbott, 2005; Trappenberg, 2010). The structure of synapse and its role for development, learning, and memory are detailed in chapter 3.

There are many different types of nerve cells in terms of differences in sizes, shapes, or electrochemical properties. For instance, the soma of a neuron can differ in diameter from 4 to 100 micrometres (Davies, 2002). Based on shape, neurons are classified into three large groups: unipolar, bipolar, or multipolar according to the number of processes that originate from the cell body taken from (Kandel et al., 2000).

2.2.2 The Membrane Potential

In all types of cells, nerve cells in particular, there is an electrical potential difference between the interior and the exterior of the cell. This is called the membrane potential or membrane voltage of the cell. The membrane potential V_m can be formulated as:

$$V_m = V_{in} - V_{out} \tag{2.1}$$

where V_{in} is the inside potential of the cell and V_{out} is the outside potential of the cell. This potential across the membrane typically ranges from -40 mV to -80 mV with respect to the outside of the cell (Kandel et al., 2000).

Ion concentrations inside the neuron are different from the concentration outside. This concentration difference generates an electrical potential which is called the reversal potential (also known as the Nernst potential) for the ion. If both outward and inward rates of ion movement are the same, the ion flux is in equilibrium. Although ions may move, the net current is zero. This electrical potential with the zero net ion flows across the membrane is called the Equilibrium potential (Trappenberg, 2010).

2.2.3 Ion Channels

Ion channels with selective ion pumps have a number of jobs to control movements of ions between inside and outside the cell. Those jobs include establishing a resting membrane voltage, shaping action potentials and other electrical signals across the membrane. Primarily involved ions in those processes of the neuronal membrane are Sodium Na^+, Potassium K^+, Calcium Ca^{2+}, and Chloride Cl^- (Hille, 2001; Dayan & Abbott, 2005; Trappenberg, 2010). Also, a schematic of the lipid bilayer section that forms the cell membrane and two ion channels is illustrated in Figure 2.2.

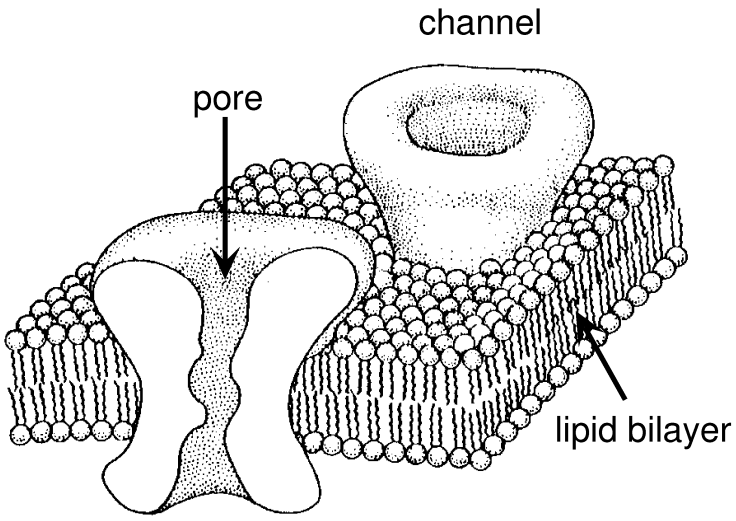


Figure 2.2: A schematic diagram of a section of the lipid bilayer that forms the cell membrane with two ion channels embedded in it. The membrane is 3 to 4 nm thick and the ion channels are about 10 nm long. Used with permission from Dayan & Abbott (2005).

From one side of the selectively permeable cell membrane to the other side, there are ion channels which can open and close in response to signals. Ion channels are selective allowing one type of ion to pass into the cell from the exterior, but block other types of ion. During the open times of those ion channels, the flow of ions causes the change in potential across

the membrane (Koch & Segev, 1999; Hille, 2001; Dayan & Abbott, 2005; Trappenberg, 2010). This potential change is detailed in the following section.

2.2.4 The Action Potential

A neuron receives signals from interconnected neurons. In response, the neuron can generate an action potential as output. The action potential occurs during a short period of time (1 ms) in which the electrical membrane potential of a cell rapidly rises and falls as seen in Figure 2.3. An action potential occurs when a neuron transmits information from the cell body to the axon terminal. It is also called a "spike" or an "impulse" (Koch & Segev, 1999; Gerstner & Kistler, 2002; Dayan & Abbott, 2005; Trappenberg, 2010).

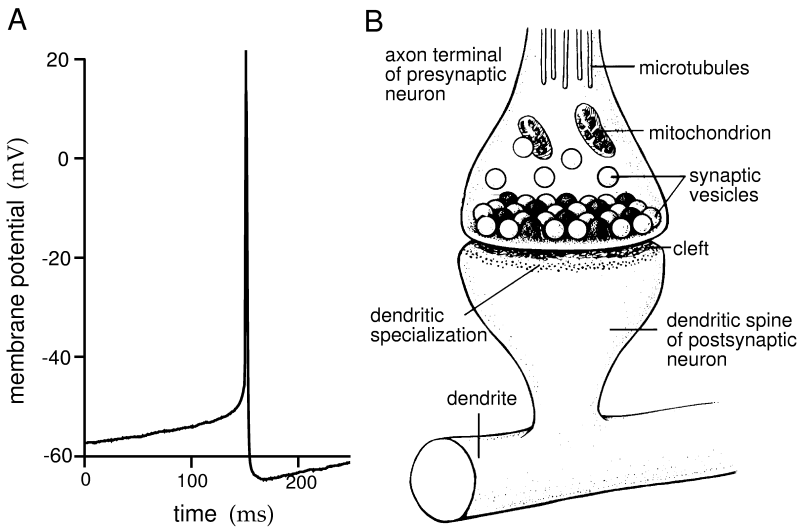


Figure 2.3: The action potential and axon-dendrite junction. See text for a detailed explanation of the figure. Used with permission from Dayan & Abbott (2005).

A single neuron may receive many thousands of input signals throughout its dendrites. The sum of all these signals (excitatory and inhibitory) determines whether the neuron fires or not. If the integration of all these potentials crosses a threshold potential, an action potential is generated (Gerstner & Kistler, 2002; Dayan & Abbott, 2005).

A prototypical form of action potential is shown in Figure 2.4. The mechanism of action potential has two components. First one, when positively charged sodium Na^+ ions flow into the neuron, the inside gets more positive until about +40 mV from −70 mV with respect to the surrounding media. The neuron becomes depolarized because of the positiveness of sodium ions. So **depolarization** is the rapid rise of the membrane potential as the first part of

an action potential until it reaches a maximum (Hille, 2001; Gerstner & Kistler, 2002; Dayan & Abbott, 2005; Trappenberg, 2010).

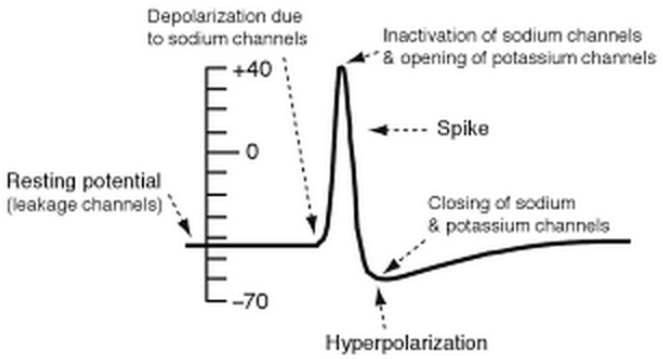


Figure 2.4: Typical form of an action potential based on Hodgkin-Huxley model. The course of the action potential is performed by two main currents, these are Sodium and Potassium. See text for a detailed explanation of the figure. Used with permission from Trappenberg (2010).

Following change can happen when potassium K^+ ions leave the axon which is the falling phase illustrated in Figure 2.4. Those channels open after a delay from depolarization because potassium channels move slower so the opening of them is taking longer. At the same time, sodium channels across the membrane close, allowing the potential to go back toward repolarization state. So, the sharp decrease of membrane potential from positive values to negative potentials is called **repolarization**. **Hyperpolarization** is the time period when the membrane potential dips even more negative than resting potential until about -90 mV (Hille, 2001; Dayan & Abbott, 2005)..

2.3 Relation between Artificial and Biological Neurons

An artificial neuron captures the essence of neural computation by simplification and abstraction analogy to biological systems (Gerstner & Kistler, 2002). The primary goal to model biological neural systems is to achieve improved results in Machine Learning (ML) tasks. The main difference of the artificial neuron model with real neurons is to leave out much of the details of how the biological neurons work using high-level descriptions (Gerstner & Kistler, 2002; Trappenberg, 2010). An Artificial Neural Network (ANN) consists of a set of artificial neurons with weighted links in order to perform the learning task well.

The standard abstraction used in Artificial Neural Network in order to represent biological neural components (Gerstner & Kistler, 2002; Dayan & Abbott, 2005; Trappenberg, 2010) is summarized as follows:

- *Nodes*: Processing units or artificial neurons.
- *Connections between nodes*: Weighted connections, is commonly called weights, in order to represent the relative strengths of the connections.
- *An integration (transfer) function*: The most common integration function is performed via the sum of weighted outputs from all upstream neighbours.
- *An activation function*: The activation function transforms the integrated incoming signals into an activation level for the neuron. This activation level is the output of the node.

The structural and functional mapping between ANNs and the biological neural networks (Gerstner & Kistler, 2002; Dayan & Abbott, 2005) is as follows:

- Biological neurons → Nodes.
- Synapses → Connections between nodes.
- Total depolarization at the cell membrane is the sum of depolarization as caused by each synaptic inputs → Integration function.
- The rate of action potential firing in the cell → Activation function.

The basic computational diagram of a common mathematical neuron model is demonstrated in Figure 2.5. This processing unit has a fixed number of input signals n through its dendrites from other units; each input has a synaptic weight, w_{ij} (w_{1j}, w_{2j}, w_{3j}, ..., w_{nj} in Figure 2.5), which is modified during training examples in order to better approximate the desired function response to input data. The artificial unit computes the weighted sum of all inputs x_1, x_2, ... x_n as net_j. Then, to the sum of results through an integration function φ is applied in order to determine the neuron's output o_j. If the final sum is above a threshold θ_j, the neuron produces an output along its axon based on its activation function (Rojas, 1996). The output

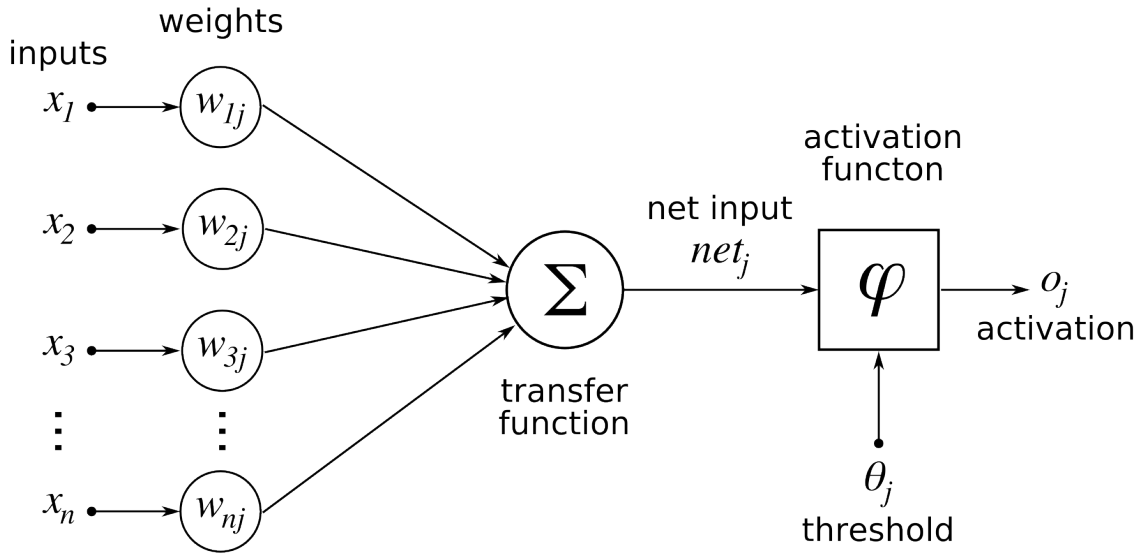


Figure 2.5: The artificial neuron model from human neuron, taken from Wikipedia (2016). See text for a detailed explanation of the figure and the notation used.

from each neuron is propagated to its connected downstream neurons, to serve as input.

The activation function determines the neuron model in ANNs. Although there are various types of activation functions such as step, ramp, sigmoid and Gaussian functions, processing units in many of ANNs use sigmoid function as the activation function. The formulation of neuron output is expressed in Equation 2.2. The axon connects via synapses to dendrites of following neuron units. This is the basic structure used in an ANN.

$$o_j = \varphi((\sum_{i=1}^{n} w_{ij}x_{ij}) + \theta_j) \tag{2.2}$$

where o_j is the output signal of the j^{th} neuron, x_{ij} is the i^{th} incoming signal to neuron j, w_{ij} is the synaptic strength of connection between the input i and neuron j, θ_j is the threshold value of j^{th} neuron, φ is the activation function. For the mathematical convenience, the threshold θ_j is represented with (+) sign in the activation formula. ANNs have continuous signals compared with the discontinuous nature of spiking in Spiking Neural Networks (SNNs) detailed in chapter 6. SNNs use rate based coding is detailed later.

2.4 History from Artificial Neural Network to Spiking Neural Network

Animals receive information of recognizing food or danger during interaction with the environment. Then they take proper actions after processing incoming signals. Mimicking this

mechanism of natural neuronal processing from a computing point of view is the paradigm known as Artificial Neural Networks (ANNs) (Gerstner & Kistler, 2002; Dayan & Abbott, 2005; Trappenberg, 2010).

One of the efficient methods to solve complex problems is to apply the divide-and-conquer strategy. A complex problem might be broken into simpler sub-problems, in order to be able to solve it directly. Furthermore, simple components might be assembled to generate a complex system (Bar-Yam, 1997). Applying networks, and particularly neural networks, is an important approach in order to deal with either a highly complex task, or where no algorithm exists to solve the specified task.

The term "Neural Network" is quite broad in terms of various approaches and models. Neural Network is an information processing paradigm which is different from the usual approach based on algorithmic computation. Neural Networks use learning techniques that are inspired by how biological nervous systems such as the brain or hippocampus learn. Then, they are applied to practical applications such as path navigation or object recognition to evaluate how good they are (Bohte et al., 2002b).

An Artificial Neural Network is an abstract mathematical paradigm that is inspired by biological nervous systems such as the brain (Bohte et al., 2002a, b). ANNs are composed of a large number of interconnected neurons. The neurons inside the ANN can be configured for a specific task to solve a pre-determined problem. During the learning process, the network can adjust the dynamics of synaptic connections such as delays or weights similar to biological systems.

Artificial Neural Networks have three classes based on their computational units (Maass, 1996; Gerstner & Kistler, 2002; Bohte et al., 2002b, a). The first generation of ANNs have perceptrons or threshold gates as computational units. Those neurons have a binary output when their total input reaches the threshold. The second generation of ANNs has an activation function which maps the inputs of a neuron into a continuous output. The sigmoid function is a commonly chosen activation function. Spiking Neural Networks (SNNs) as the third generation of Neural Network models inspired by features found in biology are able to encode and process the information using timing of individual spikes arriving at a neuron (Gerstner

& Kistler, 2002; Dayan & Abbott, 2005; Trappenberg, 2010). Although the first and second generation of ANNs can emulate many features of biological Neural Networks as plasticity, summation and thresholding, they cannot capture all of the biological features.

Individual computational neurons in each layer of SNNs can be mathematically modelled in a number of different ways. Those neuron models are reviewed in section 2.5 as "Full compartmental models" in subsection 2.5.1, "Hodgkin-Huxley model" in subsection 2.5.2, "Integrate-and-Fire model" in subsection 2.5.3, "Leaky-Integrate-and-Fire model" in subsection 2.5.4, "Spike Response Model" in subsection 2.5.5, "Izhikevich model" in subsection 2.5.6. Each of these simplified neuron models has different advantages and drawbacks.

Spiking Neural Networks are bio-inspired, adaptive, computationally powerful structures compared with conventional Artificial Neural Networks (Maass, 1996; Trappenberg, 2010). SNNs differ from conventional ANNs models as information is transmitted using spikes that is more biologically plausible than previous generations (Maass, 1996; Gerstner & Kistler, 2002). They process information encoded in the timing between neuron firings. As biological neurons communicate by receiving and transmitting pulses known as "spikes" to indicate their short and transient nature, then SNNs also carry information across synapses between other neurons in the network. It is believed that SNNs have richer dynamics as they can exploit the temporal domain to transmit data in the form of individual spikes (Gerstner & Kistler, 2002; Dayan & Abbott, 2005; Trappenberg, 2010).

The complexity and the computational power of artificial SNNs are restricted compared with the abilities of biological neural systems (Kasinski & Ponulak, 2006; Trappenberg, 2010). However, developing efficient learning techniques and recognizing information codes in SNNs are still open problems from a computational point of view. Therefore, various characteristics of SNNs such as plasticity algorithms and theoretical models could be considered to understand the ability of SNNs.

2.5 Spiking Neuron Models

Although artificial neuron models have made important progress in recent decades, they are highly abstract and still much simpler than biological neurons. Therefore, a spiking

neuron model is proposed by Maass (1996). It is a mathematical model to represent firing characteristics of neuron described in section 2.2. Neurons communicate with each other via action potentials or spike trains in Spiking Neural Networks similar to biological systems. Therefore, the following models propose various representations of real neurons through describing membrane potential and action potential generation in their model. Each neuron model has different simplifications. Some of them ignore most of the ion-channels; others simplify the structure of neuron. Details of different neuron models are discussed in this section. Most of the models are translated into the (Brian) simulator environment to visualize their behaviours.

2.5.1 Compartmental Models

One standard approach in order to construct a detailed neuron model is to break the neuron into a finite number of interconnected anatomical compartments. For this purpose, cable theory is extended for dendrites in 1959 using the Rall model (Rall, 1959). A simplified model for the compartments of dendrites, a soma, and an axon is illustrated in Figure 2.6.

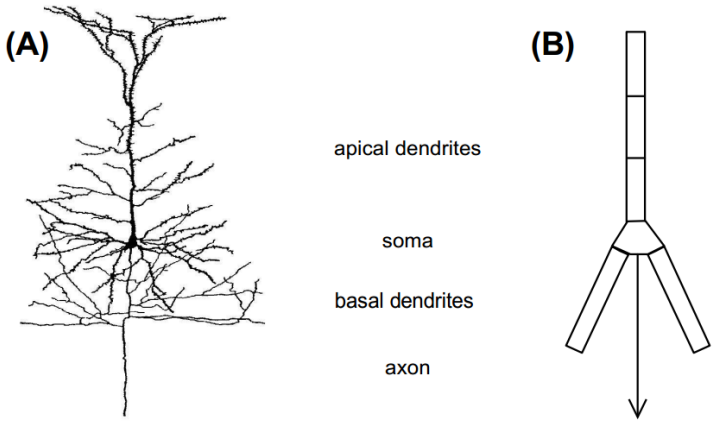


Figure 2.6: Compartmental model for pyramidal cell: (A) A pyramidal cell with dendrites, soma, and axon. (B) A simplified discrete compartmental model of the same neuron taken from Bower & Beeman (2003).

If analytical solutions for the voltage along a passive cable with suitable geometrical and electrical properties are applied to each compartment, various solutions can be constructed based on the variations of compartmental features. However it is difficult to find a solution for the membrane potential analytically (Abbott et al., 1991).

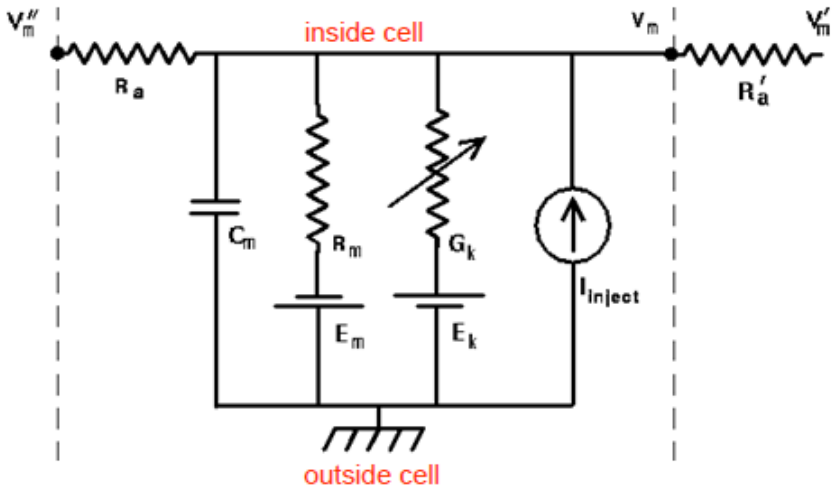


Figure 2.7: The equivalent circuit for a generic neural compartment. Used with permission from Bower & Beeman (2003).

Each compartment is modelled with equivalent electrical circuit (Rall, 1959). Figure 2.7 shows a circuit diagram for a generic neural compartment. With the appropriate differential equations of compartments, the behaviour and the interactions of compartments can be formulated. Although multi-compartmental models incorporate the complexity of real membrane behaviours, they are not computationally tractable. However, single compartmental models as class of point-neuron models can describe the neuron's membrane potential by a single compartment by ignoring spatial variations. Those models can reproduce accurately the complex dynamics of real neurons.

In Figure 2.7, V_m is the membrane potential at a point inside the compartment, C_m is the membrane capacitance, $V_m^{'}$ and $V_m^{''}$ are membrane potentials of adjacent compartments, $R_a^{'}$ and $R_a^{''}$ are resistances along the dendrite, R_m is the leakage resistance, E_m is the equilibrium potential.

2.5.2 Hodgkin-Huxley Model

Alan Lloyd Hodgkin and Andrew Fielding Huxley, from 1942 to 1952, developed a mathematical model to describe the electrical behaviour of the membrane through action potential generation in the giant axon of the squid (Hodgkin & Huxley, 1952). This model belongs to conductance-based compartment models as a type of compartmental models described in previous section. This model is one of the most common mathematical frameworks that formulate how action potentials in neurons are generated and transmitted. The Hodgkin-

Huxley equations are also discussed in more recent studies (Moore & Ramon, 1976; Hille, 2001).

The Hodgkin-Huxley model assumes that the electrical activity of the axon is primarily due to the movement of Na^+ and K^+ ions across the membrane. Therefore, the model contains Na^+ channels, K^+ channels and leakage channels for other ions. The leakage channels with relatively low conductance are mainly responsible for the resting membrane potential. The Na^+ and K^+ ion channels with their voltage-dependent conductance across the membrane are responsible for generating the action potential (Moore & Ramon, 1976; Kandel et al., 2000; Hille, 2001; Dayan & Abbott, 2005; Trappenberg, 2010).

Based on Hodgkin and Huxley's previous works, they proposed an equivalent electrical circuit of nerve membrane as illustrated in Figure 2.8 (Hodgkin & Huxley, 1952). The cell membrane of the neuron acts as a capacitor since it separates intracellular and extracellular ion regions. Once the potential of the capacitor exceeds the threshold voltage, charge is released. The period of capacitor discharge time is analogous to the refractory time of the neuron membrane once it has fired. The Hodgkin-Huxley model assumes that the membrane consists of Na^+ channels, K^+ channels and a leakage channel for other ions. The embedded protein channels behave like resistors with a driving force given by the difference between the membrane potential, $\mathrm{V} = V_m - E_{ion}$ and the reversal potential of the channel. Sodium and potassium ion channels are represented by the conductance R_{Na} and R_K, respectively. The conductance R_L represents the leakage channel.

Before demonstrating more recent models of action potential generation, the Hodgkin-Huxley model is studied to describe the change of electric charge across the neuron. Hodgkin-Huxley equations can be mathematically summarized by a set of coupled differential equations (Hodgkin & Huxley, 1952).

The fundamental differential equation relating the change in membrane potential to the currents flowing across the membrane is derived from the electrical circuit illustrated in Figure 2.8 (Hodgkin & Huxley, 1952) as:

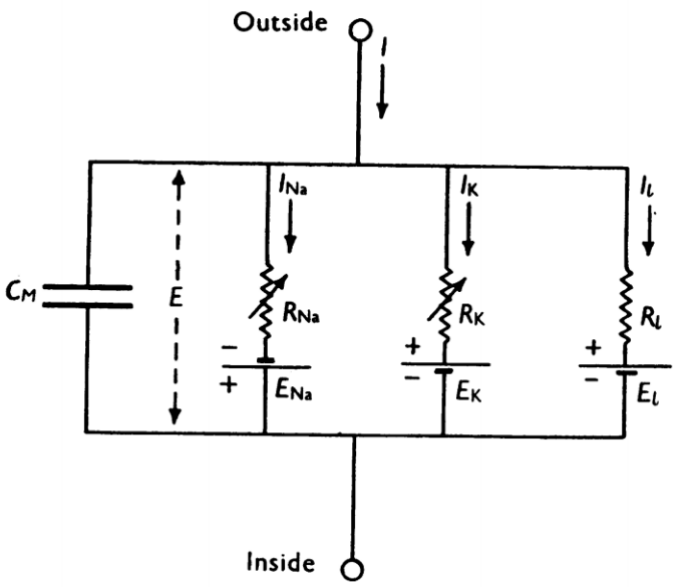


Figure 2.8: The proposed equivalent electrical circuit with three channels by Hodgkin-Huxley to describe nerve excitation taken from Hodgkin & Huxley (1952).

$$\begin{aligned} I_{ext} &= I_C + I_{ion} \\ &= C_m \frac{dV_m}{dt} + I_{ion} \end{aligned} \tag{2.3}$$

where I_C at time t is the capacitive current flowing through the lipid bilayer, I_{ext} is an externally applied current, such as might be introduced through an intracellular electrode with ionic current I_{ion} as

$$I_{ion} = I_{Na} + I_K + I_{leak} \tag{2.4}$$

where I_{ion} is ionic currents from the flow of Na^+ ions I_{Na}, K^+ ions I_K and other unspecified ions primarily Chloride ions as leak current I_{leak} as shown in Figure 2.8. Hodgkin and Huxley point that the ionic movement during action potential could be broken down primarily into movement of Na and K^+ ions (Hodgkin & Huxley, 1952).

Referring to the electric circuit above, the relation between the ionic currents and the conductance is described by Ohm's law $I = gV$ according to Hodgkin and Huxley as

$$I_i = g_i(V_m - E_i) \tag{2.5}$$

where E_i is the individual equilibrium potential of the i-th ion channel (i = Na^+, K^+, leak), V_m is the membrane voltage and g_i are the voltage dependent ionic conductance (Hodgkin & Huxley, 1952).

Hence, the total membrane current is obtained by Kirchhoff's conservation of current law via substituting Equation 2.4 and Equation 2.5 as:

$$\begin{aligned} Iext &= I_C + I_{ion} \\ &= I_C + I_{Na} + I_K + I_{leak} \\ &= C_m \frac{dV_m}{dt} + g_{Na}(V_m - E_{Na}) + g_K(V_m - E_K) + g_{leak}(V_m - E_{leak}) \end{aligned} \tag{2.6}$$

where parameters are summarized in Table 2.1. g_{leak} is the "leak" conductance, including all other conductance apart from Na^+ and K^+. The neuron membrane is able to store electrical charge and separate potential from the inside and the outside of the cell. C_m is the membrane capacitance. The total conductance of the membrane, the inverse of membrane resistance g_m ($g_m = 1/R_m$), take into account all parallel conductance.

Parameter	Unit	Explanation
I	[mA/cm^2]	Total membrane current per unit area
C_m	[uF/cm^2]	Membrane capacitance per unit area
V_m	[mV]	Membrane potential
E_{Na}	[mV]	Nernst (reversal) potential for sodium
E_K	[mV]	Nernst (reversal) potential for potassium
E_{leak}	[mV]	Nernst (reversal) potential for leakage ions
g_{Na}	[S/cm^2]	Sodium conductance per unit area
g_K	[S/cm^2]	Potassium conductance per unit area
g_{leak}	[S/cm^2]	Leakage conductance per unit area

Table 2.1: Parameters for the neuron models and synapses.

Three gate variables describe the conductance of potassium and the sodium channels are used in Hodgkin-Huxley model (Hodgkin & Huxley, 1952). The conductance of the potassium (K^+) and the sodium (Na^+) channels are empirically formulated as a power function of the probabilities n, m and h. n and m are the probability of activation gates being in the permissive state. h is the probability of inactivation gate being in the permissive state. Hodgkin and Huxley find that channel conductance can be expressed by products of gating

variables and maximum conductance (Hodgkin & Huxley, 1952) as:

$$g_K = \overline{g_K} n^4 \tag{2.7}$$

$$g_{Na} = \overline{g_{Na}} m^3 h \tag{2.8}$$

where $\overline{g_K}$ is the maximum potassium conductance, $\overline{g_{Na}}$ is the maximum sodium conductance, n, m and h are scalar gating variables $\in [0, 1]$. In the model, m represents activation of gates and h represents inactivation of gates in sodium channels. The potassium channel has only a single activation variable n and it is assumed that the probability of a potassium channel is opened with n^4 (Hodgkin & Huxley, 1952).

Summarizing the ionic currents in the Hodgkin-Huxley model (Hodgkin & Huxley, 1952) in standard notation, we have:

$$\begin{aligned} I_{ext} &= C_m \frac{dV_m}{dt} + g_{Na}(V_m - E_{Na}) + g_K(V_m - E_K) + g_{leak}(V_m - E_{leak}) \\ &= C_m \frac{dV_m}{dt} + \overline{g_{Na}} m^3 h(V_m - E_{Na}) + \overline{g_K} n^4 (V_m - E_K) + g_{leak}(V_m - E_{leak}) \end{aligned} \tag{2.9}$$

where parameters are summarized in Table 2.1. All potential and conductance parameters for Hodgkin-Huxley model are shown in Table 2.2.

i	E_i (mV)	$g_i(mS/cm^2)$
Na	115	120
K	-12	36
$leak$	10	0.3

Table 2.2: The parameters of the Hodgkin-Huxley equations. The membrane capacity is C = $1\mu F/cm^2$, (Hodgkin & Huxley, 1952).

The three gating variables m, n, and h evolve according to the following differential equations:

$$\dot{m} = \alpha_m(u)(1 - m) - \beta_m(u)m \tag{2.10}$$

$$\dot{n} = \alpha_n(u)(1 - n) - \beta_n(u)n \tag{2.11}$$

$$\dot{h} = \alpha_h(u)(1 - h) - \beta_h(u)h \tag{2.12}$$

with $\dot{m} = dm/dt$, $\dot{n} = dn/dt$ and $\dot{h} = dh/dt$. α and β are opening and closing rates for each channel. All gating variables based on empirical observations are shown in Table 2.3.

i	$\alpha_i(u/mV)$	$\beta_i(u/mV)$
m	(0.1-0.01u)/[exp(1-0.1u)-1]	0.125exp(-u/80)
n	(2.5-0.1u)/[exp(2.5-0.1u)-1]	4exp(-u/18)
h	0.07exp(-u/20)	1/[exp(3-0.1u) + 1]

Table 2.3: Empirical functions of α and β by Hodgkin and Huxley : To represent the data of the giant axon of the squid, (Hodgkin & Huxley, 1952).

where $u(t)$ is the membrane potential. The values in Table 2.3 are derived by fitting curves to experimental data the mathematical formulation of the model.

The dynamics of a Hodgkin-Huxley model in response to an applied input current is demonstrated in Figure 2.6 based on the computational simulation. Used parameters for the simulation are summarized in Table 2.2 and in Table 2.3.

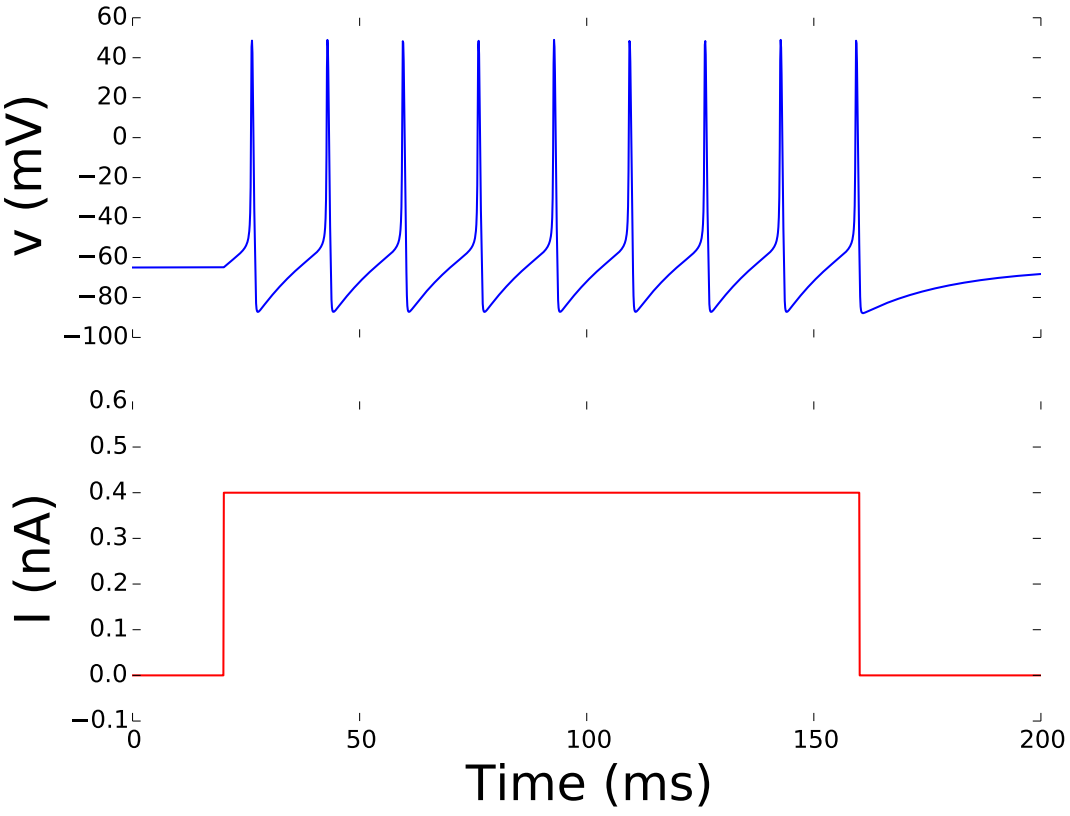


Figure 2.9: Simulation of a Hodgkin-Huxley neuron in response to an external input current. It generates regular spikes. The 200 ms time-course of the membrane potential $V_m(t)$ (top) during the current trajectory (bottom) switches from 0 nA (less than threshold voltage) at 20 ms to +0.4 nA (exceeding the threshold voltage) at 160 ms. All neuron parameters are summarized in Table 2.2 and Table 2.3.

2.5.3 Integrate-and-Fire Model

While the Hodgkin-Huxley model accurately reproduces the shape of action potentials, refractory period, rest response and repetitive firing, the model is computationally complex and nonlinear. Hence, simplified (lower dimensional and mathematically tractable) neuron models are proposed with a higher level of abstraction. One of the most widely used models for analysing the behaviour of nerve cells is the Integrate-and-Fire neuron model which is developed by Louis Lapicqu in 1907 (Abbott, 1999). The model considers a neuron which only emits a spike if the total excitation is sufficiently large.

Membrane time constant or membrane decay constant represented by the symbol τ_m is derived by the product of the membrane resistance R_m and membrane capacitance C_m. Membrane time constant τ_m is the time for the membrane potential to fall from the resting to a fraction of $1 - 1/e$, or 63%, of its final value in the charging (Nicholls et al., 1992). It is a measure of how long it takes the membrane to charge or discharge. It shows the neuron remembrance for inputs. Larger time constants cause longer memory and shorter time constants forget inputs quickly. The membrane time constant τ_m of different real neurons can vary between 10 and 100 ms (Koch et al., 1996).

The neuron membrane acts like a capacitor because the insulator is not perfect, the charge is slowly leak through the cell membrane and they are therefore leaky (Gerstner et al., 2014). However, the model of Integrate-and-Fire neuron does not represent this biological phenomena so the model is commonly named non-leaky. Unlike Leaky-Integrate-and-Fire (see section 2.5.4), the leakage of the membrane is not represented through the summed contributions to the membrane voltage decay with membrane time constant τ_m (Abbott, 1999; Naud et al., 2008; Gerstner et al., 2014). If this decay over time is neglected, the model becomes a perfect integrator as in Integrate-and-Fire (Gerstein & Mandelbrot, 1964). However, if there is a below-threshold incoming signal at some time, the model retains that voltage until it fires again. This causes no time-dependent memory of previous values of membrane potential.

This Integrate-and-Fire (IF) model is developed in terms of Neural Network dynamics (Hill, 1936; Stein, 1965; Geisler & Goldberg, 1966; Tuckwell, 1988). The state of neuron is expressed by its membrane potential in the Integrate-and-Fire model. Equivalent circuit diagram of IF can be seen in Figure 2.10. The circuit consists of a capacitor C_m in parallel with a switch and

a current $I(t)$ (Gerstner & Kistler, 2002). The switch closes when the threshold is achieved, then resets after the refractory period.

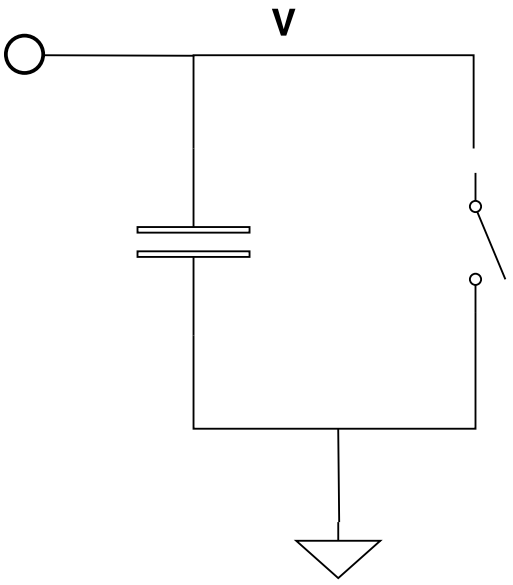


Figure 2.10: Equivalent circuit diagram for the IF neuron model.

The IF model is described mathematically by:

$$\frac{dV_m}{dt} = \frac{1}{C_m} I(t) \tag{2.13}$$

$$\text{when} \quad V_m(t) > V_{th} \quad \text{then} \quad V_m(t) \rightarrow V_{reset} \tag{2.14}$$

where C_m is the total capacitance of the membrane, $I(t)$ is the applied input current to the membrane. The model considers the neuron membrane as a capacitor C_m. The firing frequency of the IF model increases linearly with $I(t)$. When the membrane potential V_m reaches a constant threshold V_{th}, an output spike occurs and the voltage is reset to V_{rest} the resting potential as illustrated in Figure 2.11. Then, the neuron is inactivated for a period of time corresponding to the refractory period t_{ref}. It causes the neuron to enter a state of hyperpolarisation. Inactivity in the sodium channels and a lag in the closing of potassium channels are the reasons for refractory time. By including the refractory time t_{ref} in models, the firing frequency of a neuron can be limited by preventing it from firing during the refractory period. Figure 2.11 has no refractory period in the simulation parameters. Firing times are defined iteratively as:

$$t_i^n = [t | V_m(t) \geq V_{th}; t \geq t_i^{n-1}] \tag{2.15}$$

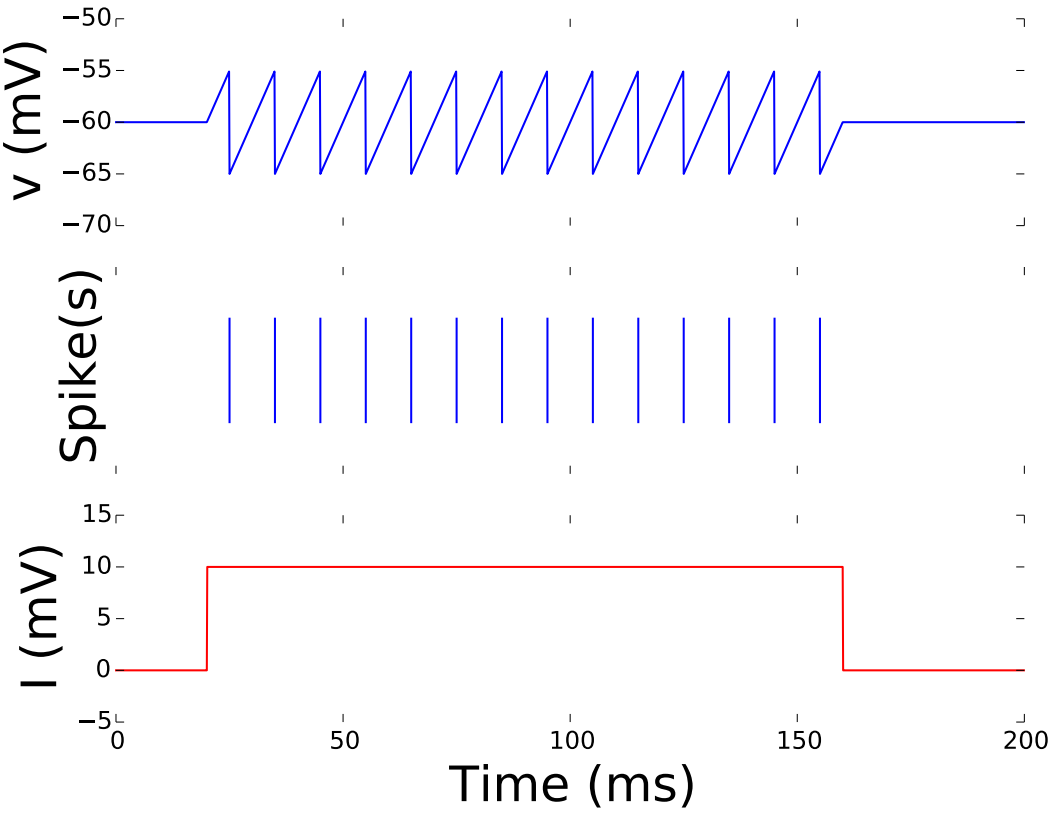


Figure 2.11: Simulation of a single IF neuron by an external input current. The 200 ms time-course of the membrane potential $V_m(t)$ (top) during the current trajectory (bottom) switches from 0 mV (less than threshold voltage) at 20 ms to +10 mV (exceeding the threshold voltage) at 160 ms. It has no refractory period $t_{ref} = 0$. Decay constant is $\tau_m = 10ms$. Middle graph shows where spikes are occurred for the membrane potential in the simulation. All neuron parameters are demonstrated in Table 2.4.

In summary, the non-leaky IF model as a perfect integrator is described by neglecting the membrane potential decay. However, it is not a particularly realistic way to model neuron behaviour although it is the simplest spiking neuron model to implement. Therefore, the more plausible model LIF is discussed in the following section.

2.5.4 Leaky-Integrate-and-Fire Model

In this section, another highly simplified version of the Hodgkin-Huxley model, the Leaky-Integrate-and-Fire (LIF) neuron model is reviewed. LIF is an improvement to the IF model and it is more biologically plausible model (Abbott, 1999; Gerstner & Kistler, 2002). In this model the cell membrane is represented by a simple RC electrical circuit as in Figure 2.12. In a LIF neuron, the potential decays exponentially which is more biologically plausible compared with (non-leaky) IF model because synaptic signals decay rapidly in a real neuron. The

memory problem in the Integrate-and-Fire model is also eliminated by adding a leak current component (Burkitt, 2006).

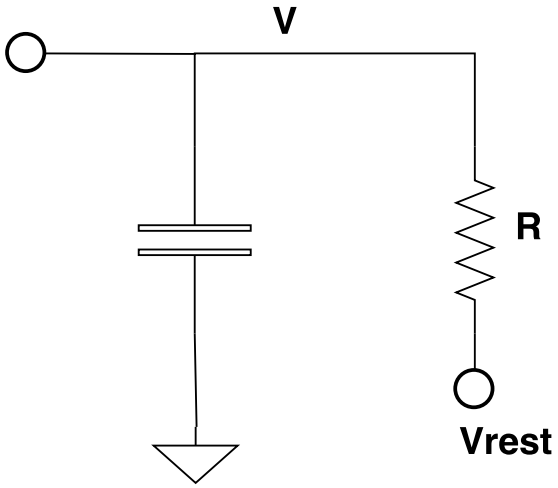


Figure 2.12: Equivalent circuit diagram for the LIF neuron model.

Equivalent circuit diagram of LIF can be seen in Figure 2.12. The model of the neuron membrane potential consists of a resistor and capacitor in parallel, representing the conductivity of the membrane and the leakage capacitance, respectively.

$$C_m \frac{dV_m(t)}{dt} = I(t) - \frac{V_m(t) - V_{rest}}{R_m} \tag{2.16}$$

or equivalently by substituting:

$$\frac{dV_m(t)}{dt} = -\frac{1}{\tau_m}(V_m(t) - V_{rest}) + \frac{1}{C_m} I(t) \tag{2.17}$$

$$\text{when} \quad V_m(t) > V_{th} \quad \text{then} \quad V_m(t) \rightarrow V_{reset} \tag{2.18}$$

where R_m is the membrane resistance and C_m the membrane capacitance. The model considers the neuron membrane as a capacitor C_m in parallel with a resistor R_m subject to a synaptic charging current $I(t)$. The constant current required to reach threshold is $I_{th} = V_{th}/R_m$. Therefore, the cell fires when I exceeds I_{th}. Immediately after the membrane voltage reaches the firing threshold V_{th}, the neuron generates a spike, then the voltage is reset to the resting potential V_{rest} as illustrated in Figure 2.13. The simulation in Figure 2.13 has no refractory

period $t_{ref} = 0$. The firing frequency can be formulated by Koch & Segev (1999) as:

$$f(I) = \begin{cases} 0, & \text{if } I \leq I_{th} \\ [t_{ref} - R_m C_m log(1 - \frac{V_{th}}{IR_m})]^{-1}, & \text{elsif } I > I_{th} \end{cases} \tag{2.19}$$

where R_m is the membrane resistance, C_m is the membrane capacitance, I_{th} is the threshold current, V_{th} is the threshold membrane potential and t_{ref} is the absolute refractory time.

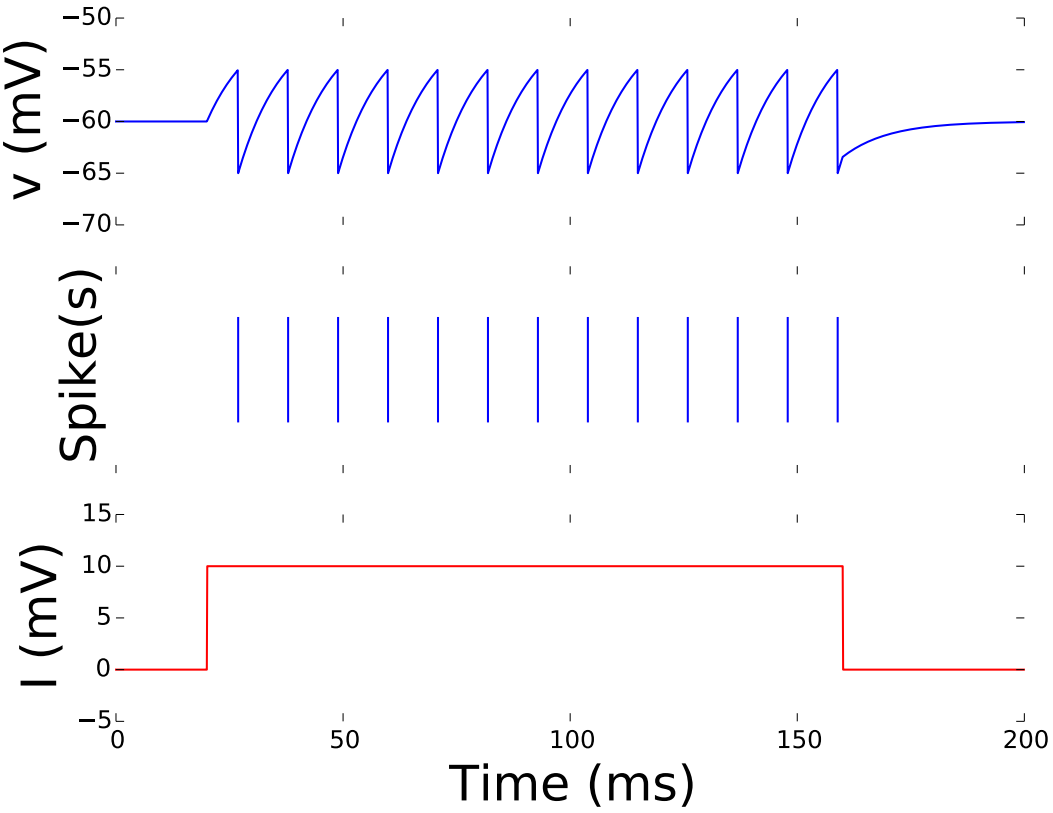


Figure 2.13: Simulation of a single LIF neuron by an external input current. The 200ms time-course of the membrane potential $V_m(t)$ (top) during the current trajectory (bottom) switches from 0 mV (less than threshold voltage) at 20 ms to +10 mV (exceeding the threshold voltage) at 160 ms. It has no refractory period $t_{ref} = 0$. Decay constant is $\tau_m = 10$ ms. Middle graph shows where spikes are occurred for the membrane potential in the simulation. All neuron parameters are demonstrated in Table 2.4.

So far an isolated neuron that is stimulated by an external current I_{ext} has been considered. In a more realistic situation, the input current is generated by pre-synaptic neurons. This type of Leaky-Integrate-and-Fire model as a part of a larger Spiking Neural Network is used in our experiments.

In this framework, the total input current to neuron j is the sum over all current pulses as:

$$I_j(t) = \sum_i \sum_f w_{ij} \kappa(t - t_i^f) \tag{2.20}$$

where w_{ij} is the synaptic efficacy between neuron i and neuron j. The f^{th} firing time of the pre-synaptic neuron i is denoted by t_i^f. The function $\kappa(t)$ describes the form of post-synaptic response. $\kappa(t)$ can be an α function as defined in section A.2, δ function as defined in section A.1 or other types in order to define various Post-synaptic Potentials (PSPs) which are the changes in the membrane potential of the post-synaptic terminal of a synapse. $\kappa(t - t_i^f)$ is a causal filter that only takes into account spikes $t_i^f \leq t$. Each afferent pulse causes a change in the potential of post-synaptic neuron. The level of the PSP is modulated by the synaptic efficacy w_{ij}. Finally, the total received current into the neuron is a summation over all afferents.

Table 2.4 summarizes the neuron parameters which are used in experiments in following sections. Here $\tau_m = R_m * C_m$ denotes the decay time constants of membrane integration. We choose $\tau_m = 10$ ms by considering $R_m = 1$ MΩ and $C_m = 10$ nF as the membrane resistance and capacitance, respectively. Typical potential values are summarized as V_{rest} = -60 mV, V_{reset} = -65 mV, V_{th} = -55 mV, the resting potential, the reset potential, and threshold potential, respectively. There is no refractory time $t_{ref} = 0$ ms. These parameters in Table 2.4 are typical values (Gerstner & Kistler, 2002).

Name	Value	Unit
V_{th}	-55	mV
V_{rest}	-60	mV
V_{reset}	-65	mV
τ_m	10	ms
t_{ref}	0	ms

Table 2.4: Model parameters for the Leaky-Integrate-and-Fire simulation.

In summary, LIF is a more plausible model than IF as it includes a leak current. It is proven that it is useful in addressing many of the questions about how neurons and neural architectures process information. Therefore, LIF is one of the most commonly used spiking neuron models with its simple architecture for analysing the behaviour of neural systems

(Burkitt, 2006). The model is also used during our further experiments.

2.5.5 Spike Response Model

The Spike Response Model (SRM) as defined by Kistler et al. (1997) is another approach and more straightforward to implement compared with the differential equation neuron models. The SRM is a generalization of the LIF model by defining the membrane potential V_j as an integral over time t. Although, parameters in the leaky IF model make voltage dependent, parameters in the SRM depend on the time $(t - \hat{t})$ since the last output spike (Gerstner & Kistler, 2002).

Suppose that the last spike time of neuron j is $\hat{t}$:

$$\hat{t} = max\{t^{(f)} < t\} \tag{2.21}$$

The evolution of the membrane potential V_j as a function of time t is described (Kistler et al., 1997) by:

$$V_j(t) = \eta(t - \hat{t}) + \sum_i w_{ij} \sum_f \in_{ij} (t - \hat{t}, s) + \int_0^\infty \kappa(t - \hat{t}, s) I_{ext}(t - s) ds \tag{2.22}$$

where η is a reset kernel which models the reset after a spike, $I_{ext}(t)$ is external current with a filter $\kappa(s)$ which is the response of the membrane potential to an incoming pulse. $t^{(f)}$ are the times of pulses from pre-synaptic neurons i and w_{ij} is the synaptic efficacy between pre-synaptic neuron i and post-synaptic neuron j. Each pre-synaptic spike evokes a Post-synaptic Potential (PSP) in post-synaptic neuron j and the trajectory of the PSP is expressed by a function $\in_{ij} (t)$ where $s = t - t^{(f)}$.

A threshold condition is taken into account for spike generation:

$$\text{when} \quad V_j(t) > V_{th} \quad \text{and} \quad \dot{V}_j(t) > 0 \quad \text{then} \quad \hat{t} = t \tag{2.23}$$

If, after the summation of the effects of all incoming pulses, the membrane potential V_j achieves the threshold potential V_{th}, the post-synaptic neuron fires. In contrast to the LIF model, the threshold parameter V_{th} is not fixed but it depends on the time since the last

generated spike as:

$$V_{th} \rightarrow V_{th}(t - \hat{t}) \tag{2.24}$$

The kernels can be expressed as:

$$\eta = (E_{reset} - E_{rest})e^{-t/\tau}H(t) \tag{2.25}$$

$$\kappa = 1/C_m e^{-t/\tau}H(t) \tag{2.26}$$

where $H(t)$ is Heaviside step function described in section A.4 and membrane time constant $\tau_m = C_m/g_L$. Reset kernel function η determines the form of action potential and the after-hyperpolarisation potential. If there is not any input (at rest), the membrane potential equals the resting voltage value, E_{rest}.

A special case of the SRM is simplified SRM_0 modelled by neglecting the dependence of κ and ϵ. The voltage of a post-synaptic neuron is described as:

$$V_j(t) = \eta(t - \hat{t}) + \sum_i w_{ij} \sum_f \in_{ij} (t - \hat{t}, s) + V_{rest} \tag{2.27}$$

To sum up, the SRM is an extension of the LIF model. The SRM implementation of LIF is used in some of our following experiments in chapter 8 and chapter 9 due to its flexibility in defining the neuron response and the synaptic response.

2.5.6 Izhikevich model

Although the Hodgkin-Huxley model is still the foundation for most models of biological neurons today, it is computationally expensive to build and simulate especially in case of large neural networks. The Izhikevich model (IM) (Izhikevich, 2003) is a good compromise between biological plausibility and computational efficiency. The dynamics of a single Izhikevich spiking neuron can be formulated via the following set of ordinary differential equations Equation 2.28.

$$\begin{aligned} \dot{v}(t) &= 0.04v^2(t) + 5v(t) + 140 - u(t) + I(t) \\ \dot{u}(t) &= a(bv(t) - u(t)) \end{aligned} \tag{2.28}$$

with the after-spike resetting of the form in Equation 2.29. $v(t)$ represents the membrane voltage of a neuron with $\dot{v}(t) = dv(t)/dt$ as the derivative of function with respect to time. $u(t)$ represents a recovery variable with $\dot{u}(t) = du(t)/dt$. The coefficients of 0.04, 5 and 140 have been chosen by Izhikevich (2004) because those constants can be used to simulate around 20 different biologically meaningful spiking behaviours.

$$\text{if } v(t) \geq 30 \text{ mV, then } \begin{cases} v(t), & v(t) \leftarrow c \\ u(t), & u(t) \leftarrow u(t) + d \end{cases} \tag{2.29}$$

where *v(t)* and *u(t)* are membrane voltage and recovery function respectively. a, b, c and d are adjustable scaler parameters of the Izhikevich model as explained below (Izhikevich, 2004).

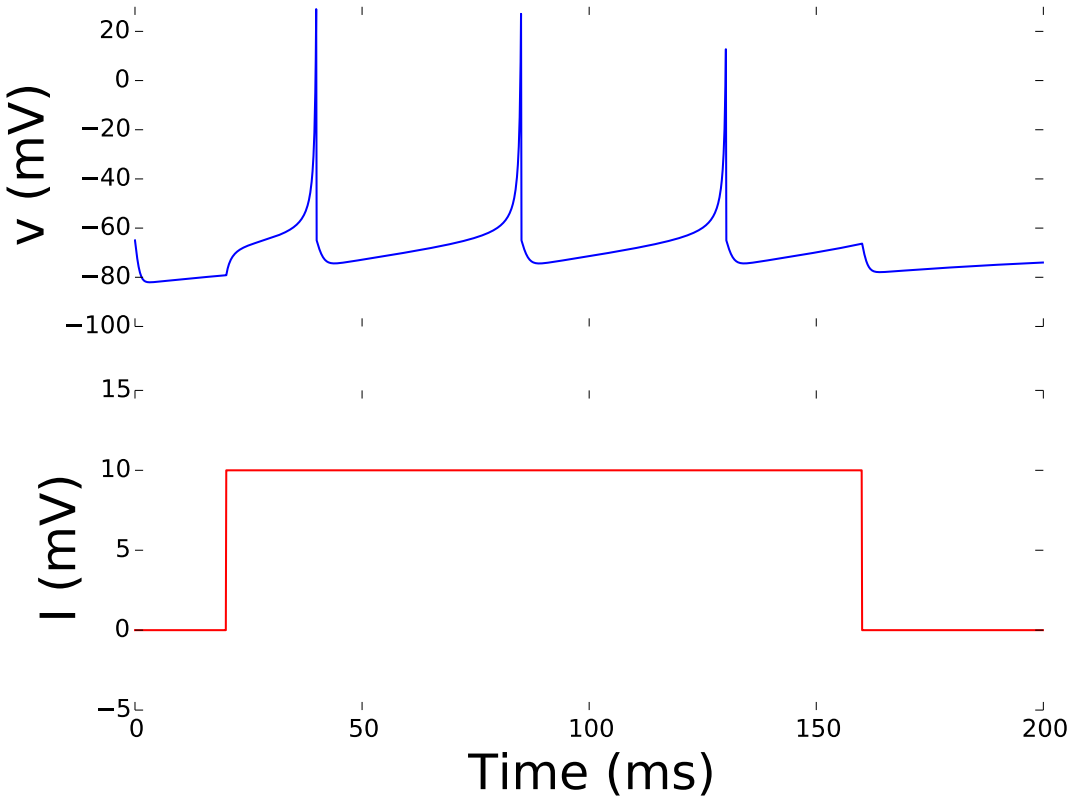


Figure 2.14: Simulation of a single Izhikevich model neuron by an external input current. It is generating regular spike as in Izhikevich model (Izhikevich, 2003). The 200 ms time-course of the membrane potential $V_m(t)$ (top) during the current trajectory (bottom) switches from 0 mV (less than threshold voltage) at 20 ms to +10 mV (exceeding the threshold voltage) at 160 ms. Initial membrane potential is set to V_{rest}. All neuron parameters are given in Table 2.5.

When the neuron membrane voltage reaches the level of +30 mV or above, the spiking

a ($1/sec$)	b ($1/sec$)	c ($volt$)	d ($volt/sec$)	V_{th} ($volt$)
0.02/ms	0.2/ms	-65 mV	8 mV/ms	30 mV

Table 2.5: Model parameters for the Izhikevich model simulation.

condition is occurred, and a spike is emitted illustrated in Figure 2.14. Then the membrane potential (v) and the recovery variable (u) are reset as in Equation 2.29.

The parameters of the model are:

a $\rightarrow$ The time scale of the membrane recovery variable *u(t)*. Smaller values correspond to slower recovery. A typical value for a is *0.02*.

b $\rightarrow$ The sensitivity of the recovery variable *u(t)* to the sub-threshold fluctuations of the membrane potential *v(t)*. It determines the resting potential between -70 mV and -60 mV in the model. Greater values couple *v(t)* and *u(t)* more strongly resulting in possible sub-threshold oscillations and low-threshold spiking dynamics. A typical value for b is *0.2*.

c $\rightarrow$ The after-spike reset value of the membrane potential *v(t)* is caused by the fast high-threshold K^+ conductance. A typical value for c is *-65 mV*.

d $\rightarrow$ After-spike reset of the recovery variable *u(t)* is caused by slow high-threshold Na^+ and K^+ conductance. A typical value for d is *2*.

In summary, the Izhikevich neuron model embeds many observed patterns of neurons in two dimensional differential equations. Although this reduces the bio-physical accuracy compared with the Hodgkin-Huxley model, the methodology of Izhikevich gives computational advantages (Izhikevich, 2007a). The model is implemented to compare the performance of learning algorithms through other neuron models.

2.5.7 Discussion

Different dynamical models of biological neuron are presented. Example response of each model to constant current input is illustrated by solving their dynamical equations numerically using (Brian) SNN simulator (detailed in section 2.7).

Hodgkin-Huxley equations based on the electrophysiological experiments on the giant axon of the squid are the core mathematical framework for neural modelling. The model describes ionic movements involved in generation of action potential. The parameters of the Hodgkin-Huxley model are biophysically meaningful and measurable. Although Hodgkin-Huxley as a conductance-based model is able to reproduce many of the phenomena observed in real axons, it has higher dimensionality. The model is computationally expensive, especially for the large scale simulations of SNNs (Izhikevich, 2004; Gerstner et al., 2014).

The Integrate-and-Fire neuron derived from the Hodgkin-Huxley model can fire tonic spikes at fixed frequency, and it is an integrator (Izhikevich, 2004). It cannot have phasic spiking, bursting of any kind, or having other properties of cortical neurons because it has a single variable (one-dimensional) in order to control the action potential generation (Izhikevich, 2004; Naud et al., 2008; Gerstner et al., 2014). On the other hand, the LIF model is quite popular because it is much more computationally tractable.

The LIF neuron as a type of IF neuron has high simplification compared to Hodgkin-Huxley model. The LIF model is described by a single first-order linear differential equation. In this model, an action potential is emitted when the value of the membrane potential crosses a fixed threshold value. It has less biophysical plausibility with decreasing computational cost compared to Hodgkin-Huxley model (Izhikevich, 2004; Naud et al., 2008; Gerstner et al., 2014). The LIF is improved model of IF by inserting a leak current which decreases the stored membrane potential over time. Hence, it is more realistic modelling of biologic neuron, as it does not hold electrical charge indefinitely.

The Spike Response Model is more intuitive to understand and more straightforward to implement in SNNs (Paugam & Bohte, 2012). Despite its simplicity, the SRM is more general than IF neuron models and is often able to compete with the Hodgkin-Huxley model for simulating complex neuro-computational properties. The Spike Response Model depends on the time since the last generated spike, whereas the LIF model is voltage dependent (Gerstner & Kistler, 2002; Gerstner et al., 2014). The dependence of the membrane potential upon the last spike reduces the responsiveness after the spike, increases the threshold after firing, and causes hyper-polarization spike after potential (Gerstner & Kistler, 2002). Although Leaky-Integrate-and-Fire is expressed with differential equations, the membrane potential in

SRM is described as an integral over previous times.

Izhikevich model (Izhikevich, 2003, 2004, 2007a) is described as a two-dimensional system of ordinary differential equations with four dimensionless parameters. The model can reproduce many different spiking and bursting behaviours observed in variety of cortical neurons by choosing different values of these parameters (Izhikevich, 2004). Although Izhikevich model has much less computational cost compared to the Hodgkin-Huxley model, it takes into account all possible information about ionic currents as Hodgkin-Huxley type models (Izhikevich, 2004).

One of the main purposes throughout the thesis is to investigate diverse plasticity paradigms relied on spike-timing in SNNs. Therefore, instead of the models with a large quantity of differential equations to mimic entire ion-channels, we use Leaky-Integrate-and-Fire model in order to reduce the amount of computation is performed. Despite its relative simplicity, the LIF model can reproduce quantitatively various aspects of neuronal dynamics (Abbott, 1999; Gerstner & Kistler, 2002; Burkitt, 2006). Also, we use SRM_0 the special case of the more general SRM as an extension of the LIF model (Gerstner & Kistler, 2002) during experiments because of its straightforward implementation.

Inside the SNNs as a third generation of NNs, those biologically-plausible models of neurons are used in order to bridge the gap between neuroscience and Machine Learning. General types of plasticity mechanisms considering with spiking networks are reviewed on the core of Machine Learning theory in the following section.

2.6 Supervised, Unsupervised, and Reinforcement Learning

Learning procedures are conventionally divided into three main types: supervised learning, unsupervised learning and Reinforcement Learning (Rojas, 1996; Fellous & Suri, 2003). Those types are important learning paradigms in the context of Machine Learning.

Supervised learning algorithms generate a classifier or predictor from a set of training examples by extracting the relationships and dependencies between the input and output features (Fellous & Suri, 2003; Kasinski & Ponulak, 2006; Ponulak & Kasinski, 2010). Calculated error measure through the comparison between actual and desired output used as a feedback signal.

Hence, supervised techniques aim to minimize their error function as in gradient descent search methods which minimize a mean square error between the desired output and the actual outputs in SNNs. Analogous to the classical Back Propagation (BP) algorithm in traditional NNs, SpikeProp algorithm is described in more detail in section 8.2 is one of the first supervised learning methods for SNNs (Schrauwen & Campenhout, 2004; Kasinski & Ponulak, 2006).

Unsupervised learning adapts its parameters based on the statistical properties of the input space despite using the guidance of performance evaluation in supervised learning (Bohte et al., 2002a; Fellous & Suri, 2003). Hence, it learns a structure in its input without involving target values. Unsupervised techniques try to discover the hidden patterns or features in the input data, or extract an optimal representation of the data, or reduce dimensionality of input data without getting any external teaching or feedback signal.

Unsupervised learning paradigm is particularly suitable for biologically motivated plasticities that use intuitive primitives like neural competition and cooperation between each other. Spike-based unsupervised mechanisms in SNNs allow for association/classification between inputs without class labels using biologically plausible architectures (Bohte et al., 2002a). For instance, Hebbian plasticity (described in more detail in section 3.3) is considered one particular class of unsupervised learning methods based on Machine Learning theory (Hertz et al., 1991; Gerstner et al., 2014). Synaptic efficacy between neurons in Hebbian learning is determined based on the correlations of pre-synaptic and post-synaptic firing activities (Hebb, 1949; Sutton & Barto, 1998; Brown & Milner, 2003; Johansen et al., 2014). Also, the STDP rule (detailed in section 3.4) based on Hebbian plasticity provides a biologically inspired mechanism in Spiking Neural Networks as an unsupervised learning scheme (Bi & Poo, 1998; Izhikevich, 2007b).

Reinforcement Learning (RL) performs the learning through maximizing its numerical values which represent a long-term objective (Kaelbling et al., 1996; Sutton & Barto, 1998; Farries & Fairhall, 2007). In RL theory, this is generally formalized with optimizing an unknown reward function (Kaelbling et al., 1996; Sutton & Barto, 1998) detailed in chapter 4. Each observation about the state of the environment helps to evaluate the goodness of present situation. Tasks are performed through explorations around the environment without having examples of correct behaviour. Actions are selected according to observed the resulting reward

from the previous interaction. RL techniques differ in the details of how the exploration is performed and how learning performance is calculated (Kaelbling et al., 1996; Sutton & Barto, 1998). In the recent decades, Temporal-Difference (TD) learning becomes one of the most popular RL algorithms proposed by Sutton & Barto (1998).

Supervised learning algorithms and Reinforcement Learning algorithms are optimization methods in Machine Learning. Both try to optimize performance estimates or measures of the Neural Networks. One of main differences from supervised learning is the goal in RL tasks is to find numerical value set which gives maximum reward unlike to predict the output values based on a given set of examples in supervised learning. Therefore, the aim is to maximize the sum of rewards in the long term at a single location instead of minimizing expected future prediction error over the entire sample space. Another key difference is that RL processes have no fixed distribution over training set unlike supervised learning tasks because RL agent is in charge of choosing values of each input through environment interactions.

After the review of plasticity types in the field of Machine Learning, we discuss how proposed plasticity rules, neuron models, and entire underlying mechanisms for SNNs can be transferred into available computer simulators. The motivation about the choice of simulator among other simulators is justified in the following section.

2.7 Simulators

Although simulation of biologically-inspired Spiking Neural Networks is generally a complex problem, various computer simulators are developed in order to describe and understand biochemical processes in neurons and different Neural Networks. These simulators provide various neuron models and biological neuron mechanisms in networks. These models can vary from very abstract mathematical neuron behaviours to detailed morphology of neurons.

Also, updating the simulation dynamics varies in SNN simulators. There are mainly two fashions: clock-driven (time-driven) and event-driven. In the clock-driven paradigm, objects/parameters are updated at every time point $t = 0, dt, 2dt, 3dt, ...$ This can be potentially very time consuming for same cases, because all parameters are updated at every time step along with the simulation time. For instance, updating synapses in contrast to neurons can

be less efficient strategy in clock-driven approach. However, in event-driven fashion, indicated objects/parameters are updated only at the times of events. For the synapse example, it is evaluated only on the arrival of a pre-synaptic spike rather than every time grid.

The models in section 2.5 can be simulated on any supported simulator such as NEURON (see section B.1), GENESIS (see section B.3), Nengo (see section B.4), Brian or on neuromorphic hardware (Brette et al., 2007). We present an overview of some of the available simulation environments for Spiking Neural Networks to implement our further experiments.

2.7.1 Brian

The Brian simulation package [1] is based on the programming language Python which allows flexible model definition by writing readable textual descriptions based on mathematical expressions (Goodman & Brette, 2008; Stimberg et al., 2014). The Python code for all simulations including neurons, synapses and network models based on Brian architecture is also available on the author's web page [2].

Brian is a clock-driven simulator. However, some object updates such as differential equations or synaptic parameters in Brian can be adjusted in an event-driven fashion as well. Although the default update is clock-driven, this could be changed if it is needed. For instance, the implementation of Spike-timing Dependent Plasticity (STDP) (see chapter 9) pre-synaptic and post-synaptic traces are much efficient with event-driven mechanism rather than clock-driven approach.

Neuron models are defined by differential equations in Brian. Connectivity model between neurons can be implemented by direct specifying, all-to-all or random connections. Those connections can have various custom parameters such as synaptic weights, delays, and conductance.

We would like to prepare a custom environment based on Brian through the construction of neuron models and network architectures. This gives us a chance to explore effects of various parameters through the simulation. Instead of using tool-specific language or GUI-specific as in NEURON, we would like to use more broad language as in Brian's text and equation

[1] http://www.briansimulator.org
[2] http://www.ozturkibrahim.com

based syntax on Python. Due to various scientific libraries in Python language, Brian is an important option to define new models based on existing models. Therefore, Python can be the middleware between Brian kernel and the numerical package for initializing and prototyping of neuron and synapse models according to complicated probability distributions. Brian simulator has intuitive syntax to start with computational neuroscience through ordinary mathematical notation of differential equations. It is a cross platform tool unlike the NEST simulator which is based on Linux. Most simulators have a high-level scripting interface to low-level neural simulators. However, Brian provides a convenient access to each low-level element in the architecture. In Brian, it is also straightforward to manipulate the neuron mechanisms and models in contrast with the NEST.

Brian uses vectorisation to overcome the lower performance of Python. Although it does not have significant overhead ratio for large networks, it has reasonable overhead ratio for small networks which are designed during our experiments. Some simulators such as NEURON and GENESIS are originally designed for detailed neuronal modelling at the ionic channel level. However, Brian is built specifically to run Spiking Neural Networks and is optimized with SNNs in mind.

Re-implementing the model in another language such as C++, Java or in an embedded platform is not a very difficult task like in NEURON and most of above listed simulators. This reimplementation can be for various reasons such as faster simulations, benchmark comparisons and so on. C++ implementation is already provided by Brian. For others, the SWIG [3] wrapper tool can be used to convert it into target language.

In summary, Brian allows to write equations in standard mathematical notation. It is well documented and it has an active support group. The simulator software is optimized for Spiking Neural Networks and allows multiple neuron models and plasticity features in the same simulation. Most objects and parameters are updated in either clock-driven fashion or event-driven approach. Brian is better suited for simulating a range of single cell models and customized plasticity rules. Although it provides some predefined neuron models such as Leaky-Integrate-and-Fire and Hodgkin-Huxley neurons, any custom neuron and plasticity models can be described straightforwardly. It is flexible and easily extensible. Entire simulations

[3] http://www.swig.org/

throughout the thesis are implemented in using Python.

2.8 Summary

In this chapter, basic concepts of the neuron electrophysiology are reviewed and mechanisms underlying the generation of action potentials in neurons based on the Hodgkin-Huxley theory are described. Then, the relation between biological and artificial neurons is presented with the structural and functional inspirations by neurobiological findings. A brief history from a standard Artificial Neural Network (ANN) to biological Spiking Neural Network (SNN) is provided. Then, three main types of learning procedures as supervised learning, unsupervised learning and Reinforcement Learning are discussed. Details of descriptive mathematical analysis of the compartmental approach, Hodgkin-Huxley model, Integrate-and-Fire model, Leaky-Integrate-and-Fire model, Spike Response Model and Izhikevich model are explained with illustrative figures. Finally, to simulate Spiking Neural Networks in software, various modern Spiking Neural Network simulators NEURON, NEST, GENESIS, Nengo and Brian are presented. The Brian tool based on the Python language is used during experiments in chapter 8 and chapter 9. The motivation about the choice of Brian tool among listed simulators is also justified.

Chapter 3

Learning Mechanisms in Spiking Neural Networks

Contents

3.1 Introduction . . . 73
3.2 Synapse . . . 75
3.2.1 Synaptic Transmission . . . 76
3.2.2 Synapse Model . . . 77
3.2.3 Synaptic Plasticity . . . 77
3.3 Hebbian Plasticity . . . 78
3.3.1 Long Term Potentiation (LTP) . . . 79
3.3.2 Long Term Depression (LTD) . . . 79
3.4 Spike-timing Dependent Plasticity (STDP) . . . 80
3.4.1 Modeling of STDP . . . 82
3.4.2 Weight Dependence . . . 86
3.4.3 Implementation of STDP . . . 87
3.4.4 Weight Bounds . . . 89
3.5 Homeostatic Plasticity . . . 89
3.5.1 Synaptic Scaling . . . 90
3.6 Dopamine Modulated Plasticity . . . 91
3.7 Summary . . . 93

3.1 Introduction

Since Spiking Neural Networks (SNNs) are based on spiking neuron models that are so close to the biological neurons, many of the biological principles can be applied to the networks. Various learning mechanisms, especially the biologically plausible plasticity rules, for Spiking

Neural Networks are introduced. This chapter considers how learning mechanisms can be incorporated into artificial networks of spiking neurons.

Spiking Neural Networks consist of nodes and the connections between them. The class of nodes includes spiking neurons and objects which enable stimulating or recording of neurons. A connection carries information from a sending node (pre-synaptic neuron) to a receiving node (post-synaptic neuron). Typically, a connection includes at least a synaptic weight and a synaptic delay. The weight determines the impact of the pre-synaptic signal on the post-synaptic potential. The delay determines how long it takes for the signal to travel from the pre-synaptic to the post-synaptic neuron.

One underlying mechanism for learning and memory in biological systems is the regulation of synaptic connection strength, a process referred to as synaptic plasticity. If the synaptic connection is intended to be plastic (not a fixed weight), the definition of the synapse has to include the mechanism of weight dynamics. The facilitation or depression of a synapse between neurons can be handled with weight dynamics in SNNs.

Like traditional Neural Networks, learning in SNNs can be achieved by three strategies: supervised, unsupervised and reinforcement. Supervised learning and Reinforcement Learning are the most commonly used learning approaches in SNNs (Maass, 1996; Schultz et al., 1997; Schultz, 1998; Froemke et al., 2005; Ponulak, 2008; Ponulak & Kasinski, 2010; Gerstner et al., 2014).

This chapter describes some fundamental synaptic plasticity approaches inspired from biological observations that can be designed for SNNs. Those paradigms also inspire the learning techniques for spiking networks proposed in our experiments. Described techniques such as Spike-timing Dependent Plasticity and homeostatic plasticity are fundamental parts throughout further descriptions in chapter 8 and chapter 9.

Firstly, the structure of synapse is detailed in section 3.2 with its brief biological background and its basic artificial interpretation. Then, Hebbian plasticity determines the facilitation or depression mechanisms of a synapse, as discussed in section 3.3. As an extension to the original Hebbian hypothesis, experiments demonstrate that spike timing between pre-synaptic

firing and post-synaptic firing times plays a fundamental role in plasticity mechanism. This timing relationship is called Spike-timing Dependent Plasticity (STDP) which reinforces the synaptic strength (weight) when the spike arrives before the firing of the neuron and decreases the strength (weight) when the spike arrives after the neuron has fired. STDP is discussed in section 3.4 with its model details, implementation aspects, and variants.

In order to maintain the stabilization of neuronal functions in the network, homeostatic plasticity is observed in neocortical circuits (Turrigiano, 2012), where neurons can regulate their own excitability relative to overall network activity. Synaptic scaling is one of potential mechanisms as homeostatic plasticity and it is detailed in section 3.5. Also, modulation of dendritic excitability is a key aspect of synaptic integration (Sjostrom et al., 2008). Hence, biological background of Dopamine modulated plasticity is described in section 3.6. This can be used as biological inspiration basis for the following chapter 4. In this chapter, we describe various plasticities which are basis for the following approaches in section 8.3, section 8.9 and chapter 9.

3.2 Synapse

In the nervous system, the small gap at the junction between an axon and a neuron is called a synapse which allows information to flow from one neuron to another. Those gaps of the neuron axon to next nerve cell are generally chemical contacts rather than electrical contacts. Those junctions enable neurons to exchange signals by diffusion of neurotransmitters. According to the type of synapses, electrical potential can be increased for excitatory synapse or decreased for inhibitory. The physical and chemical characteristics of synapses determine the strength of the incoming signal. Synapses are believed to be the neurobiological basis of learning and memory in the mind (Squire & Kandel, 1999).

Neurons interact via synapses which transmit spiking activities from the pre-synaptic neuron to the post-synaptic one. Synaptic connections are made onto the dendrites and cell bodies of other neurons. Depending on the site of transfer between neurons, the first or previous neuron is called the pre-synaptic neuron and the following neuron receiving the information is called the post-synaptic neuron. Basic schematic structure is shown in Figure 3.2.

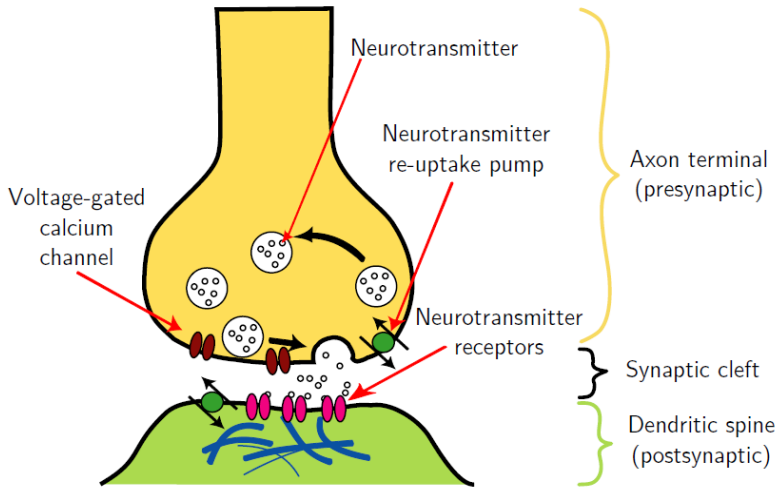


Figure 3.1: Illustration of the main parts of a synapse, taken from Bekolay (2011).

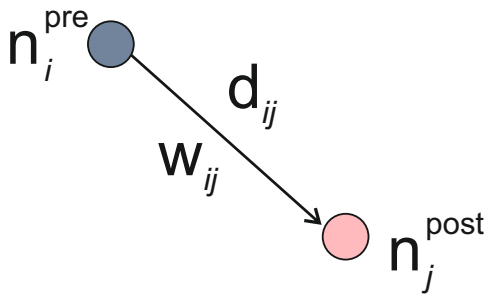


Figure 3.2: Schematic representation of a synapse from the source (pre-synaptic) neuron i, n_i, to the target (post-synaptic) neuron j, n_j. i, j are the pre-synaptic and post-synaptic neuron index, respectively. There is a single synapse s_{ij} with synaptic strength (weight) w_{ij} and axonal delay d_{ij}.

3.2.1 Synaptic Transmission

At most synapses, the information is transmitted in the form of small chemical messengers called neurotransmitters that bind to receptors on other neurons, or cells. Those molecules are released by a pre-synaptic neuron into the synaptic cleft and cause ion channels in target neuron to activate, resulting in a change in the post-synaptic potential. The structure of a synapse is illustrated in Figure 3.1. One of the most common examples of neurotransmitters is acetylcholine (ACh) which is one of the primary neurotransmitters of the peripheral nervous system (Nicholls et al., 1992; Kandel et al., 2000).

Excitatory Post-synaptic Potential (EPSP) is a temporary change in the membrane potential of a neuron. Excitatory synapses increase the electric polarization of the membrane, which is called Excitatory Post-synaptic Potential. On the other hand, Inhibitory Post-synaptic

Potential (IPSP) changes the charge negatively across the nerve membrane resulting in a lowering of the membrane potential to be further from the firing threshold.

3.2.2 Synapse Model

The state of synapses in common synaptic models is represented by a weight variable, w, which is the amount by which the post-synaptic membrane potential is increased. When the membrane potential of the pre-synaptic neuron exceeds the threshold, a spike is emitted. This spike is sent to all post-synaptic neurons forming synapses with it and it causes change in the input current $I_j(t)$ of post-synaptic neuron for a short period of time depending on the weight, w_{ij}. The input current due to synaptic activation is commonly a weighted sum of pre-synaptic currents as:

$$I_j(t) = \sum_{i=1}^{N_{\text{synapses}}} w_{ij}(t) \cdot f(t - t_i^{\text{f}}) \tag{3.1}$$

where f is synaptic current function which can be modelled as an exponential decay, Dirac Delta function (see section A.1), or Alpha function (see section A.2), w_{ij} is the synaptic strength illustrated in Figure 3.2. Throughout the experiments for a synaptic current, we use the Dirac Delta function $\delta(t)$ where it is 1 if $t = 0$, otherwise it is 0. The model can be visualized in Figure 2.5. Synaptic strengths w_{ij} are allowed to contain a mix of both positive and negative weight values. Synaptic connections with negative weights behave like inhibitory synapses.

3.2.3 Synaptic Plasticity

One of the most fundamental and fascinating properties of the mammalian brain is its plasticity; the capacity of the neural activity generated by an experience to modify neural circuit function (Citri & Malenkas, 2008). Synaptic plasticity involves processes on synapses to transfer the information between neurons or between a neuron and a muscle cell. Synapses are able to change their strength and activity because of their feature of extreme plasticity which is widely believed to underlay learning. Based on the amount of released neurotransmitters across the synapse, the efficacy of synapses can be increased (more sensitive) or decreased (less sensitive) (Nicholls et al., 1992; Squire & Kandel, 1999; Kandel et al., 2000).

There are a number of different forms of synaptic plasticity which are observed and described

such as Short-Term Synaptic Plasticity or Long-Term Synaptic Plasticity. Short-Term Synaptic Plasticity (Zucker & Regehr, 2002) is observed that lasts of the order of milliseconds to several minutes. This form of plasticity might a play role in short-term adaptations to sensory inputs, transient changes in behavioural states, and short-lasting forms of memory. On the other hand, Long-Term Synaptic Plasticity has two types called Long Term Potentiation (LTP) and Long Term Depression (LTD). Those synaptic mechanisms are the most prominent forms in the mammalian brain. In our experiments, we use Long-Term Synaptic Plasticity with LTP/LTD based on Hebbian learning models.

3.3 Hebbian Plasticity

In 1949, a psychologist named Donald O. Hebb proposes a mechanism of synaptic plasticity that has a fundamental influence on psychology and neuroscience (Hebb, 1949; Brown & Milner, 2003). The main idea of the theory is that correlated activation in the pre-synaptic and the post-synaptic neuron causes the strengthening of the synaptic efficacy between those two neurons. The original definition is stated by Hebb in his book (Hebb, 1949) as:

"Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability... When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." (Hebb, 1949).

From the point of Artificial Neural Networks view, Hebbian plasticity determines how the synaptic strengths between artificial neurons change. In other words simultaneous neuron activation of pre-synaptic and post-synaptic cells corresponds to the increase of the weight between them; on the other hand, separate neuron activation corresponds to the decrease of the weight between those neurons. This plasticity rule is known as Hebbian plasticity and the synapses which follow the proposed rule are known as Hebb synapses. Hebbian plasticity can be classified as an unsupervised method based on Machine Learning theory (Hertz et al., 1991; Gerstner et al., 2014).

One of basic mathematical descriptions of Hebbian learning with a single pre-synaptic neuron

n_j connected to a single post-synaptic neuron n_i as shown in Figure 3.2 is usually formulated as:

$$\Delta w_{ij} = \eta x_i x_j \tag{3.2}$$

where w_{ij} is the synaptic efficacy between neuron n_j and neuron n_i, Δw_{ij} is the change in synaptic weight w_{ij}, η is the learning rate with small, fixed, and positive value, i indexes a pre-synaptic neuron n_i, j indexes a post-synaptic neuron n_j, x_i, x_j are the activation in the pre-synaptic neuron n_i and post-synaptic neuron n_j, respectively.

3.3.1 Long Term Potentiation (LTP)

In 1966, one form of long term synaptic plasticity, called hippocampal LTP, in the mammalian brain is discovered in the CA1 region of the hippocampus by Terje Lomo (Lomo, 2003). Figure 3.3 from Citri & Malenkas (2008) summarizes their results. Sample experiments illustrates NMDAR-dependent (The N-methyl-D-aspartate receptor dependent) LTP and LTD at hippocampal CA1 synapses. The axons coming from one part of a rat's hippocampal formation, the entorhinal cortex, are stimulated many times in rapid succession, which leads to an increase in activity in another part of the hippocampal formation, the dentate gyrus. The importance of the phenomenon is that they provide an important key to understanding some of the cellular and molecular mechanisms by which memories are formed (Martin et al., 2000; Pastalkova et al., 2006).

The amplitude of the Excitatory Post-synaptic Potential of the population is correlated with the amount of current which flows into post-synaptic neurons. If EPSP increases, pre-synaptic spikes have more influence on post-synaptic firings by imparting more current into the post-synaptic neuron. To sum up, the long-lasting increase in the strength of these synapses can persist for many days (Bliss & Gardner-Medwin, 1973; Nicoll, 2017). LTP remains to this day one of the most inspirational models for memory and learning (Nicoll, 2017).

3.3.2 Long Term Depression (LTD)

Long Term Depression (LTD) is an activity-dependent decrease which results in the strength of synaptic transmission as the reverse of LTP. LTD, is a long lasting depression process, which

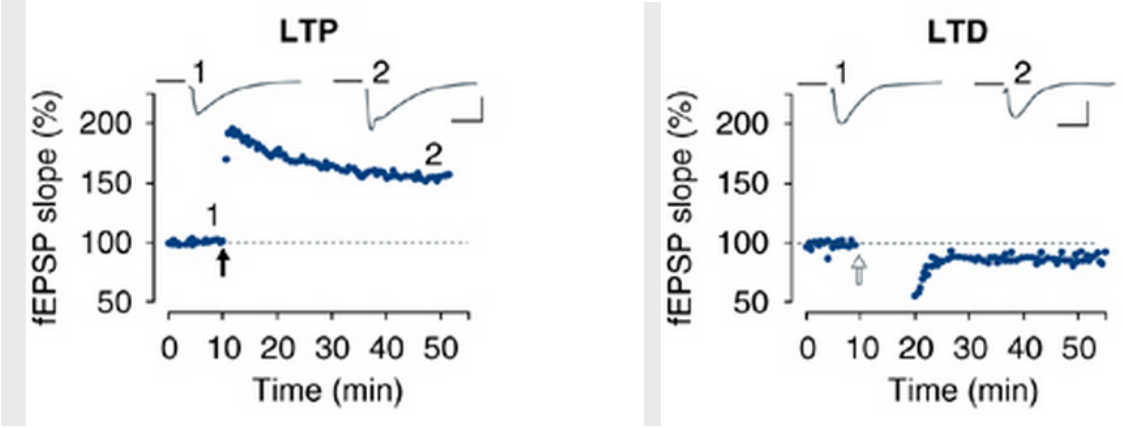


Figure 3.3: Sample experiments illustrating LTP (left) and LTD (right) in the CA1 region of the hippocampus. (Left) Synaptic strength, defined as the initial slope of the field excitatory post-synaptic potential (fEPSP; normalized to baseline) is plotted as a function of time. Data traces are taken at the times indicated by the numbers on the graphs (scale bar: 0.5 mV; 10 ms). (Right) It demonstrates LTD elicited by high-frequency tetanic stimulation (100 Hz stimulation for 1 s; white arrowhead). Right panel illustrates LTD elicited by low-frequency stimulation (5 Hz stimulation for 3 min given twice with a 3 min interval; open arrow). Data traces are taken at the times indicated by the numbers on the graphs (scale bar: 0.5 mV; 10 ms), taken from Citri & Malenkas (2008).

persists over days and months. Because of the correlation of LTD and motor learning, LTD is considered as a critical cellular mechanism for memory and motor learning (Hirano, 2013). Motor learning is a set of processes associated with practice or a novel experience leading to relatively permanent changes in our neuromuscular system functions (Schmidt & Lee, 1999).

The first evidence of LTD is discovered almost a decade after LTP by Lynch et al. (1977). They find that after a tetanic stimulation induced LTP to individual neurons, other afferents to the same neuron have weaker EPSPs, i.e. their synaptic strengths are weakened. This type of LTD does not need pre-synaptic activity to be depressed. Therefore, the process is not predicted by the classic Hebbian mechanism, and it is defined as *anti-Hebbian*, the opposite of Hebbian learning (Massey & Bashir, 2007).

3.4 Spike-timing Dependent Plasticity (STDP)

Spike-timing Dependent Plasticity (STDP) is a further development of Hebbian plasticity which depends on the relative firing times of the pre-synaptic and post-synaptic neurons instead of rate-based Hebb mechanisms (Bi & Poo, 1998, 2001; Froemke et al., 2005; Morrison et al., 2008). STDP learning window or STDP function is a function of the synaptic weights

which is shaped through the relative timing of pre-synaptic spike arrival to post-synaptic action potentials. STDP mechanism has mainly three components: pre-synaptic timing, post-synaptic timing and the efficiency of learning for a time window (Bi & Poo, 1998). The time difference between post-synaptic t_j^{post} and pre-synaptic firing times t_i^{pre} is denoted as $\Delta t = t_j^{post} - t_i^{pre}$. This is the determining factor in how large the synaptic weight needs to be changed.

Hebbian learning strengthens synaptic weights once pre-synaptic neuron fires before post-synaptic neuron (see section 3.3). Since STDP is consistent with the postulate of Hebb, it is also referred to as Hebbian-STDP (Roberts & Bell, 2002; El-Laithy & Bogdan, 2011; Florian, 2007). Tenporal kernels of Hebbian-STDP are shown on the right in Figure 3.4 and Figure 3.5. Anti-Hebbian learning is the opposite of Hebbian learning (see section 3.3). It strengthens the synapses if the post-synaptic neuron fires before the pre-synaptic neuron and weakens the association between neurons once pre-synaptic neuron fires before post-synaptic neuron. On the other hand, there are two main shapes of the STDP form as symmetric in Figure 3.4 and asymmetric in Figure 3.5. Tenporal kernels of anti-Hebbian-STDP are shown on the left in Figure 3.4 and Figure 3.5.

Symmetric Spike-timing Dependent Plasticity (Symmetric-STDP) is only dependent on time difference between pre-synaptic and post-synaptic firing times (Roberts & Bell, 2002). Hence, the sign of synaptic changes does not vary with the sign of the relative spike timing illustrated in Figure 3.4. On the other hand, Asymmetric Spike-timing Dependent Plasticity (Asymmetric-STDP) is also dependent on the temporal order of pre- and post- firing times addition to relative timings with the exact opposite effect of STDP (Roberts & Bell, 2002; Gerstner & Kistler, 2002). This process weakens distal synapses, and strengthens proximal synapses. In other words, if pre-synaptic neuron fires before its post-synaptic neuron, those synaptic weights increase and other synaptic weights decrease if post-synaptic neuron fires before its pre-synaptic neuron. Asymmetric-STDP includes a decay term which is not part of the original Hebbian formulation (Roberts & Bell, 2002).

By Spike-timing Dependent Plasticity mechanism, a network can classify data through self-organisation after training (Roberts & Bell, 2002; Bohte et al., 2002b). In Figure 3.6, the STDP function illustrates the change of synaptic weight connections as a function of pre-

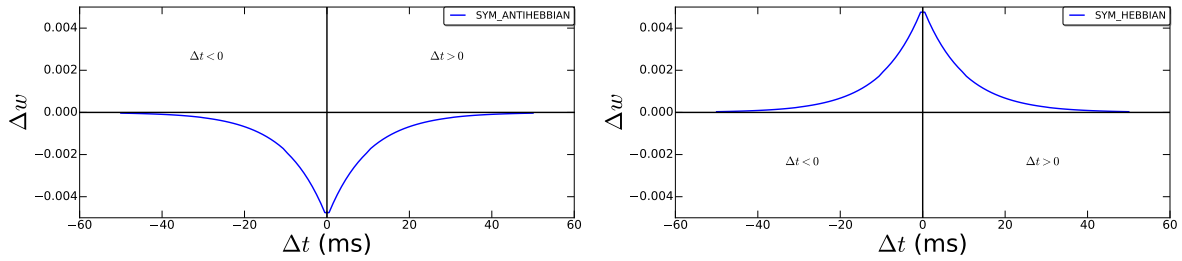


Figure 3.4: Symmetric-STDP. The left hand plot shows symmetric anti-Hebbian-STDP window. The right hand plot shows symmetric Hebbian-STDP window. Δt is the time difference between the post-synaptic firing times t_j^{post} and pre-synaptic firing times t_i^{pre}, $(\Delta t = t_j^{post} - t_i^{pre})$.

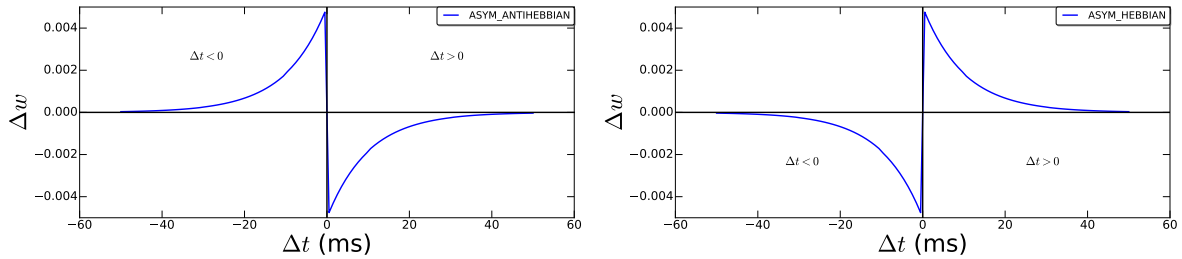


Figure 3.5: Asymmetric-STDP. Regardless of the order of pre- and post-synaptic firing times, STDP function $W(\Delta t)$ modify the weights in the same manner for $\Delta t < 0$ and $\Delta t > 0$. The left hand plot shows asymmetric anti-Hebbian-STDP window. The right hand plot shows asymmetric Hebbian-STDP window. Δt is the time difference between the post-synaptic firing times t_j^{post} and pre-synaptic firing times t_i^{pre}, $(\Delta t = t_j^{post} - t_i^{pre})$.

synaptic and post-synaptic spike timings. STDP exhibits the component of LTD and LTP as it is explained previously. STDP response based on the timing window determines the synaptic modification.

An example of unsupervised learning in SNNs is proposed by (Masquelier et al., 2008) where STDP is applied to detect a single input spatio-temporal spike pattern. From simulations, it is demonstrated that the post-synaptic neuron can learn to respond to the spike pattern with a precisely timed output spike.

3.4.1 Modeling of STDP

The relative timing between pre-synaptic arrivals and post-synaptic firings determines the weight change between neurons within a critical timing window. The critical timing window for potentiation and depression is typically 20-25 ms according to experimental studies (Bi

& Poo, 1998; Song et al., 2000). Outside this window, potentiation or depression does not occur. Experimental observations show that, if a post-synaptic firing follows a pre-synaptic firing activity within the timing window, potentiation takes place (Figure 3.6). Also, if a pre-synaptic firing follows a post-synaptic firing activity within the timing window, depression occurs (Figure 3.6). The weight change Δw_{ij} of a synapse from a pre-synaptic neuron j to post-synaptic neuron i is illustrated in Figure 3.6. The total weight change Δw_{ij} is proposed by Gerstner et al. (1996) as it is formulated in Equation 3.4.

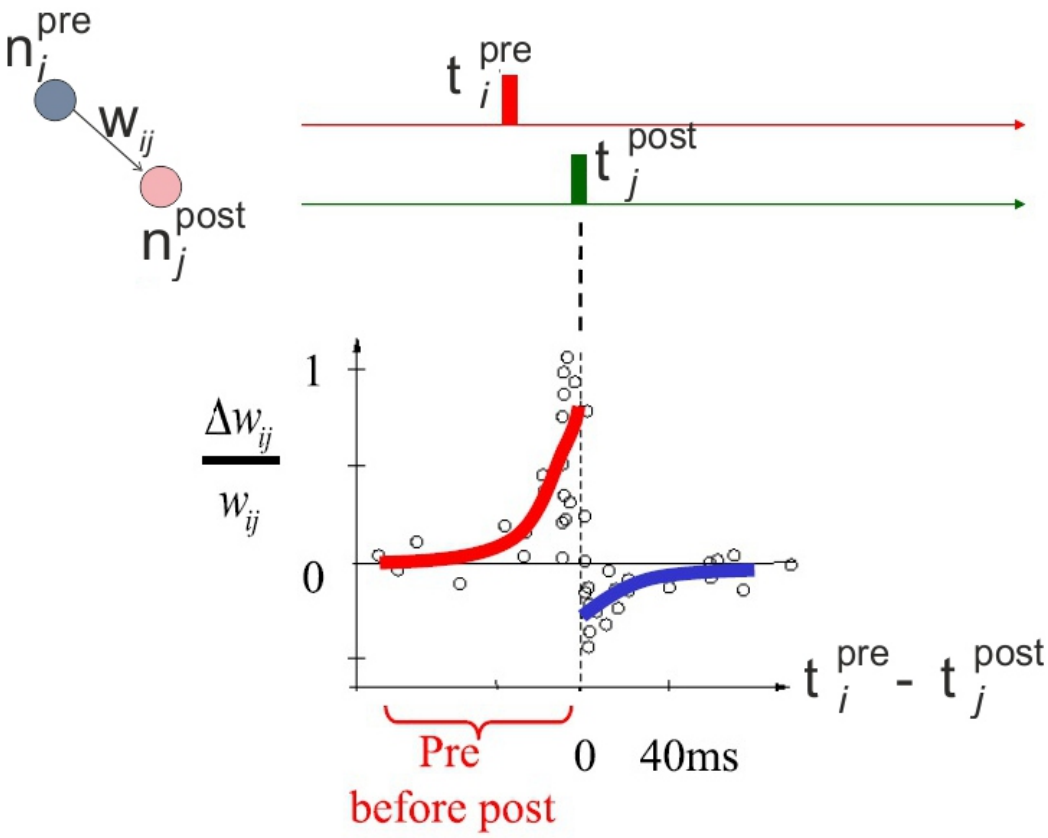


Figure 3.6: The schematic of STDP window function illustrates the change of synaptic strengths as the vertical scale (dimensionless) based on temporal relationship between pre- and post-synaptic spikes (Sjostrom & Gerstner, 2012). Horizontal scale shows the time difference $\Delta t = t_i^{pre} - t_j^{post}$. Convention about the time difference Δt in this graph is slightly different than the one used in the thesis. (Note that the notation of $\Delta t = t_j^{post} - t_i^{pre}$ is used throughout the thesis). For instance, if there is no time difference ($\Delta t = t_i^{pre} - t_j^{post} = 0$), the current weight will not be increased or decreased $\Delta w_{ij} = 0$. The plotted STDP measured in biological tissue (rat hippocampal neurons) (Bi & Poo, 1998). This figure includes experimental data taken from Bi & Poo (1998). Pre-synaptic neuron and post-synaptic neuron are shown with i and j, respectively.

The pre-synaptic spike arrival times can be described as t_i^f at synapse i with f as pre-synaptic firing index, $f = 1, 2, 3, ...$ shows the sequence of pre-synaptic spikes as it illustrated in Fig-

ure 3.6. Similarly, the post-synaptic firing times can be similarly named as t_j^n at synapse j with n as post-synaptic firing index, $n = 1, 2, 3, ...$ represents the firing times of the post-synaptic neuron. Based on that, synaptic modification as:

$$\Delta w_{ij} = \sum_{i=1}^{N} \sum_{n=1}^{N} W(t_j^n - t_i^f) \tag{3.3}$$

Any proposed STDP model for weight function in Equation 3.3 describes the amount of synaptic modification from a pre-synaptic to post-synaptic spikes. As a simple Spike-timing Dependent Plasticity model, one of the popular choice for weight of the synapse, $W(\Delta t)$, can be formulated with following set of equations.

$$w(t+1) = w(t) + \Delta w \tag{3.4}$$

where Δw is the change in weight with:

$$\Delta t = t_j^n - t_i^f \tag{3.5}$$

where Δt is the time difference between the post-synaptic firing times t_j^n and pre-synaptic firing times t_i^f. We refer to the conditions $\Delta t > 0$ as positive and $\Delta t < 0$ as negative spike-pair timing.

The standard rule for STDP is used which adjusts each connection strength in response to the time difference between the spikes of pre- and post-synaptic neurons. Integral kernels as τ_{pre} and τ_{post} define the shape of STDP process (Gerstner & Kistler, 2002). The weight update W, is also called the STDP modification function, can be modelled in a common form as follows:

$$W(\Delta t) = \begin{cases} +F_{pre}(w)e^{-\frac{\Delta t}{\tau_{pre}}}, & \text{if} \quad \Delta t \geq 0, \\ -F_{post}(w)e^{+\frac{\Delta t}{\tau_{post}}}, & \text{if} \quad \Delta t < 0. \end{cases} \tag{3.6}$$

where $\Delta t = t_j^n - t_i^f$ is the difference between the posts- and pre-synaptic neuron firing times. $F_{pre}(w)$ and $F_{post}(w)$ describe the dependence of the update on the current weight of the synapse as detailed in subsection 3.4.2. Typically a positive weight update ($W(\Delta t) > 0$) for a pre- before post-synaptic firing activity ($\Delta t > 0$), and a negative weight update ($W(\Delta t) < 0$) for a pre- after post-synaptic spike ($\Delta t < 0$) are applied through this learning. τ_{pre} and τ_{post}

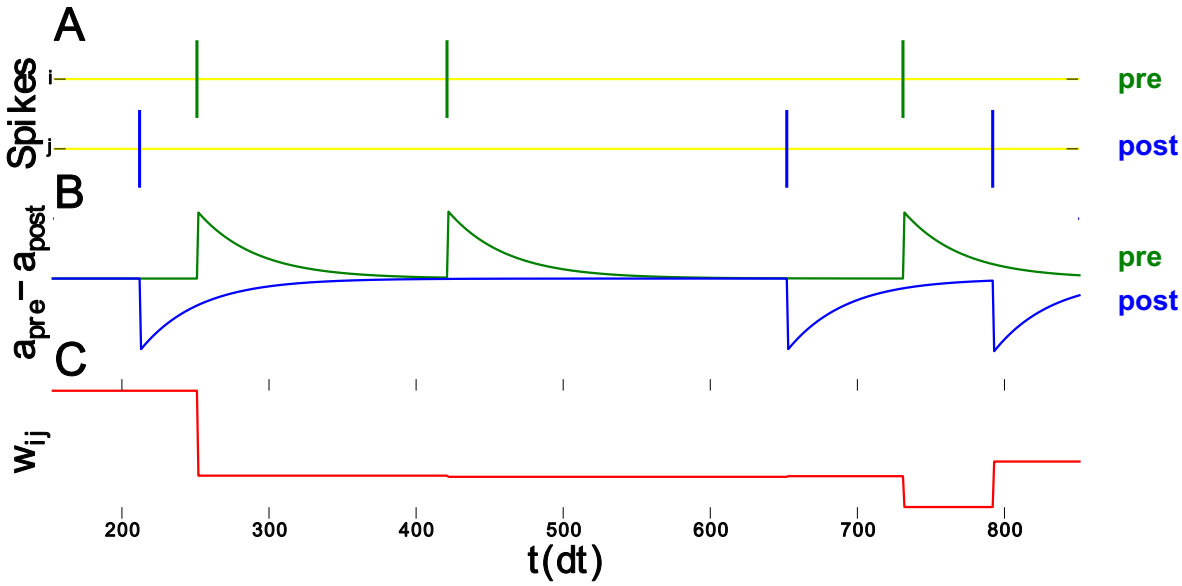


Figure 3.7: Illustration of the implementation of the STDP process over simulation time steps, $dt = 1$ms. There is a pre-synaptic neuron n_i and post-synaptic neuron n_j with a synaptic weight w_{ij} and an axonal delay $d_{ij} = 0$. **A)** The pre-synaptic firing times as the top row at t_i^f, green, and the post-synaptic spike train as the bottom row at t_j^n, blue. **B)** LTP in green and LTD in blue subwindows for excitatory connections can bee seen with a_{pre} and a_{post} based on pre-synaptic and post-synaptic spikes from (A). A_{pre} and A_{post} in Table 3.1 are maximum amplitude of a_{pre} and a_{post} traces, respectively. The synaptic change ΔW (y-axis) determines the amount of the weight modifications. **C)** Shorter time difference Δt cause larger weight update Δw within τ_{pre} and τ_{post} which are the decay constants of the learning window (see Table 3.1). First weight modification is a decrease because of the post-synaptic spike generation shortly before the pre-synaptic spike. The actual time difference between pre-post spiking times is $\sim -4ms$. Next modification is slight weight decrease because of longer time difference $\sim -8ms$ between pre-post spiking times. Increasing weight changes is induced by the pre-synaptic spike followed by the post-synaptic spike nearly at the end of the presentation. For instance, the relation of 2^{nd} pre-synaptic spike at $\sim 42ms$ and 2^{nd} post-synaptic spike at $\sim 65ms$ does not cause any increase or decrease in the synaptic weight because the current time difference between pre- and post-synaptic spikes is out of the time windows of STDP, τ_{pre} and τ_{post}.

> 0 are the decay constants of the STDP learning window for LTP and LTD (Song et al., 2000).

The updating functions $F_{pre}(w)$ and $F_{post}(w) > 0$ control the amplitude of the learning window once pre-synaptic and post-synaptic events are happened, respectively. In other words, $F_{pre}(w)$ and $F_{post}(w)$ determine the scale of contribution to synaptic plasticity: higher values indicate larger contributions. For the current plasticity modelling, all parameters are summarized in Table 3.1.

Although, various shapes of STDP windows are proposed in the literature (Song et al., 2000;

Parameter Type	Parameter Names	Values
Plasticity amplitudes	A_{pre}, A_{post}	0.005, 0.005
Decay constants	τ_{pre}, τ_{post}	5ms, 5ms
Weight dependence factor	μ	0

Table 3.1: Model parameters used for the computer simulations through STDP.

Senn et al., 2001; Izhikevich et al., 2004; Kistler, 2002), in this work we use one of the most common learning windows for the STDP process as Asymmetric-STDP illustrated in Figure 3.7. This figures shows the simulation of post-pre spiking events, and confirms that the operation of the STDP rule is consistent with the theory presented. The STDP shape in the figure is obtained by using parameters summarized in Table 3.1. A_{pre} and A_{post} as a form of $F_{pre}(w)$ and $F_{post}(w)$ are positive constants. Although it is common that $A_{pre} \leq A_{post}$ for network stability, we set $A_{pre} = A_{post} = 0.005$ here because this selection is the basis for further learning techniques described in section 8.3 and section 8.9. A_{pre} and A_{post} denote the maximum change on synaptic efficacies. Decay times τ_{pre} and τ_{post} are determined as the half of the maximum Inter-spike Interval in order to capture entire trace activity. We set τ_{pre} $= \tau_{post} = 5$ ms here summarized in Table 3.1.

3.4.2 Weight Dependence

Experimental evidence for the weight dependence of STDP is already discussed (Bi & Poo, 1998). Although the way to adjust the weight dependence of the synaptic changes differ, a common form of weight dependence through updating functions of F_{pre} and F_{post} can be expressed as:

$$\begin{aligned} F_{pre}(w_{ij}) &= \eta_S(1 - w_{ij})^{\mu} \\ F_{post}(w_{ij}) &= \eta_S \varphi w_{ij}^{\mu} \end{aligned} \tag{3.7}$$

where η_S is the learning rate for STDP, $\varphi > 0$ is an asymmetry factor, μ is a non-negative exponent in order to control the dependence of weight changes on the current value of w_{ij} (Gutig et al., 2003).

The selection of parameters in Equation 3.7 allows various forms of dependency. Firstly, Additive-STDP (Kempter et al., 1999; Song et al., 2000) is weight independent ($\Delta w/w \approx$

constant). This rule can be driven from Equation 3.7 by choosing $\mu = 0$ as:

$$\begin{aligned} F_{pre}(w_{ij}) &= \eta_S \quad &= A_{pre} \\ F_{post}(w_{ij}) &= \eta_S \varphi \quad &= A_{post} \end{aligned} \tag{3.8}$$

with $A_{pre}\tau_{pre} \leq A_{post}\tau_{post}$. If $A_{pre}\tau_{pre} < A_{post}\tau_{post}$, depression overpowers potentiation. However, in the current experiment we use same values as $A_{pre}\tau_{pre} = A_{post}\tau_{post}$ shown in Table 3.1 because of further considerations in following learning techniques in chapter 8.

Secondly, Multiplicative-STDP (Gutig et al., 2003) has a linear weight dependence ($\Delta w \propto w$) for depression and constant potentiation (van Rossum et al., 2000). This rule also can be driven from Equation 3.7 by choosing $\mu = 1$ as:

$$\begin{aligned} F_{pre}(w_{ij}) &= \eta(1 - w_{ij}) \\ F_{post}(w_{ij}) &= \eta \varphi w_{ij} \end{aligned} \tag{3.9}$$

There are some forms such as Power-STDP by choosing intermediate values for μ between (0,1) (Rubin et al., 2001) and Log-STDP proposed by Gilson & Fukai (2011). In our simulations throughout the project, $\mu = 0$ using an additive form for the STDP weight update. After this point, STDP refers to Additive-STDP unless otherwise explicitly expressed.

3.4.3 Implementation of STDP

If we rewrite Equation 3.6 based on Additive-STDP ($\eta_S \ll 1$), we get:

$$W(\Delta t) = \begin{cases} +A_{pre} e^{-\frac{\Delta t}{\tau_{pre}}}, & \text{if} \quad \Delta t \geq 0, \\ -A_{post} e^{+\frac{\Delta t}{\tau_{post}}}, & \text{if} \quad \Delta t < 0. \end{cases} \tag{3.10}$$

where $\varphi = A_{post}/A_{pre}$.

Simulating STDP directly using Equation 3.10 would be inefficient because all pairs of spikes have to be summed over. This is also physiologically implausible because the neuron can not remember all its individual firing times. A more plausible and efficient approach is handled

by defining traces for both pre- and post-synaptic activity.

$$\begin{aligned} \tau_{pre}\frac{\mathrm{d}}{\mathrm{d}t}a_{pre} &= -a_{pre} \\ \tau_{post}\frac{\mathrm{d}}{\mathrm{d}t}a_{post} &= -a_{post} \end{aligned} \tag{3.11}$$

where a_{pre} and a_{post} are traces of pre- and post-synaptic activity, respectively. τ_{pre} and τ_{post} are the decay constants of the learning window.

Once a **pre-synaptic** spike is generated, the pre-synaptic trace is modified also the weight is modified based on the rule:

$$\begin{aligned} a_{pre} &\rightarrow a_{pre} + A_{pre} \\ w &\rightarrow w + a_{post} \end{aligned} \tag{3.12}$$

where $A_{pre} > 0$ is the starting amplitude of decay once a pre-synaptic event happens. Each pre-synaptic spike arrival leaves a trace a_{pre} which is updated by A_{pre} at the moment of spike arrival and decays exponentially if there is not further pre-synaptic spiking activity. The synaptic weight is updated by an amount of the post-synaptic trace a_{post} that depends on the time $\Delta t = t_j^{post} - t_i^{pre}$.

Once a **post-synaptic** spike is generated:

$$\begin{aligned} a_{post} &\rightarrow a_{post} + A_{post} \\ w &\rightarrow w + a_{pre} \end{aligned} \tag{3.13}$$

where $A_{post} > 0$ is the starting amplitude of decay once a post-synaptic event happens. Each post-synaptic spike arrival leaves a trace a_{post} which is updated by A_{post} at the moment of spike arrival and decays exponentially if there is no further post-synaptic spiking activity. The synaptic weight is updated by an amount of the pre-synaptic trace a_{pre} that depends on the time $\Delta t = t_j^{post} - t_i^{pre}$.

3.4.4 Weight Bounds

For biological reasons, keeping the synaptic weights in a pre-defined range is desirable. This is expressed as:

$$w_{min} < w_{ij} < w_{max} \tag{3.14}$$

where all synaptic strengths w_{ij} have essentially values between a minimum weight w_{min} or a maximal weight w_{max}.

In our experiments, we apply minimum and maximum values for the synaptic weights as:

$$\begin{aligned} if \quad & w_{ij} > w_{max}, && w_{ij} \rightarrow w_{max} \\ if \quad & w_{ij} < w_{min}, && w_{ij} \rightarrow w_{min} \end{aligned} \tag{3.15}$$

So all weights are clipped to its minimum w_{min} or maximum w_{max} values, respectively.

3.5 Homeostatic Plasticity

Hebbian plasticity alone is insufficient to explain activity-dependent development because it tends to destabilize the activity of neural circuits (Zenke & Gerstner, 2017). In order to maintain stable activity in neural circuits, additional compensatory processes are required. In the literature (Turrigiano, 2008; Watt & Desai, 2010; Turrigiano, 2012; Zenke & Gerstner, 2017), a diversity of homeostatic plasticity phenomenon observed in neural circuits is considered it involves to operate this role. Hence, average neuronal activity levels are maintained by a set of homeostatic plasticity mechanisms that dynamically refine synaptic strengths to promote stability.

Homeostatic mechanisms stabilize neuronal activity during learning and development (Turrigiano, 2012). The network must maintain its stable functionality in the face of synaptic strength changes during neuronal development. Homeostatic plasticity mechanisms are typically interpreted as negative feedback processes, which rely on an error signal to maintain neuronal activities around functional operating points (Turrigiano, 2012; Zenke & Gerstner, 2017). For instance, this set point is generally defined as the average post-synaptic firing rate (Turrigiano, 2008; Watt & Desai, 2010; Turrigiano, 2012; Zenke & Gerstner, 2017).

3.5.1 Synaptic Scaling

Synaptic scaling is proposed as one of potential mechanisms for the stabilization of neuronal function. Neurons detect changes in their own firing rates and then regulate their own excitability. This mechanism is synaptic activity-dependent scaling (Turrigiano, 1999) which allows neurons to regulate their overall firing rate without touching the stability of the trained weight distribution. So neural activities through a network can perform without either dying out or increasing uncontrollably. A simplified version of this synaptic scaling (van Rossum et al., 2000) is applied in our experiments as:

$$\frac{dw_{ij}(t)}{dt} = \beta w_{ij}(t)[N_{des} - N_{act}] \tag{3.16}$$

where w_{ij} indicates the synaptic weight from pre-synaptic neuron i to post-synaptic neuron j as shown in Figure 3.2; β is a constant determining the strength of the synaptic scaling globally. Post-synaptic activity as a measure of post synaptic activity in fixed time window from the desired neuron and actual neuron are N_{des} and N_{act}, respectively. This mechanism controls the spiking activity of all neurons by multiplicatively scaling synaptic strengths in the correct direction to maintain neuronal firing rates.

However, this up or down scaling cause instability in the trained weights because homoeostatic plasticity is active during our experiments. Therefore, a similar approach using a range of spike numbers is used as a way to maintain the stability inspired from Turrigiano (2008).

$$\frac{dw_{ij}(t)}{dt} = \begin{cases} \beta w_{ij}(t)[N_{des}^{max} - N_{act}], & \text{if} \quad N_{act} > N_{des}^{max}, \\ \beta w_{ij}(t)[N_{des}^{min} - N_{act}], & \text{if} \quad N_{act} < N_{des}^{min}, \end{cases} \tag{3.17}$$

where N_{des}^{min} and N_{des}^{max} are minimum and maximum desired post-synaptic spike numbers, respectively. Instead of keeping the firing rate in a precise rate, the above form keeps the post-synaptic firing rate inside a range $N_{des}^{min} < N_{act} < N_{des}^{max}$ by iteratively scaling weights. If the neural activity is inside the range, the main local plasticity mechanism of STDP is responsible for the learning. Otherwise, the scaling mechanism adjusts all of a neuron's synapses globally up or down to keep the activity in the specified range. This range can be

determined as:

$$N_{des}^{max} = \lfloor N_{des} * (1+p) \rceil$$
$$N_{des}^{min} = \lfloor N_{des} * (1-p) \rceil \tag{3.18}$$

where p is the scaling range factor with $0 \leq p \leq 1$, $\lfloor N_{des} * (1+p) \rceil$ is the nearest integer function which returns the nearest integer to $N_{des} * (1+p)$. This factor p can be chosen as a reasonable amount such as %20 (p=0.2). However, this set point can be different as well. Lower values decrease the range width and increase the sensitivity of scaling, and vice versa. Considering with one of proposed experiments in section 8.3, if the output firing rate r_{des} = 0.03 Hz, the scaling range rate $p = 0.20$, the windowing length $T = 100$ ms, $N_{des} = r_{des} * T$ = 3 spikes. $N_{des}^{min} = 2$, $N_{des}^{max} = 4$.

In the proposed mechanism, other parameters N_{des}^{min} and N_{des}^{max} are calculated based on scaling range factor p. If p is relatively low, the mechanism of synaptic scaling is more active throughout training runs. On the other hand, if p is relatively high, synaptic scaling is rarely active. For instance, if p is high as $p = 1$, $N_{des}^{max} = 6$ and $N_{des}^{min} = 0$ in this example. If this is selected, as long as the number of actual spike output is between 0 and 6, there is no scaling during learning. The value of scaling range factor p is determined experimentally.

Activity-dependent scaling of synaptic weights preserves stability of the network by preventing under- and over-activation levels from all synapses throughout training runs. This homeostatic plasticity is controlled by an error signal that is the difference between the actual spike numbers and the spike number of the nearest lower or upper bound. It tunes all synaptic strengths in correct direction if it is necessary.

3.6 Dopamine Modulated Plasticity

Dopamine (DA) and functions of DA are first identified in the central nervous system (CNS) in the late 1950s (Beninger, 1983). Dopamine is a neuromodulator in the nervous system that regulates many functions in the CNS. The neurons of DA system are closely associated with various motor and cognitive functions, particularly Reinforcement Learning (or reward-driven learning), movement selection, working memory and attention (Fellous & Suri, 2003). Through the burst stimulation of the DA neurons, the DA concentration at the DA modulated synapses

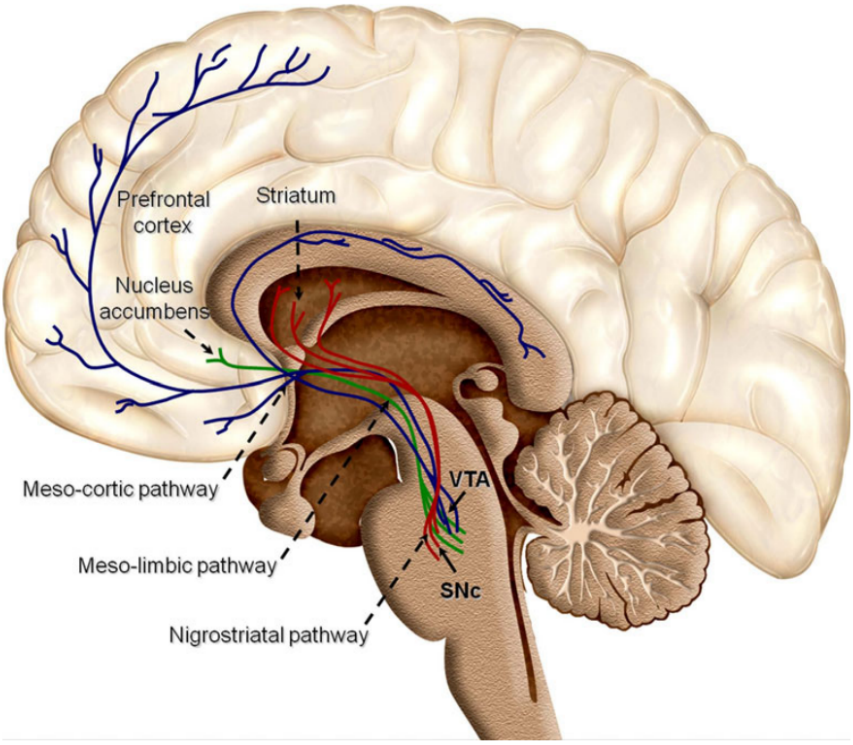


Figure 3.8: Overview of reward structures in the human brain : Dopaminergic neurons are located in the midbrain structures substantia nigra (SNc) and the ventral tegmental area (VTA). Their axons project to the striatum (caudate nucleus, putamen and ventral striatum including nucleus accumbens), the dorsal and ventral prefrontal cortex. Additional brain structures influenced by reward include the supplementary motor area in the frontal lobe, the rhinal cortex in the temporal lobe, the pallidum and subthalamic nucleus in the basal ganglia, and a few others. Figure is taken from Arias-Carrion et al. (2010).

is affected, which adjusts the long-term potentiation or depression of the synapses (Otani et al., 2003). Although dopaminergic neurons can be localized in the mesencephalon, diencephalon and the olfactory bulb (Arias-Carrion & Poppel, 2007), nearly all DA cells reside in the ventral part of the mesencephalon as illustrated in Figure 3.8.

Based on behavioural learning paradigms, animals not only change their behaviour according to the received rewards like juice or food, but also avoid aversive stimuli like foot shocks. Although the psychological phenomenology of behavioural learning is well developed (Rescorla & Wagner, 1972) and many algorithmic approaches to reward learning such as Reinforcement Learning are available (Sutton & Barto, 1998), the relation of behavioural learning to synaptic plasticity is not fully understood (Fremaux et al., 2010). On the other hand, DA appears to be important for learning and memory processes (Schultz, 2007) and it is studied that the influential interpretation of subcortical Dopamine signals is reward-prediction error like Temporal-Difference learning (Sutton & Barto, 1998) in Reinforcement Learning (Schultz,

2007).

3.7 Summary

The mechanisms of synaptic inputs and synaptic transmission have been presented in this chapter. Learning in Spiking Neural Networks is a complex process since the information is encoded in time domain. Hence, various synaptic plasticities in SNNs have been detailed with their brief biological backgrounds. We have initially started with Hebbian plasticity, then the structure of Spike-timing Dependent Plasticity is discussed to capture the essence of unsupervised learning in the nervous system. Although there are various forms of Spike-timing Dependent Plasticity, the learning method based on Additive-STDP is implemented to train the network during experiments. Beyond STDP, homeostatic plasticity as a further process in regulating synapses has been discussed as well. It has an important role to maintain an optimal level of firing activity in the nervous system. The ideas are developed in the chapter 8 and chapter 9.

The next chapter outlines Reinforcement Learning from the viewpoint of Machine Learning. Reinforcement Learning is linked with Dopamine modulation and Spike-timing Dependent Plasticity in chapter 9.

Chapter 4

Reinforcement Learning

Contents

4.1 Introduction to Reinforcement Learning 95
4.1.1 Reinforcement Learning in Spiking Neurons 97
4.2 The Agent-Environment Interface 98
4.3 The Markov Property 100
4.4 An Example Markov Chain 101
4.5 Markov Decision Process 104
4.6 Value Function 105
4.7 Bellman Equations 107
4.8 Optimisation 108
4.9 Algorithms for Reinforcement Learning 109
4.9.1 Temporal-Difference Learning 110
4.9.2 Eligibility Traces 110
4.9.3 Q-Learning 111
4.9.4 Replacement Rules for Maze Navigation 114
4.10 Summary 116

4.1 Introduction to Reinforcement Learning

Reinforcement Learning (RL), a particular branch of Machine Learning, deals with the problems faced by an agent that learns through trial-and-error interactions with a dynamic environment (Kaelbling et al., 1996). In other words, without specifying how the task is to be fulfilled, the agent can be given reward and punishment by Reinforcement Learning techniques.

A number of successful applications based on Reinforcement Learning techniques are applied in various fields such as an elevator scheduling (Crites & Barto, 1998), channel allocation of

cell phone network (Singh & Bertsekas, 1997) or individual chemotherapy treatment decisions (Zhao et al., 2009). Also, the hypothetical maze task running from start to finish is used in the past as a benchmark application for learning algorithms, particularly Reinforcement Learning (Kaelbling et al., 1996; Sutton & Barto, 1998; Matignon et al., 2006; Vasilaki et al., 2009; Venkata et al., 2011; Fremaux et al., 2013; Nowe & Brys, 2016).

RL algorithms are generally divided into two categories: value-function based methods and policy search methods. Value-function-based methods approximate value function iteratively by moving through the world (Sutton & Barto, 1998). Whereas, policy search algorithms run an explicit policy, and perform actions taken from that policy (Sutton et al., 1999). Policy gradient methods, a subset of policy search methods, update their policy parameters approximately proportional to the gradient using differentiable parametrized function approximators. Those methods can select actions without consulting a value function which might still be used to learn the policy parameter (Sutton & Barto, 1998).

Temporal-Difference (TD) learning is one of the common value-function based methods from Reinforcement Learning approaches proposed by Sutton & Barto (1998). This algorithm approximates the expected future cost of a Markov chain problem based on observations. For a large-scale problem, discount factor is taken into account to help converge solutions (Sutton & Barto, 1998). The discount factor γ quantifies the present value of future rewards against to immediate rewards.

Reinforcement Learning problem is faced by an agent that must learn behaviour through trial-and-error search with an environment. Reinforcement Learning has two fundamental features: trial-and-error interactions and delayed reward. A detailed specification of any Reinforcement Learning problem is discussed later as a Markov Decision Process (MDP) in section 4.5. However, it can be mentioned here that the basic description of the reinforcement problem is to achieve a *goal* by *actions* through the *sensation* of environmental states (Dayan & Hinton, 1997; Sutton & Barto, 1998).

In this chapter, the maze task, a standard episodic Reinforcement Learning framework (Sutton & Barto, 1998), is used as an example to illustrate the concept of RL. which is commonly used (Kaelbling et al., 1996; Sutton & Barto, 1998; Matignon et al., 2006; Vasilaki et al., 2009;

Venkata et al., 2011; Fremaux et al., 2013; Nowe & Brys, 2016). Our objective in this chapter is to provide an overview to Reinforcement Learning at a level combined with Spiking Neural Networks techniques to offer a more optimal way for trading off between exploration and exploitation challenges. In this chapter, we introduce the maze task that we try to evaluate our algorithms on during the rest of the thesis.

Section 4.2 presents an overview of the Agent-Environment Interface for Reinforcement Learning problems. Following that, Markov Decision Process is discussed to provide the formal basis underlying Reinforcement Learning methodologies in section 4.3. To illustrate the theoretical features of Markov processes on a real problem, we introduce a maze task scenario with Markov chain properties in section 4.4. In section 4.5, Markov Decision Process Framework is defined for reinforcement problems. Then, the solution of any MDPs for RL is proposed.

4.1.1 Reinforcement Learning in Spiking Neurons

In the physiological realm, memory is thought to be stored in the synaptic connections between neurons (Bliss & Collingridge, 1993; Vasilaki et al., 2009). The strength of synapses can be changed by learning. In the theoretical realm, synaptic weight values are modifiable parameters in order to perform learning. Policy gradient methods optimize their policy through neural parameters like synaptic weights using differentiable parametrized function approximators such as Spiking Neural Networks.

The spiking networks used for biological models are biologically plausible; while non-spiking networks are not intended to be biologically plausible. Also, spiking networks are able to encode temporal information in their signals with the help of more plausible plasticity rules, whereas non-spiking networks encode information with static input (Vasilaki et al., 2009).

Gradient descent methods are also linked to three-factor rules in Spiking Neural Networks through reward optimization tasks (Vasilaki et al., 2009). Associative (Hebbian) plasticity (see section 3.3) indicates association between the timing of pre- and post- neurons' activity called two-factor (term) learning rule (Bi & Poo, 1998). However, recent studies demonstrate that the original associative plasticity can be regulated in various ways such as through neuromodulators (Reynolds et al., 2001; Gu, 2002; Seol et al., 2007; Johansen et al., 2014),

glial factors (Ben Achour & Pascual, 2010) and GABAergic inputs (Paille et al., 2013). This is called three-factor (term) learning rule.

A very general form of three-factor learning rules is written (Vasilaki et al., 2009) as:

$$\Delta w_{ij} = gH(x_j, x_i) \tag{4.1}$$

where w_{ij} is the synaptic efficacy between pre-synaptic neuron n_j and post-synaptic neuron n_i (see Figure 3.2), Δw_{ij} is the change in synaptic weight w_{ij}, g is a third factor modulating the synaptic plasticity, x_j, x_i are the activation in the pre-synaptic neuron n_j and post-synaptic neuron n_i, respectively. H is a generalized Hebbian term (see Equation 3.2), which depends on the correlation between pre- and post-synaptic activities.

Although Reinforcement Learning provides autonomous learning systems, it has also some challenges such as the trade-off between exploration (attempting to discover new things you are not sure about and possibly gathering more information) and exploitation (getting the best results based on what we currently know), and its scalability to high-dimensional continuous state-action systems (Sutton & Barto, 1998). Therefore, in new developments, RL researchers start to combine various algorithms with classical RL approaches in order to deal with more complex learning system (Dayan & Hinton, 1997; Peters & Schaal, 2008). In this thesis, we also combine RL approach with Spike-timing Dependent Plasticity (STDP) described in section 3.4. Hence, this chapter introduces fundamentals of RL for the following algorithms (see chapter 9) with performed experiments.

4.2 The Agent-Environment Interface

The Reinforcement Learning problem is to achieve a goal step by step by trial-and-error interactions with agent's environment. The goal of the agent is to maximize the total reward it receives over time. In this paradigm, the agent can perceive its state and perform actions. Based on each action, a numerical reward is given. Therefore, the RL agent can learn from the consequences of its actions and it can make decisions based on its past experiences. Figure 4.1 shows the agent-environment interaction.

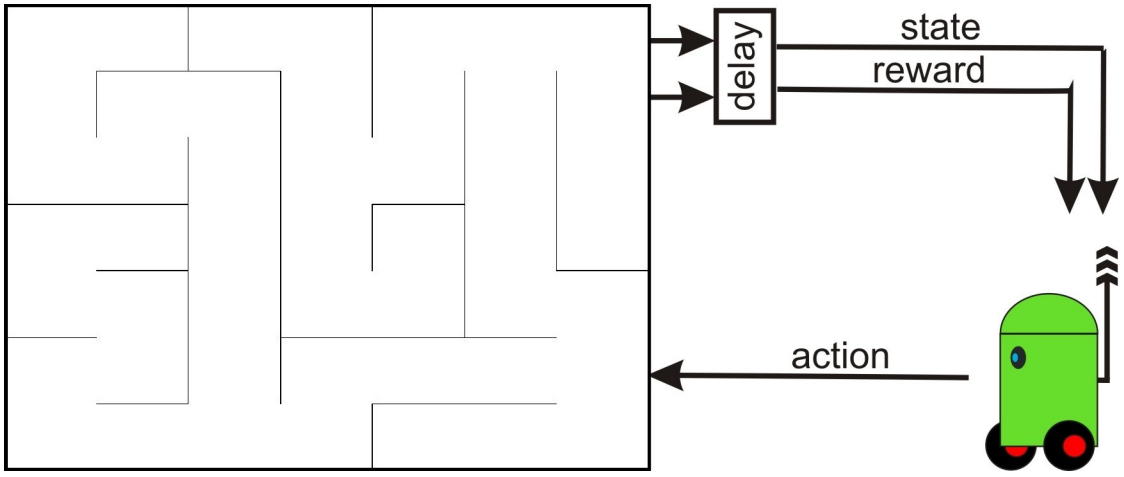


Figure 4.1: The Reinforcement Learning framework. An agent (the robot with wheels on the right bottom) interacts with an environment (maze on the left). This interaction provides feedback about the next state, the validity of a certain action and the reward for a certain action. The agent selects another action in the next state in order to learn more about the environment by trial-and-error through maximizing the received reward in the long run.

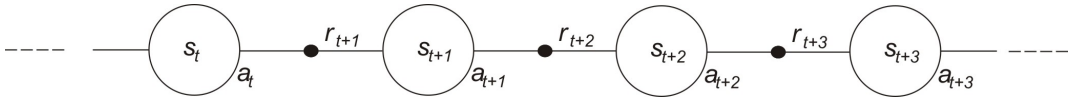


Figure 4.2: The Reinforcement Learning state transition.

In the Reinforcement Learning problem, after the interaction between the agent and the environment at each discrete time step: *t= 0, 1, 2, 3, ...*, the agent receives the environment's state information, $s_t \in S$, where S is the set of possible states. Formally, this environment for Reinforcement Learning can be described as Markov Decision Process (MDP) which is explained in later sections. Based on possible states, the agent performs an action, $a_t \in A(s_t)$, where $A(s_t)$ is the set of actions for s_t. In the next step, as a consequence of its previous action, the agent receives a numerical reward, $r_{t+1} \in R$, and it finds a next state which is s_{t+1} as can be seen in Figure 4.2.

Here, we should mention that traditional Reinforcement Learning is also linked to a new paradigm of simulation-based tree search, Monte Carlo tree search (MCTS) (Coulom, 2006), because of their search and planning framework (Silver, 2009). Similar to RL, MCTS uses the gathered information by focusing on improvements of policies. Both techniques of RL and MCTS optimally need to determine actions and balance the exploration and exploitation phases. However, RL techniques often guide the selection policy by updating the gained knowledge through state transitions in Figure 4.2 (Silver, 2009; Vodopivec et al., 2017). Unlike RL, MCTS techniques usually focus on creating strong selection policy (Coulom, 2006;

Vodopivec et al., 2017).

4.3 The Markov Property

A Discrete Time Markov Chain (DTMC) is a set of possible environment states that fulfil the "*Markov Property*" which means that future states are independent of the past given the present state (Norris, 1997). The possible states of the chain are $s_1, s_2, s_3, s_4, ..., s_t$ throughout the state space S. The chain is finite-state if the set S is finite ($S = \{0, 1, ..., T\}$).

The present state captures all the relevant information which is included in the value of s_t as the state space of the chain. In other words, any information about the process's behaviour before time t is irrelevant in terms of the influence on the future of the process. Therefore, the term is also known as the "*memoryless property*" of a stochastic process.

Assuming that s_t is the state of the Markov chain after the t^{th} transition, $\{s_t : t \geq 0\}$ is the stochastic process (Markov chain). At time epochs *t= 0, 1, 2, 3, ...* the process changes from one state i to another state j with the transition probability $p_{i,j}$:

$$p_{i,j} = \mathbb{P}(s_{t+1} = j | s_t = i) \tag{4.2}$$

where $p_{i,j}$ is the entry in the i^{th} row and j^{th} column in a transition matrix that is defined below.

The Markov property asserts that the conditional probability of future states, given the present states and information about past states, depends only on the present state. Hence, the Markov property can be formulated as:

$$\mathbb{P}(s_{t+1} = j | s_t = i, s_0 = i_0, s_1 = i_1, ..., s_{t-1} = i_{t-1}) = \mathbb{P}(s_{t+1} = j | s_t = i) \tag{4.3}$$

where i_k for all $k = 0, 1, 2, ..., t$ are possible past states of the stochastic process $\{s_0 = i_0, s_1 = i_1, ..., s_{t-1} = i_{t-1}\}$.

In the DTMC definition formula, $p_{i,j}(t)$ is called the transition probability from state i to state j on a state space at time t (Norris, 1997). Also, $p_{i,j}(a)$ can be called the probability of

reaching state j from state i after performing action a. The state transition matrix of the DTMC, $\mathbb{P}_{i,j}$, is formed by transition probabilities $p_{i,j}$ from i (all possible states) and j (all possible successor state) into a matrix ($\mathbb{P}_{i,j} = [p_{i,j}]$):

$$\mathbb{P}_{i,j} = \begin{pmatrix} p_{0,0} & p_{0,1} & p_{0,2} & \cdots \\ p_{1,0} & p_{1,1} & p_{1,2} & \cdots \\ p_{2,0} & p_{2,1} & p_{2,2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{4.4}$$

where $\forall i, j \in S$ and $\mathbb{P}_{i,j} \geq 0$. The transition matrix of the chain is stochastic. $\mathbb{P}_{i,j}$ is a square matrix with the size the state space if it is countable. Each row of the transition matrix $\mathbb{P}_{i,j}$ corresponds to all possible states and sums to 1 as $\sum_{i \in S} p_{i,j} = 1, \forall j \in S$. It is also possible to represent a Markov chain by a transition graph as illustrated in Figure 4.4.

4.4 An Example Markov Chain

To explain a Markov chain, we introduce here a simple grid world as our environment. The agent's potential movements around a maze environment can be modelled as a Markov chain. The example space is arranged in a four-by-four array of rooms, with doorways connecting the rooms, as shown in the Figure 4.3. It contains 13 reachable rooms (white colour) and 3 blocked rooms (black colour) from all 16 rooms. The number of room is denoted at lower right. This room number is also used to show the agent's state when it is in that room.

The state of starting point is represented for s_0, labelled room 1, which is also labelled with "S" (room 1) in Figure 4.3. The agent is in the state s_t after making t moves from starting state s_0 (room 1). The goal state is labelled with "G" (room 16). Although the starting and target point can be selected randomly, it is determined here as the room at the top-left for starting state and the bottom-right for target state, respectively.

Transition probabilities through the task can be specified by the transition matrix as described in Equation 4.4. In this example, making a Discrete Time Markov Chain model means that we define a 13x13 Markov transition matrix because of the number of reachable rooms.

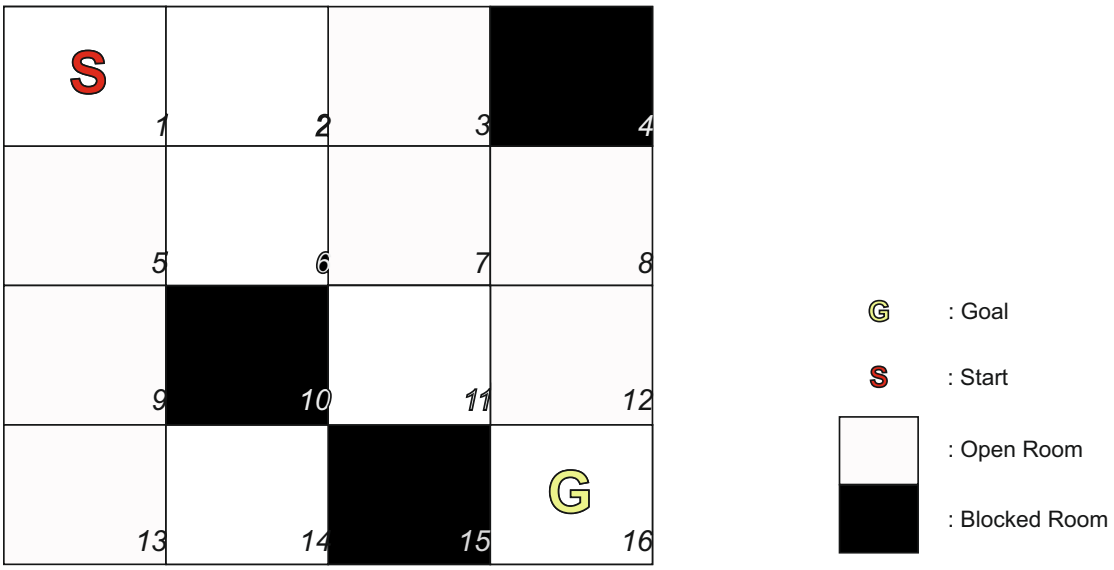


Figure 4.3: Possible states from a four-by-four maze platform. The space contains 13 reachable rooms (white colour) from all 16 rooms. The starting room and goal room are shown with "S" and "G", respectively. The number of each room is labelled at the bottom-right corner of each room in order to indicate the state in this place.

The position of our agent at each time step can be represented by $s_{t|t=0,1,2,3,...}$. Here, the agent as we discussed in section 4.3 does not have any memory of rooms visited previously. Then, if we rewrite the transition probability in Equation 4.3 by renaming current state as s and next state as successor state s', the state transition probability can be defined as:

$$\begin{aligned} P[s_{t+1}|s_t] &= P[s_{t+1}|s_1, s_2, ..., s_t] \\ P_{ss'} &= P[s'|s] \end{aligned} \tag{4.5}$$

based on the Equation 4.3.

Figure 4.4 shows a Markov chain with the initial transition probabilities and states based on previous maze environment. In this design, each field represents one state. If a room is reachable from other rooms, there is a room number within a circle. If there is no path from other rooms, only the place number in the state transition diagram is shown.

In the proposed model, we enable only horizontal and vertical movements $(\rightarrow, \downarrow, \leftarrow \rightarrow)$. For

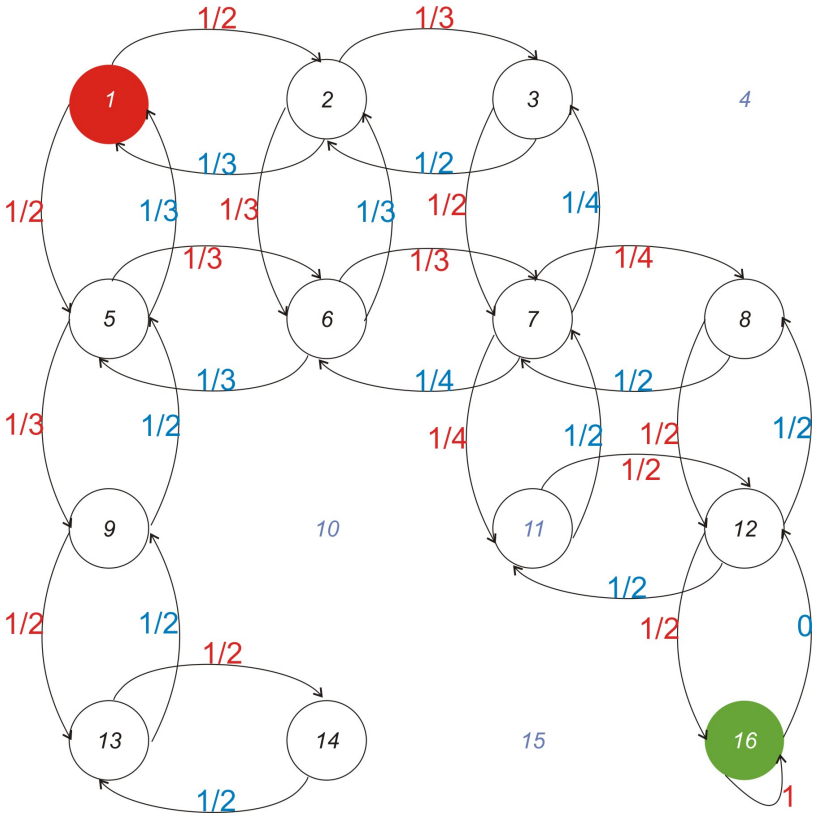


Figure 4.4: Discrete Time Markov Chain Graphic Model based on given maze in Figure 4.3 assuming that only horizontal and vertical movements are allowed. States are depicted as nodes, while possible actions are represented by the arrows. After removing unreachable states, there are 11 possible states with all 16 states (4x4 maze). The state space and state transition of the task are drawn here. Each number with a circle represents a possible state and each number without circle show impossible states (unreachable states). There are a maximum of 4 possible actions in each state: North, South, East and West.

instance, possible actions on some rooms can be listed as:

$A(1) = \rightarrow, \downarrow$

$A(2) = \leftarrow, \rightarrow, \downarrow$

...

$A(14) = \leftarrow$

$A(16) = \uparrow$

where in the form of $A(r)$, r is the room number. For instance, at room 1 there are only two possible actions as the directions of East ($\rightarrow$) and South ($\downarrow$).

The following matrix in Equation 4.6 illustrates the computation of the steady-state distribution of the maze's Markov chain. Each accessible room is considered as a separate state Markov chain with the state space $s_1, s_2, s_3, s_4, .., s_{16}$. Names for columns and rows in the equation indicate the state number/room number from/to, respectively. Each element inside the matrix

describes the corresponding transition probability. For instance, if the Markov chain is in state 1, s_1; it switches from the current state to one of the other possible states s_2, s_5, each with probability 1/2. If it is in state 2, s_2; then it switches to one of s_1, s_3, s_6 with probability 1/3, and so on. Thus, the transition matrix is shown in Equation 4.6. The zeros in the Equation 4.6 represent that there is not a link between states. For example, state 1 (s_1) cannot go to state 9 (s_9).

$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \end{array} \begin{array}{c} \begin{array}{cccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \end{array} \\ \begin{pmatrix} 0 & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 1/3 & 0 & 0 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/4 & 0 & 0 & 1/4 & 0 & 1/4 & 0 & 0 & 1/4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 0 & 1/3 & 0 & 0 & 0 & 0 & 1/3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \tag{4.6}$$

As a reward after each single trial, it can be given any reward value, although rewards are typically +1 or +100 (Tesauro, 1995; Sutton & Barto, 1998; Matignon et al., 2006; Nowe & Brys, 2016). Here our reward is chosen as 100 to demonstrate other state rewards as a percent of target state reward which is the maximum reward. Two example episodes ending in the goal state that produces a reward of 100 are given here:

$s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_7 \rightarrow s_8 \rightarrow s_{12} \rightarrow s_{16}$

$s_1 \rightarrow s_5 \rightarrow s_9 \rightarrow s_{13} \rightarrow s_{14} \rightarrow s_{13} \rightarrow s_9 \rightarrow s_5 \rightarrow s_6 \rightarrow s_7 \rightarrow s_{11} \rightarrow s_{12} \rightarrow s_{16}$

4.5 Markov Decision Process

A Reinforcement Learning task that satisfies the Markov property is called a Markov Decision Process (MDP). MDPs can be formally described as fully observable environments. The importance of MDP is almost all RL problems can be modelled as framework of finite Markov Decision Processs such as achieving the target inside the maze.

Based on the notation of Sutton & Barto (1998), a Markov Decision Process can be defined by its state and action sets:

- S is a set of finite states, $\forall s_t \in S$
- A is a set of finite available actions, $\forall a_t, \forall a \in A$

Based on Equation 4.3, important dynamics of MDP are described in RL context. The probability of each possible next state, called transition probabilities, is formulated as:

$$\mathbb{P}(s'|s,a) = \boldsymbol{Pr}\{S_{t+1} = s'|S_t = s, A_t = a\} \tag{4.7}$$

where s and a is any state and action, s' is any possible next state. Similarly, the expected value of the next reward can be formalized:

$$R(s,a,s') = \mathbb{E}\{R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'\} \tag{4.8}$$

The underlying mathematical model of the RL algorithms is the Markov Decision Process defined above. As an empirical domain to evaluate proposed algorithms, a maze navigation problem is commonly used. This is a well-defined and feasible platform for MDPs because MDP can be applied to the maze task. Therefore, in this project proposed algorithms are performed on the maze task that is shown in Figure 4.3.

4.6 Value Function

In the previous section, we define the Markov Decision Process in RL context but how should we solve MDPs or how can we find the optimal solution for them? The use of a reward to embody the idea of a goal is one of the most intrinsic features of RL (Sutton & Barto, 1998). The goal for the RL task is to maximize received reward in the long run. However, how this maximization is formulated? If we define total rewards through time, then maximizing this sequence of rewards can be the focus aspect for the objective.

Total reward is the sum of previous rewards as:

$$R_t = \overbrace{r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T}^{\text{sequence of future rewards after step t}}, \tag{4.9}$$

where R_t is total reward, t is time step and T is a final step if there is a terminal state. However, if the interaction of agent and environment does not break into sub episodes, the final time step would be $T = \infty$. However, in our maze task each learning cycle ends in a

goal room; so, T is finite.

Also, the discounting concept is important in this approach (Sutton & Barto, 1998). Best action is chosen in order to maximize the expected discounted return:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \\ &= \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \end{aligned} \tag{4.10}$$

where γ is a discount factor in order to determine the present value of future rewards with $0 \leq \gamma < 1$. $r_t, r_{t+1}, ...$ are generated by following policy π starting at state s. Hence, the discount factor γ quantifies how much importance it is given for future rewards against to immediate rewards. If the discount factor is closer to zero, the agent tends to consider only immediate rewards. If the discount factor is closer to one, the agent considers future rewards with greater weight, willing to delay the reward.

Based on the agent's strategy, the value of each state is the expected reward starting from that state using the state-value function, $V_\pi(s)$, defined as:

$$\begin{aligned} V_\pi(s) &= E_\pi\{R_t | s_t = s\} \\ &= E_\pi\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} ... | s_t = s\} \\ &= E_\pi\{\sum_{i=0}^{\infty} \gamma^i r_{t+i} | s_t = s\} \end{aligned} \tag{4.11}$$

where $\pi(a|s)$ is the probability of taking action a in state s under policy π, $E\{\}$ is the expectation operator, and $E_\pi\{\}$ is the expectation under policy π. E_π is subscripted by π to indicate that they are conditional on policy π being followed.

The action that an agent takes in any given state is called the policy or strategy,π_t at step t, which is solution for the MDP (Sutton & Barto, 1998). In other words, a policy is a mapping from possible states to action probabilities as $\pi_t(s, a)$, where $a_t = a$ and $s_t = s$.

Based on Markovian Property, the objective is to maximize the expected return, $E_\pi\{R_t\}$, for each step t, with an optimal policy on how the agent should achieve the goal. Similarly, the

value of taking action a in state s under a policy π, the action-value function ($Q_\pi(s)$), can be demonstrated (Sutton & Barto, 1998) by:

$$\begin{aligned} Q_\pi(s) &= E_\pi\{R_t|s_t = s, a_t = a\} \\ &= E_\pi\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2}...|s_t = s, a_t = a\} \\ &= E_\pi\{\sum_{i=0}^{\infty} \gamma^i r_{t+i}|s_t = s, a_t = a\} \end{aligned} \tag{4.12}$$

where $\pi(a|s)$ is the probability of taking action a in state s under policy π, $E\{\}$ is the expectation operator, and $E_\pi\{\}$ is the expectation under policy π.

4.7 Bellman Equations

The Bellman equation or dynamic programming equation is proposed by Richard Bellman for objective functions to minimize the cost or maximize the reward of multi-stage decision problems (Bellman, 1957; Dreyfus, 2002). In other words, the proposed equations help to compute the state-value function V_π for an arbitrary policy π. The Bellman equations express a relationship between state s and its successor states *s'*. Therefore, the objective function is typically keeping the evolution of decisions over time.

$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \gamma^4 r_{t+4}... \\ &= r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4}...) \\ &= r_t + \gamma R_{t+1} \end{aligned} \tag{4.13}$$

where R_t is the accumulated reward at time t, T is the final step and and $0 \leq \gamma < 1$, is the discount factor for future rewards.

In Equation 4.13, the reward gained with action a at time step t is r_t. The rest of the reward ($\gamma r_{t+1}+\gamma^2 r_{t+2}+\gamma^3 r_{t+3}+\gamma^4 r_{t+4}...$) or ($\gamma R_{t+1}$) can be described using the following policy.

Therefore, if we rewrite value function from Equation 4.11, the value of a state is the expected

return starting from that state; which depends on the agent's policy, π:

$$\begin{aligned} V_\pi(s) &= E_\pi\{R_t|s_t = s\} \\ &= E_\pi\{r_{t+1} + \gamma V(s_{t+1})|s_t = s\} \\ &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[r^a_{ss'} + \gamma V_\pi(s')] \end{aligned} \tag{4.14}$$

where $E\{\}$ is the expectation operator, and $E_\pi\{\}$ is the expectation under policy π, R_t is the accumulated reward, t is time step, r_{t+1} current reward at time step $t+1$, γ is the discount factor for future rewards with $0 \leq \gamma < 1$, s_{t+1} current state at time step $t+1$, $P^a_{ss'}$ is the state transition probability from current state s to next Markov state as its successor states s' with the performed action a, previously described in Equation 4.5. This function expresses a recursive relationship between the value of a state and the value of its successor state.

4.8 Optimisation

Up to this point, we have formally introduced the Markov property and Markov Decision Process. In addition, the solution of MDPs is also formulated with state-value and action-value functions. However, there is always at least one or more policies that is better than, or equal, to all the other strategies. This is called optimal policy, π^*. Therefore, this section briefly describes the value or policy optimisation algorithms.

The efficient method for solving decision problems is the same as determining the optimal value function for them. Therefore, the computing and learning task is to learn the optimal policy. A policy π is defined to be better than or equal to a policy $\pi^{'}$ if its expected return is greater than or equal to that of for all states (Sutton & Barto, 1998) :

$$\pi \geq \pi^{'} if V_\pi(s) \geq V_{\pi'}(s) \forall s \in S \tag{4.15}$$

There is always at least one or more policies that is better than or equal to all the others which is called optimal policy π^*.

$$\pi^* = argmax_\pi V_\pi(s), \forall s \in S \tag{4.16}$$

Optimal policy is based on optimal state-value function:

$$V_{\pi^*}(s) = max_{\pi} V_{\pi}(s), \forall s \in S, a \in A(s) \tag{4.17}$$

where $V_{\pi}(s)$ is the state-value function described in Equation 4.11. Similarly, an optimal action-value function is:

$$Q_{\pi^*}(s, a) = max_{\pi} Q_{\pi}(s, a), (\forall s \text{ and } \forall a) \tag{4.18}$$

where $Q_{\pi}(s, a)$ is the action-state function described in Equation 4.12. $Q_{\pi^*}(s, a)$ describes the expected return for taking action a in state s based on an optimal policy.

4.9 Algorithms for Reinforcement Learning

Policy search is important in RL which focuses on finding efficient parameters for a given policy parametrization (Deisenroth et al., 2011). Recent studies focus on both model-based and model-free policy search. Model-based techniques build a model of the current task's structure through evaluating candidate actions (Dayan & Abbott, 2005; Rangel et al., 2008). In contrast, model-free methods such as Temporal-Difference Learning, Q-Learning, described in subsection 4.9.3, can learn a policy without any initial model (Sutton & Barto, 1998; Deisenroth et al., 2011). The primary goal of model-free and model based learning is the improvement of a behavioural policy in order to get a maximum reward. However, the agent has to learn a model of its environment in the model-free method but the agent in the model-based method already has a model as a useful entity. Therefore, a model-based agent can generate virtual experiences and can plan future actions. The drawback of model-based approaches is that acquiring a model initially increases the complexity of the problem, and inaccurate models can impair the performance compared with the model-free RL algorithms.

Model-based learning has other challenges such as learning the model, and using it intelligently to improve the performance. Furthermore, for our maze task, we do not have an initial model of the environment, and any model of our task is randomly changed for each experiment. Therefore, a model-free Reinforcement Learning approach is well fitted with our problem. Hence, the next section focuses on model-free approaches.

4.9.1 Temporal-Difference Learning

One of the most popular ways in order to estimate the future discounted reward of states is the method of Temporal-Difference (TD) learning in the field of Reinforcement Learning. Hence, using first explicit presentation in the context of Markov Decision Process by Witten (1977) TD learning is summarized in this section.

TD methods calculate an estimate of the total reward at each state formulated in the Equation 4.14. Then, the value of state-action is updated at every step with the way formalized as:

$$V(s) \leftarrow V(s) + \alpha[R(s, a, s') + \gamma V(s') - V(s)] \tag{4.19}$$

where α is the learning rate, γ is the discount factor, a is the action from s current state to next state a'. This type of Temporal-Difference learning is known as TD(0).

4.9.2 Eligibility Traces

One shortcoming of TD(0) case is that at each time step the features of only the current state are eligible to be updated. States before the current state are also affected by changes in the current state's features. Hence these features can also be updated. This is what happens with a new variable called activity or eligibility trace summarized by Sutton & Barto (1998); Wickens (2009). Here, whenever a state is visited, its eligibility becomes high and then gradually decays until it is visited again.

Eligibility for each state s at time t is computed by:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t. \\ \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t, \end{cases} \tag{4.20}$$

where γ is the discount factor, λ is a decay parameter called trace-decay which defines the update weight for each past state, $\lambda \in [0, 1]$. Hence, all traces on each state decay by $\gamma\lambda$. The Temporal-Difference update for all states s is:

$$V(s) \leftarrow V(s) + \alpha e_t(s)[R(s, a, s') + \gamma V(s') - V(s)] \tag{4.21}$$

This update scheme is known as TD(λ) (Sutton, 1988). For $\lambda = 0$, this is equivalent to TD(0) learning adjusts value estimates only one step ahead as summarized earlier in section 4.9.1. By substituting $\lambda = 0$ in Equation 4.20, eligibility for TD(0) can be formulated:

$$e_t(s) = \begin{cases} 1 & \text{if } s = s_0, \\ 0 & 0 \text{ otherwise.} \end{cases} \tag{4.22}$$

If $0 < \lambda < 1$, the trace decreases in time. For $\lambda = 1$, this is equivalent to TD(1) learning.

4.9.3 Q-Learning

As a well-known model-free Reinforcement Learning type, Q-Learning is introduced here to determine the policy for the maze problems. Q-Learning is a specific TD algorithm in order to learn the Q-function used to perform an action. At each step, we need to choose an action to maximize Q(s,a) in Equation 4.23 which tells us how good an action from the current state to the next state is (s $\stackrel{a}{\Rightarrow} s'$).

$$Q(s,a) \leftarrow [1-\alpha]Q(s,a) + \alpha[R(s,a,s') + \gamma \max_a Q(s',a)] \tag{4.23}$$

Following action is selected according to $a \leftarrow arg\max_a Q(s',a)$ which is greedy. Other choices for the action selection are possible, e.g. ϵ-greedy with decaying ϵ. Based on the chosen action, the agent gets an immediate reward and the values of the Q utility function are updated based on learning rate α, discount factor γ and the action taken a from state s to following state s'. This procedure is executed recursively until the end of the decision in which the exploration task is completed as described in Algorithm 1. At the beginning of the procedure, entire elements of Q table for all actions at each state are reset to 0.

As an example problem, we use the maze scenario in Figure 4.5. Blue rooms indicate the current positions/states in the maze, $a_{s,s'}$ corresponds to the action from s (current state) to selected next state, s'. For instance, in Figure 4.5b, from state 1 to state 2 is executed by the selection of action $a_{1,2}$ from the two available actions: $a_{1,2}$ and $a_{1,5}$. Then, it is updated as in Figure 4.5c. Blue arrows show the selected actions through the current path.

Algorithm 1 Pseudocode of Q-Learning Algorithm

```
1: procedure Q-Learning(s, a, Q)
2:     for s ← s_1, s_n do
3:         for a ← a_1, a_m do
4:             Q'(s, a) ← 0
5:         end for
6:     end for
7:     Observe the current state, s
8:     repeat
9:         Select an action, a ∈ {a_1, a_2, a_3, ..., a_m}
10:        Execute the selected action, a through s' ← γ(S, a)
11:        Receive an immediate reward, r, r(s, a)
12:        Observe a new state, s'
13:        Update the Q table entry as in Equation 4.23
14:        Assign the next state to the current state, s ← s'
15:    until ¬forever
16: end procedure
```

The actions that lead to the goal have a maximum determined reward for the task. Other actions not directly connected to the target are labelled non rewarding ones. The updating of Q function is demonstrated above through an example successful route which arrives at the goal. The same route can be seen in Figure 5.2a.

- Choose $a_{1,2}$ from available actions : $a_{1,2}, a_{1,5}$

  $Q(s_1, a_{1,2}) = r + \gamma * max(Q(s_2, a_{2,1}), Q(s_2, a_{2,6}), Q(s_2, a_{2,3})) = 0$

- Chose $a_{2,3}$ from available actions : $a_{2,1}, a_{2,6}, a_{2,3}$

  $Q(s_2, a_{2,3}) = r + \gamma * max(Q(s_3, a_{3,2}), Q(s_3, a_{3,7})) = 0$

- Choose $a_{3,7}$ from available actions : $a_{3,2}, a_{3,7}$

  $Q(s_3, a_{3,7}) = r + \gamma * max(Q(s_7, a_{7,3}), Q(s_7, a_{7,6}), Q(s_7, a_{7,8}), Q(s_7, a_{7,11})) = 0$

- Choose $a_{7,11}$ from available actions : $a_{7,3}, a_{7,6}, a_{7,8}, a_{7,11}$

  $Q(s_7, a_{7,11}) = r + \gamma * max(Q(s_{11}, a_{11,7}), Q(s_{11}, a_{11,12})) = 0$

- Choose $a_{11,12}$ from available actions : $a_{11,7}, a_{11,12}$

  $Q(s_{11}, a_{11,12}) = r + \gamma * max(Q(s_{12}, a_{12,8}), Q(s_{12}, a_{12,11}), Q(s_{12}, a_{12,16})) = 0$

- Choose $a_{12,16}$ from available actions : $a_{12,8}, a_{12,11}, a_{12,16}$

  $Q(s_{12}, a_{12,16}) = r + \gamma * max(Q(s_{12}, a_{12,8}), Q(s_{12}, a_{12,11}), Q(s_{12}, a_{12,16})) = 100$

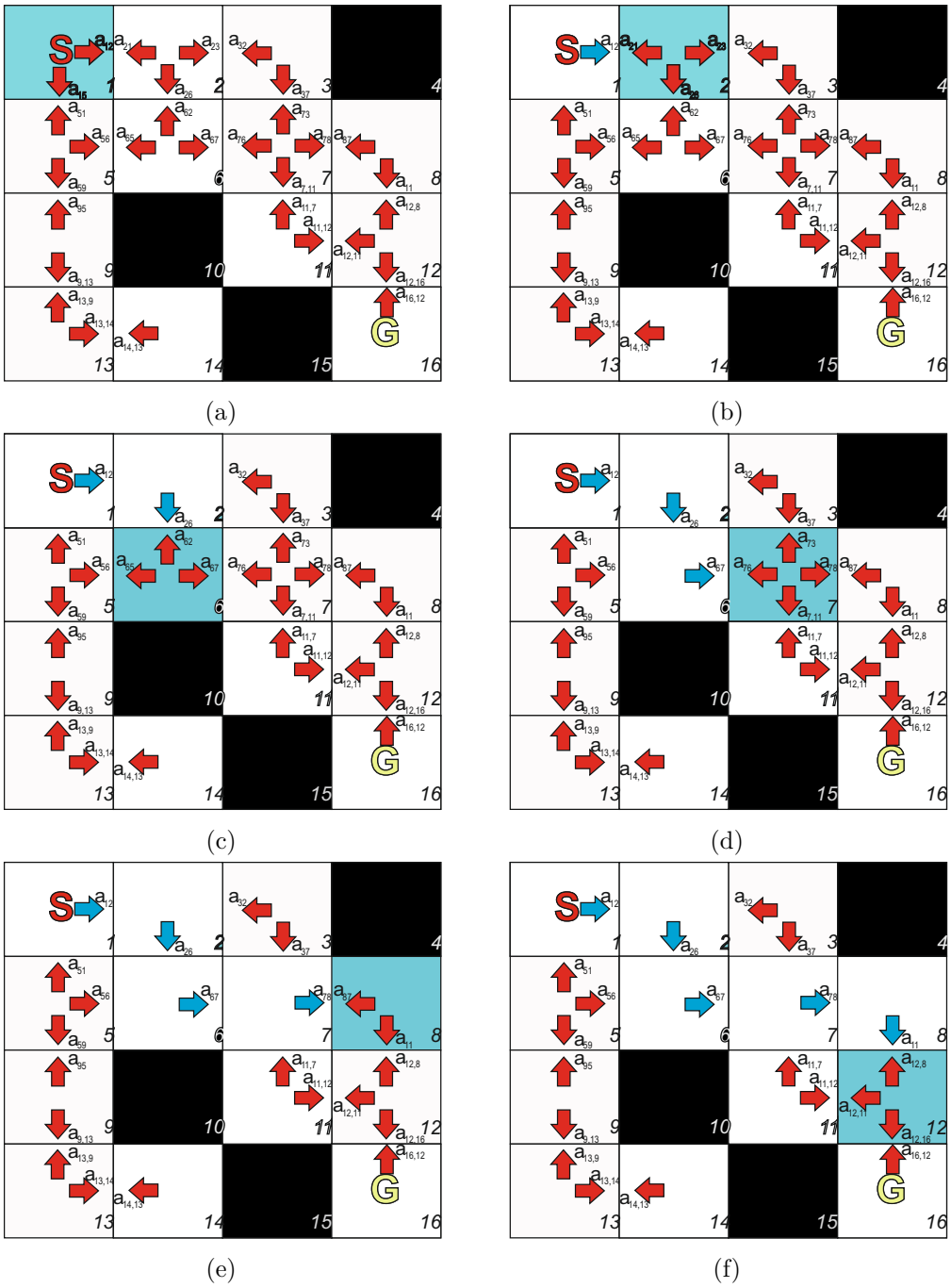
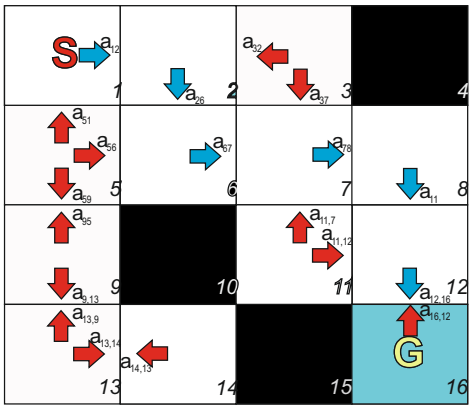


Figure 4.5: Maze task with all possible actions and their action names with an example route from start to target.

All steps of first episode are shown step by step in Figure 4.5. To give a simple example on this type of learning, we describe corresponding calculation of Q-table for each action/state transition above.



(g)

Figure 4.5: Maze task with all possible actions and their action names (continued). Maze task with all possible actions (red arrows) and their action names. For instance, once the action a_{12} is selected at room 1, s_1, to perform through a_{12} and a_{15}, the new state became s_2 as in (b). In the action subscript, first element shows target state and the second is the next state. Rooms with blue is active state in this time step. The movements of state flows from (a) to (g). Blue arrows show selected actions.

4.9.4 Replacement Rules for Maze Navigation

A hypothetical maze task running from start to finish is used as a benchmark application. Maze task is widely used for RL tasks (Kaelbling et al., 1996; Sutton & Barto, 1998; Matignon et al., 2006; Vasilaki et al., 2009; Venkata et al., 2011; Fremaux et al., 2013; Nowe & Brys, 2016). Hence, we use a maze task in order to investigate fundamentals of RL and reward mechanisms throughout practical cases. Furthermore, we test the hypothesis that if RL is combined with another technique, the learning speed up. For this purpose, we develop an approach to combine Q-Learning and extracted novel rules named Replacement Rules on the maze task.

Additional knowledge about the task can improve the performance of learning in terms of finding shortest path on mazes (Venkata et al., 2011). The approach through a short list of Replacement Rules from Venkata et al. (2011) summarized in Table 4.1 can increase the learning speed by eliminating some of dead-ends. This approach is extended by adding a number of novel rules with the proposed algorithmic generation in Figure 5.3 in order to eliminate entire dead-ends.

The shortest path through the maze environment never includes a dead-end. Therefore, every dead-end indicates the agent takes a mistaken turn in the maze. In other words, a dead-end

can be considered as the previous action is not optimal and needs to be modified. In this stage, Replacement Rules is proposed based on selected wall follower algorithm (see section D.4).

For simplicity, we start with wall follower algorithm as a maze solving algorithm. Wall follower algorithm (see section D.4), is also known as either the left-hand rule (LSR) or the right-hand rule (RSL), is very fast and uses no extra memory. From the start of the maze, the agent or robot always turns to the left (or right) wall side whenever the agent reaches a junction and navigates throughout the maze until it reaches the destination point. Wall follower algorithm is proven to be very efficient for maze environments that are wall-linked to the target point (Mishra & Bande, 2008). However, one of disadvantages of wall follower algorithms is that they cannot find the shortest path.

In order to achieve the best maze solution, the agent needs to traverse the maze twice in this algorithm. In the first run as learning phase, it applies directly LSR/RSL Rule (see subsection D.4.1 and subsection D.4.2). We record only turn directions during the first run. As soon as first run is completed, the algorithm of Replacement Rules is applied to the stored directions in order to find optimum path from the start to the goal. Taken the sequence of turns from the first run phase is simplified using the Replacement Rules as they are explained in section 5.3. Therefore, all unnecessary turns by finding the U turns made during the previous period can be avoided and eliminated on the second run via the replacement of the corrected ones.

Three Replacement Rules for LSR (see subsection D.4.1) are proposed from Venkata et al. (2011) as in Table 4.1. This approach with the list of rules in Table 4.1 is named Basic Replacement Rules for LSR (BscRepLSR) for further comparisons.

LUL $\Rightarrow$ S
LUS $\Rightarrow$ R
SUL $\Rightarrow$ R

Table 4.1: Basic Replacement Rules for LSR (BscRepLSR): Replacement Rules from Venkata et al. (2011).

Each sequence in Table 4.1 has at least one U turn. Anywhere the stored sequence consists of those sequences, it is replaced with the suggested movement (on the right in Table 4.1).

For instance, first Replacement Rule is the replacement of the sequence “LUL” with the “S”. After this replacement, an U turn movement is eliminated from the sequence of movements, resulting in shortened path.

4.10 Summary

Any problem that can be described as a Markov Decision Process can potentially be solved through a Reinforcement Learning technique. RL techniques do not require explicit input or output for training phase. Therefore, RL techniques such as Q-Learning become more popular and effective for the set of practical problems such as path navigation. Based on two common RL methods: Knowledge-Based RL and Q-Learning, the details of this algorithm are illustrated throughout the chapter.

In this chapter, we also introduce the general overview of the three-factor learning rule in Spiking Neural Networks. The regulated form of original associative plasticity by Dopamine as the three-factor rule is summarized for the basis of synaptic plasticity and memory formation in SNNs (see chapter 9). Also eligibility trace, a transient memory of past events, is examined which is the fundamental component in the proposed spiking learning in chapter 9. Those concepts in RL are extended for synaptic plasticity in Spiking Neural Networks in chapter 9.

An important goal of this chapter is to describe the computational foundations for dealing with problems like states and actions in the context of RL mechanism. In addition, this chapter examines a number of underlying concepts for the following chapters such as Temporal-Difference mechanism as an attractive formulation of RL. Reward-modulated Spike-timing Dependent Plasticity in chapter 9 as a biologically plausible learning for a network of spiking neurons is derived from the continuous Temporal-Difference formulation.

Although the agent starts learning with limited knowledge about the domain during explorations in RL tasks, we try to speed up the learning through more extracted features about the maze task. The list of extracted rules for the current domain, maze task, is named Replacement Rules. The methods of Replacement Rules, TD Learning with and without eligibility trace is illustrated on the maze using wall followers and RL approaches in chapter 5.

The improvements of Replacement Rules on TD-Learning through maze task are analysed with the extension of current Replacement Rules.

Chapter 5

Extension of Replacement Rules with Reinforcement Learning for Maze Navigation

Contents

5.1 Introduction 119
5.2 Experimental Setup 120
5.2.1 Task Description 121
5.2.2 Maze Generation 123
5.3 Extension of Replacement Rules 123
5.3.1 Algorithmic Generation of Replacement Rules 127
5.3.2 Comparison to Older Replacement Rules 129
5.4 Replacement Rules with Temporal-Difference Learning 133
5.4.1 Applying to Temporal-Difference Learning 133
5.4.2 Comparison of Proposed Algorithms 134
5.5 Summary 136

5.1 Introduction

One of the important objectives in autonomous mobile robotics is to find an optimum path from a predefined start point to given destination. Several solutions for the path planning problems are proposed based on various assumptions. In particular, to construct a mobile robot later, we illustrate a simulated path planning algorithm using construction of a map of an initially unknown environment based on Reinforcement Learning (RL) (Sutton & Barto,

1998) detailed in chapter 4. The application of the methods described in chapter 4 to a maze task is considered throughout this chapter.

In RL tasks, the agent starts learning with very limited knowledge about the domain. One of the approaches of improving the performance of RL is the using of knowledge about the domain. A knowledge based RL approach is developed to achieve an optimum return path which allows a goal-oriented mobile robot to adapt to an unknown maze task. Therefore, here we extract some properties of the domain in order to help to achieve more optimal exploration periods. The approach in Venkata et al. (2011) summarized in Table 4.1 is extended by adding a number of new rules as listed in subsection 4.9.4. We name this technique the Replacement Rule. It is thought that the same rule set can also be applied into Reinforcement Learning techniques for maze navigation such as Q-Learning. We achieve this combination with better performance in section 5.4.

Section 5.2 presents an overview of the experimental task. Following that, the dynamic generation of Replacement Rules is demonstrated algorithmically. Performance improvements through various maze sizes are demonstrated by the extended list of Replacement Rules in Figure 5.3. Then, extended list of Replacement Rules is applied into Temporal-Difference Learning with and without eligibility traces in section 5.4.

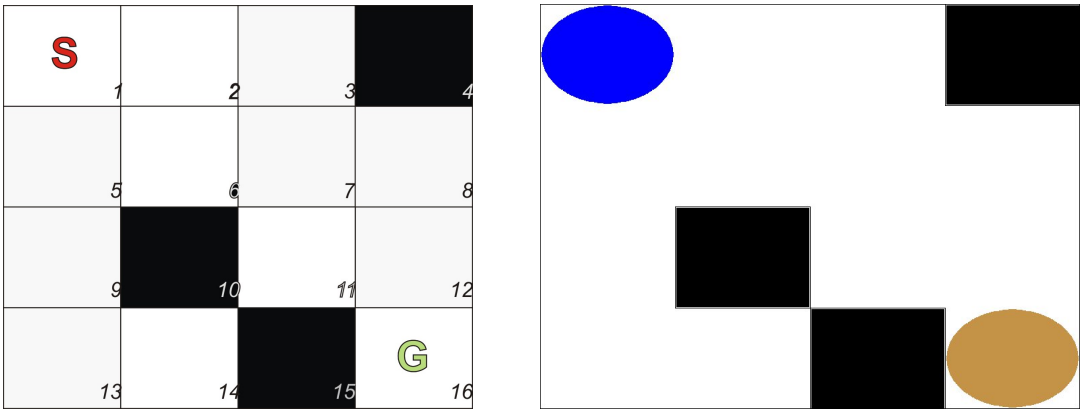
5.2 Experimental Setup

To navigate from start to target on a maze is the experimental task detailed in section 5.2.1. We start with one of wall follower algorithm as the left-hand rule (see section D.4). The left-hand rule, LSR rule, states Left direction has highest priority compared to Straight and Right directions while there are options for turns. Likewise, Straight has higher priority than Right turn.

The performance of three different algorithms as LSR Only, LSR with Venkata et al.'s rules and LSR with proposed rules are compared throughout four different maze sizes as 5x5, 10x10, 15x15 and 20x20 mazes. The detail of maze generation through experimental task is described in section 5.2.2.

5.2.1 Task Description

The task is to navigate from start (S) to goal (G) as fast as possible on gridworld. We have two similar representations of experimental tasks. The maze task in Figure 5.1a and in Figure 5.1b consists of 4x4 rooms as an illustration sample. In Figure 5.1a drawn by drawing tools, black rooms are inaccessible rooms. If all four physical walls/doors for a room exist, this room is inaccessible. This graph illustrates starting point of the agent by "S" and the goal room as "G". Also, the name of each room is denoted with a number at the right-bottom corner. For the scenarios of theoretical demonstrations, we use this representation throughout the chapter. On the other hand, for the representation of simulation results, we use the directly captured representation from the simulation environments as in Figure 5.1b. Here, rooms are separated by physical walls. The starting position for the agent is situated at the top-left corner, highlighted with a blue circle. The goal room is situated at the bottom-right, highlighted with a brown circle.

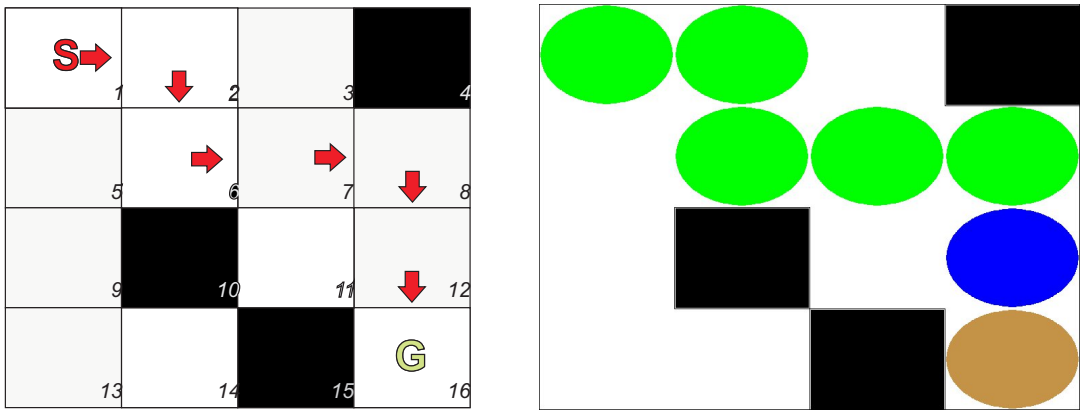


(a) Drawn graph. (b) Graph from the simulation environment.

Figure 5.1: 4x4 maze platform from drawn (a) and the simulation environment (b). (a) It is used in order to illustrate algorithm, task details manually using drawing tools. (b) It is prepared in the simulation environment using Python.

As an example path between start and goal is also demonstrated through the graph representation of simulated agent in Figure 5.2a and the captured screen of simulated agent in Figure 5.2b on the 4x4 maze task.

In the RL context, using the Markov Decision Process in Figure 4.4, it can be seen that there are 11 possible states. The state space and state transition of 4x4 grid maze environment can be visualized as in Figure 4.4. Each number with a circle represents a possible state and each



(a) Drawn graph. (b) Graph from the simulation environment.

Figure 5.2: An example path from starting room to goal through (a) manually drawn and (b) captured from the simulation environment. The initial or starting state is the top-left corner and the goal state is the bottom-right corner.

number without circle shows impossible states. An example trial is illustrated in Figure 5.3.

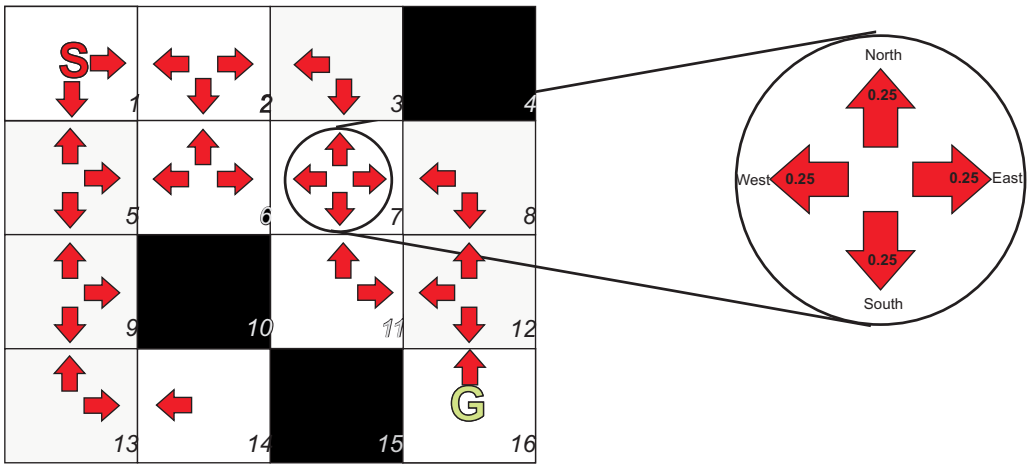


Figure 5.3: Maze platform with all possible directions and an example of possible initial possibilities on a specific room. There are a maximum of 4 possible actions in each state: North, South, East and West.

- Each room is called as a state s. Hence, the set of all possible states S on current task can be formulated as:

  S = {1,2,3,5,6,7,8,9,11,12,13,14,16}; s,$s' \in S$

- The agent's movement from one room to another is an action a. There are 4 primitive actions, for taking steps. The set of all possible actions A can be illustrated as:

  $A = \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$; a $\in A$

- $if(12 \downarrow 16)$ means that the agent moves from state 12 to 16 by the action of $\downarrow$. The

following successor state for state 12 is state 16 so the agent gets the maximum reward for this movement because this movement causes to achieve the goal state 16 from the closest state 12.

5.2.2 Maze Generation

The purpose of maze generation is to provide a task, which is a challenging maze environment in our experiments, to test the learning improvements. We generate a maze which includes a predetermined starting point and a predetermined goal point. Then, the agent finds a route between two particular nodes using the selected algorithm. On the other hand, if any trace of the maze environment has a circular chain as it is illustrated in Figure 5.4 or if any cell in this task has no eligible actions, it means that a deadlock state can occur there because there is no outgoing transition. Fortunately, we do not focus on deadlock scenarios during our experiments. Therefore, we do not use any maze tasks with dead ends.

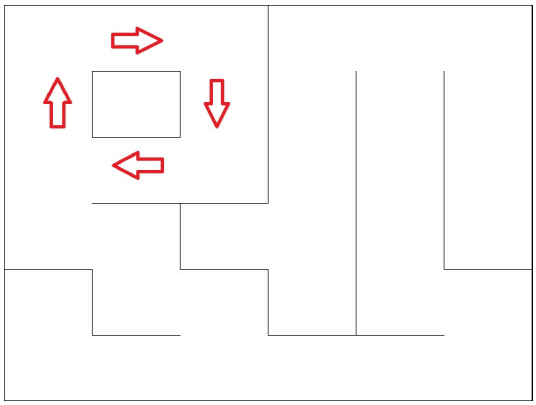


Figure 5.4: An example for deadlock state scenario.

In order to generate a maze task, there are a number of algorithms such as Depth-first search, Recursive backtracker, Randomized Kruskal's algorithm (Even, 2011). However, as an initial implementation, we use here a Recursive algorithm which is illustrated in Figure D.9. Details of the algorithm are described in section D.3.

5.3 Extension of Replacement Rules

Three Replacement Rules summarized in Table 4.1 from Venkata et al. (2011) is named Basic Replacement Rules for LSR (BscRepLSR) in section 4.9.4. BscRepLSR works fine in small

mazes. However, it cannot find the optimum path in bigger mazes. Therefore, we extend the list of Replacement Rules in order to find the best path even in bigger mazes. In addition to the extended list of rules, algorithmic approach is proposed in order to get more rules on different maze environments have different dynamics.

For the notation, initial letters are used as a symbol where L - Left turn, R - Right turn, S - Straight, U - U turn. At the end of the first run, the algorithm focuses on the U turns and selecting two more turns which are before and after turns than selected U turn.

Main approach to generate a set of rules is discussed through an example rule. For instance, one of the Replacement Rules scenario can be the replacement of "LUR" because it has at least one U turn in the sequence. Anywhere the stored sequence consists of "LUR", it is replaced with U turn (U) as demonstrated in Figure 5.5. After this replacement, the algorithm seeks next U turns in the memorized turn sequence and finally it runs until the path does not have any U turn.

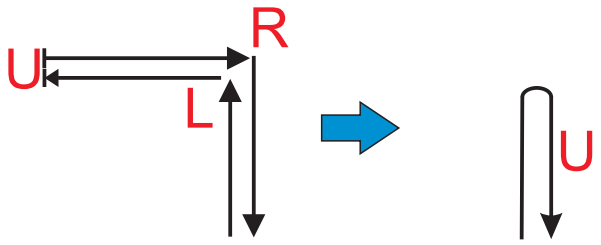


Figure 5.5: Replacement Rule LUR to U. L stands for left turn, U stands for U turn, and R stands for right turn.

Entire proposed list of novel Replacement Rules is summarized in Table 5.1 for LSR (see subsection D.4.1) to find the most optimum path. Extended version of BscRepLSR for LSR is named Extended Replacement Rules for LSR (ExtRepLSR). Initial exploration is executed by applying LSR rule to catch those rules listed in Table 5.1.

The right-hand rule, RSL rule, is similar to LSR, the only difference lies in the wall being followed. The RSL rule (see section D.4) states Right direction has highest priority compared to Straight and Left directions while there are options for turns. Likewise, Straight has higher priority than Left turn. The same replacement procedure for LSR rule can be executed through RSL, Right-Straight-Left exploration in order to extend rule list. This new list of

RUL ⇒ U
LUR ⇒ U*
SUS ⇒ U
RUS ⇒ L
SUR ⇒ L
RUR ⇒ S

Table 5.1: Extended Replacement Rules for LSR (ExtRepLSR): Replacement Rules from proposed list through LSR.

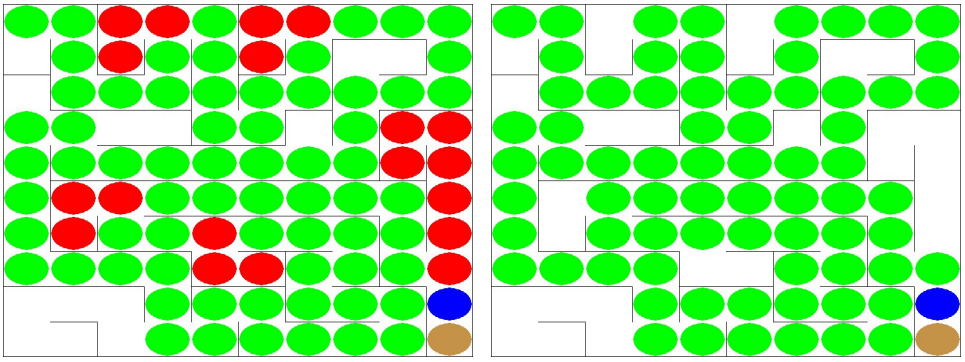
Replacement Rules is summarized in Table 5.2 for RSL similar to LSR rules. Inspiring from the BscRepLSR for LSR, this rule for RSL is named Extended Replacement Rules for RSL (ExtRepRSL). The applied algorithm for initial explorations just helps to have different variations of movement sequences. Therefore, proposed rules are not dependent the algorithm is applied during explorations. The combined version of Extended Replacement Rules for LSR in Table 5.1 and Extended Replacement Rules for RSL in Table 5.2 can be used as an extracted list for Replacement Rules.

RUR ⇒ S
RUS ⇒ L
SUR ⇒ L
RUS ⇒ U
RUL ⇒ U
SUR ⇒ L
SUS ⇒ U
LUR ⇒ U

Table 5.2: Extended Replacement Rules for RSL (ExtRepRSL): Replacement Rules from proposed list through RSL.

For instance, as initial joint movements based on LSR rule: ['L', 'L', 'U', 'S', 'L', 'U', 'S', 'L', 'S', 'L', 'L', 'L', 'L', 'U', 'L', 'L', 'U', 'S', 'L', 'U', 'S'] as can be seen in Figure 5.6a.
Then, the sequence of replaced movements: ['L', 'R', 'R', 'L', 'S', 'L', 'L', 'L', 'S', 'R', 'R'] can be seen in Figure 5.6b.

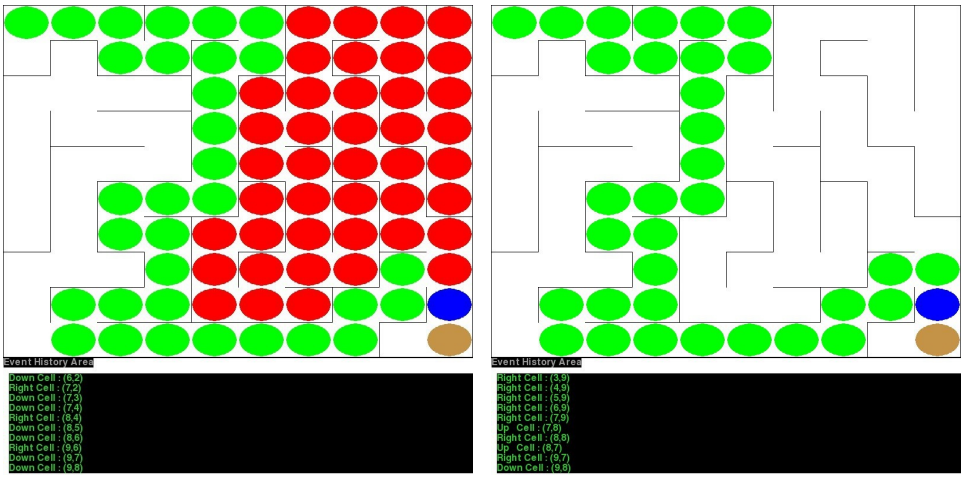
Another example, Figure 5.7a illustrates the end of the first run and Figure 5.7b demonstrates the end of the second run. The improvements between the two can be seen clearly. Initial phase from start to end based on LSR rule states Left direction has highest priority and then



(a) First Run. (b) Second Run.

Figure 5.6: First and second run screenshots (Experiment 3). Red circles represent more than one time visited rooms, green circles illustrate only one time visited rooms, blue circle demonstrates current visited room and finally brown circle shows the target point.

Straight, Right respectively (see subsection D.4.1). At the end of the first run, we replace movements with dead ends into without dead-end sequences in order to shorten the path from start to target. This rule can be executed until there are no further dead ends in the performed sequence.



(a) First Run. (b) Second Run.

Figure 5.7: First and second run screenshots (Experiment 1). Red circles represent more than one time visited rooms (named revisited rooms), green circles illustrate only one time visited rooms, blue circle demonstrates current visited room and finally brown circle shows the target point.

The same scenario happens in another example as can be seen in Figure 5.8a and Figure 5.8a. Pseudocode of the way how Replacement Rules applied is demonstrated in Algorithm 2.

Algorithm 2 Pseudocode of Applying Replacement Rules

```
procedure REPLACEMENT(self)
    replaced ← ' '
    self.temp ← self.historyForDirections
    i ← 1
    repeat
        if self.historyForDirectionsReplaced[i] is 'U' then
            if self.temp[i − 1] is 'l' and self.temp[i + 1] is 'l' then
                replaced ←' s'
            else if self.temp[i − 1] is 'l' and self.temp[i + 1] is 's' then
                replaced ←' r'
            else if self.temp[i − 1] is 's' and self.temp[i + 1] is 'l' then
                replaced ←' r'
            end if
        end if
        if replaced isnot ' ' then
            Pop last three items of self.historyForDirectionsReplaced
            Insert replaced direction to current list
            i ← i
        else
            i ← i + 1
        end if
    until ¬Scanned all directions in the array
end procedure
```

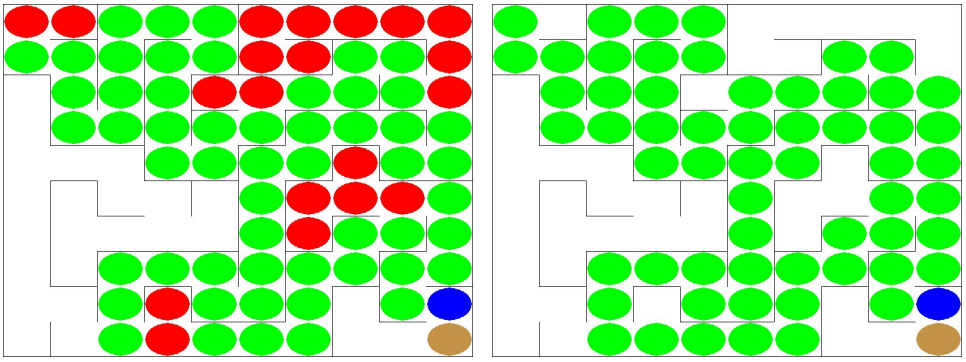
5.3.1 Algorithmic Generation of Replacement Rules

Algorithm 3 generates a list of rules dynamically. The algorithm can be applied real time or at the end of full path until finding target. Maze positions are identified by (x, y) coordinates. A y coordinate corresponds to a row index of maze rooms and an x coordinate corresponds to a column. Also, we assume that left-top corner is with the coordinate of (0,0).

In Algorithm 3, variables of d_0, d_1, d_2 symbolize consecutive three directions from the entire direction list d at intersection points and dead ends. x_0^b, x_1^b, x_2^b and y_0^b, y_1^b, y_2^b are x and y coordinates before the directions d_0, d_1, d_2. x_0^a, x_1^a, x_2^a and y_0^a, y_1^a, y_2^a are x and y coordinates after the directions d_0, d_1, d_2. $(x_0^{bprev}, y_0^{bprev})$ is the pair of coordinates a movement before the last directional movement d_0 (coordinates just before the coordinate of (x_0^b, y_0^b)). Remember that the coordinate at just before the d_0 is (x_0^b, y_0^b). $(x_0^{bprev}, y_0^{bprev})$ is previous place than (x_0^b, y_0^b). The reason to store those coordinates is to determine actual direction once we shorten the path (see line 29).

Algorithm 3 Pseudocode of Algorithmic Generation for Replacement Rules

1: **procedure** UPDATESTORINGVARIABLES
2: Store d ▷ All direction information at intersection points and dead ends
3: Store x, y ▷ Coordinate information before and after intersection points and dead ends
4: **end procedure**
5:
6: **procedure** GENERATEREPLACEMENTRULE(d, x, y)
7: d_0, d_1, d_2 ▷ Consecutive 3 directions information from d
8: x_0^{bprev}, y_0^{bprev} ▷ Previous coordinates than the coordinate of (x_0^b, y_0^b)
9: x_0^b, x_1^b, x_2^b ▷ x coordinates before the directions of d_0, d_1, d_2
10: y_0^b, y_1^b, y_2^b ▷ y coordinates before the directions of d_0, d_1, d_2
11: x_0^a, x_1^a, x_2^a ▷ x coordinates after the directions of d_0, d_1, d_2
12: y_0^a, y_1^a, y_2^a ▷ y coordinates after the directions of d_0, d_1, d_2
13: **if** $((d_1 ==' U')$ and $(d2! = None))$ **then**
14: **if** $(x_0^b == x_1^b)$ **then** ▷ change on y direction
15: **if** $((y_0^b - y_1^b) == (y_1^b - y_2^b))$ **then**
16: Call the procedure of ExtractTheRule in line 26
17: **end if**
18: **else if** $(y_0^b == y_1^b)$ **then** ▷ change on x direction
19: **if** $((x_0^b - x_1^b) == (x_1^b - x_2^b))$ **then**
20: Call the procedure of ExtractTheRule in line 26
21: **end if**
22: **end if**
23: **end if**
24: **end procedure**
25:
26: **procedure** EXTRACTTHERULE(x, y)
27: ▷ Extract the replacement rule from the route to shortened one
28: $(x_0^{bprev}, y_0^{bprev}) \rightarrow (x_0^b, y_0^b) \rightarrow (x_2^a, y_2^a) \Longrightarrow$
29: $(x_0^{bprev}, y_0^{bprev}) \rightarrow (x_0^b, y_0^b) \rightarrow (x_2^a, y_2^a)$
30: **end procedure**



(a) First Run. (b) Second Run.

Figure 5.8: First and second run screenshots (Experiment 2). Red circles represent more than one time visited rooms (named revisited rooms), green circles illustrate only one time visited rooms, blue circle demonstrates current visited room and finally brown circle shows the target point.

Generation of Replacement Rules is active if there are at least three consecutive directions means that none of d_0, d_1, d_2 is empty and the middle direction is U turn at dead end. Remember that if there is a more than one option for directions, d is selected/updated.

In line 14 and line 18, we use $x_1^b = x_1^a$ and $y_1^b = y_1^a$, respectively, because of U turns at d_1. ExtractTheRule procedure can shorten and generate all possible listed rules.

5.3.2 Comparison to Older Replacement Rules

Four different maze sizes as 5x5, 10x10, 15x15 and 20x20 mazes are chosen in order to prove the algorithm efficiencies. There are three algorithms: LSR Only, LSR with Venkata et al.'s rules from Table 4.1 (BscRepLSR), and LSR with proposed rules from the combination of Table 5.1 and Table 5.2 (ExtRepLSR). Some of abbreviations used in figures are:

N_u : Number of uniquely visited rooms,
N_t : Number of places visited,
N_e : Number of steps during at least double or more times visited rooms.

Selected four different maze sizes are 5x5 mazes in Figure 5.9, 10x10 mazes in Figure 5.10, 15x15 mazes in Figure 5.11, 20x20 mazes in Figure 5.12. Each figures are averaged of 10 different randomly generated mazes. Error bars with the standard deviation (SD) and height

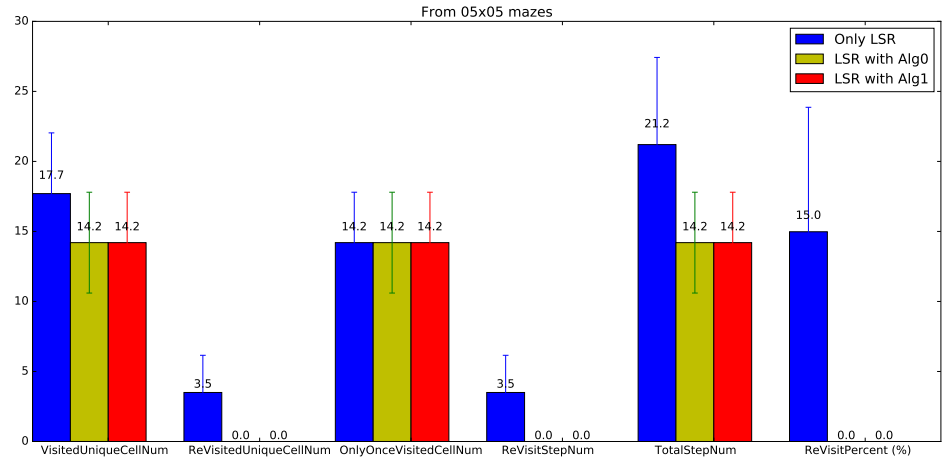


Figure 5.9: The performance of Replacement Rules through LSR rule exploration by 05x05 mazes. The legend of "Alg0" with green colour is for Basic Replacement Rules for LSR (BscRepLSR). The legend of "Alg1" with red colour is for Extended Replacement Rules for LSR (ExtRepLSR). See text for a detailed explanation of the figure and the notation used.

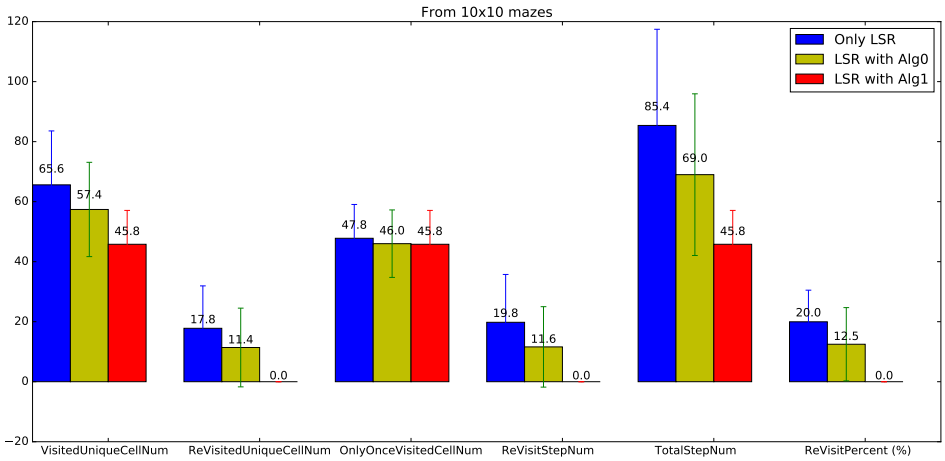


Figure 5.10: The performance of Replacement Rules through LSR rule exploration by 10x10 mazes. The legend of "Alg0" with green colour is for Basic Replacement Rules for LSR (BscRepLSR). The legend of "Alg1" with red colour is for Extended Replacement Rules for LSR (ExtRepLSR). See text for a detailed explanation of the figure and the notation used.

labels on individual bars are also illustrated. Blue bars show the results from LSR algorithm. It is named "Only LSR" in the figure legend. Green bars show the results after applying Venkata et al.'s Replacement Rules summarized in Table 4.1 (BscRepLSR). It is named "Alg0" in the figure legend. Red bars show the results from applying our proposed Replacement Rules (ExtRepLSR). This is named "Alg1" in the figure legend.

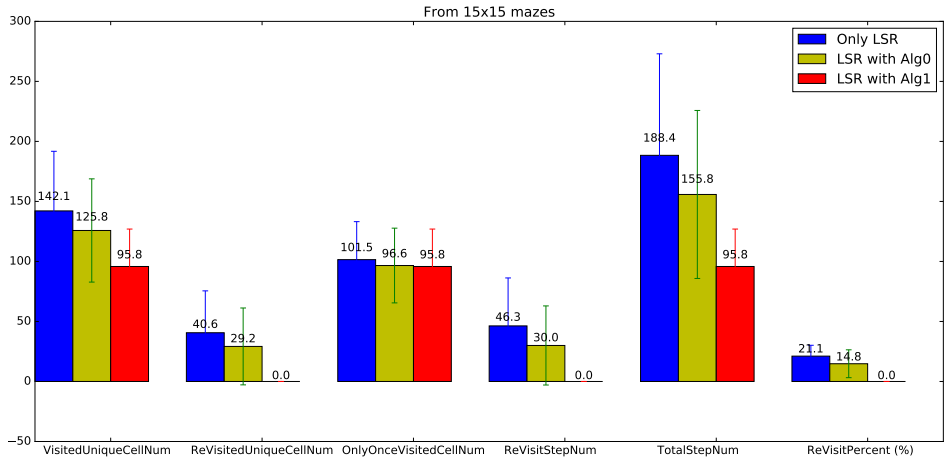


Figure 5.11: The performance of Replacement Rules through LSR rule exploration by 15x15 mazes. The legend of "Alg0" with green colour is for Basic Replacement Rules for LSR (BscRepLSR). The legend of "Alg1" with red colour is for Extended Replacement Rules for LSR (ExtRepLSR). See text for a detailed explanation of the figure and the notation used.

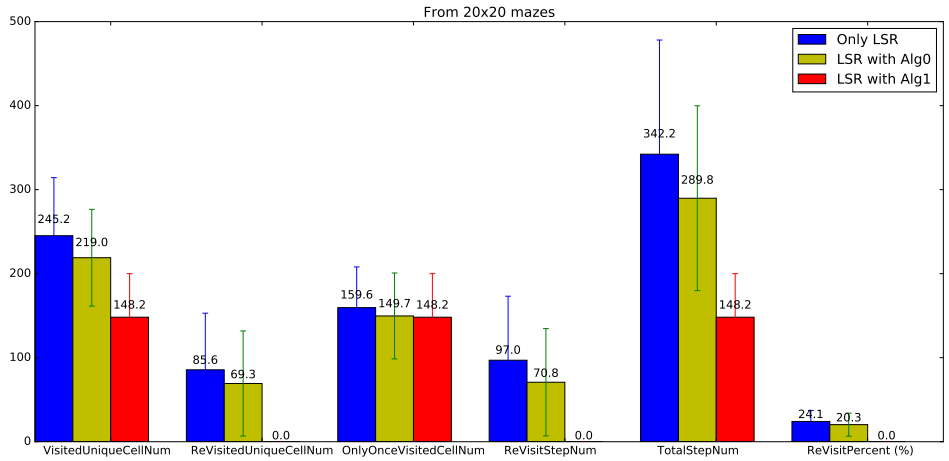


Figure 5.12: The performance of Replacement Rules through LSR rule exploration by 20x20 mazes. The legend of "Alg0" with green colour is for Basic Replacement Rules for LSR (BscRepLSR). The legend of "Alg1" with red colour is for Extended Replacement Rules for LSR (ExtRepLSR). See text for a detailed explanation of the figure and the notation used.

In each figures, six different parameters are demonstrated as :

the number of visited cells uniquely → VisitedUniqueCellNum,

the number of re-visited cells uniquely → ReVisitedUniqueCellNum,

the number of only once visited cells → OnlyOnceVisitedCellNum,

the number of re-visit steps → ReVisitStepNum,

the number of total steps → TotalStepNum,

the percent of re-visit steps → ReVisitPercent.

The difference between performance improvements can be seen more clearly once the size of maze increased. For instance, 5x5 mazes in Figure 5.9, BscRepLSR and ExtRepLSR have the same performance for all parameters. However, once the maze size is increased as in Figure 5.10, the difference between those two set of rules can be seen better. BscRepLSR become better than LSR algorithm clearly but it still cannot solve mazes without revisits unlike ExtRepLSR. The improvements become much more clear in 15x15 mazes in Figure 5.11 and 20x20 mazes in Figure 5.12. Moreover, revisit percent is quite reasonable indicator in order to show the performance of applied algorithms. Revisit percent in BscRepLSR once the size of maze increased also increases. However, this is always minimum in ExtRepLSR not only on small mazes also in bigger mazes.

Table 5.3 is filled based on experiment results in order to demonstrate and justify the performance of proposed rules. The performance improvement especially by looking bigger mazes can be analysed by looking into the number of steps during at least double or more times visited rooms, N_e. It becomes smaller once knowledge based rules are applied; in addition, it is minimum with our proposed rule list.

Algorithm →	LSR Only (LSR)			LSR with Venkata et al.'s rules (BscRepLSR)			LSR with Proposed rules (ExtRepLSR)		
Maze Sizes ↓	N_u	N_t	N_e	N_u	N_t	N_e	N_u	N_t	N_e
05x05	17.7	21.2	3.5	14.2	14.2	0.0	14.2	14.2	0.0
10x10	65.6	85.4	19.8	57.4	69.0	11.6	45.8	45.8	0.0
15x15	142.1	188.4	46.3	125.8	155.8	30.0	95.8	95.8	0.0
20x20	245.2	342.2	97.0	219.0	289.8	70.8	148.2	148.2	0.0

Table 5.3: Efficiency comparison of Replacement Rules through four different groups of maze sizes.

To sum up, we extend the list of Replacement Rules in order to find the best path even in bigger mazes. Furthermore, algorithmic approach is proposed in order to get more rules on different maze environments which have different dynamics. Comparisons through four different maze sizes show that the performance of learning with the proposed set of rules is

not decreased once the size of mazes increased unlike Venkata et al.'s Replacement Rules. On the other hand, extracted properties of the domain can also be used with RL algorithms in order to help to speed up the exploration phase. Therefore, the same rule set is applied into the exploration phase of Q-Learning in the following section.

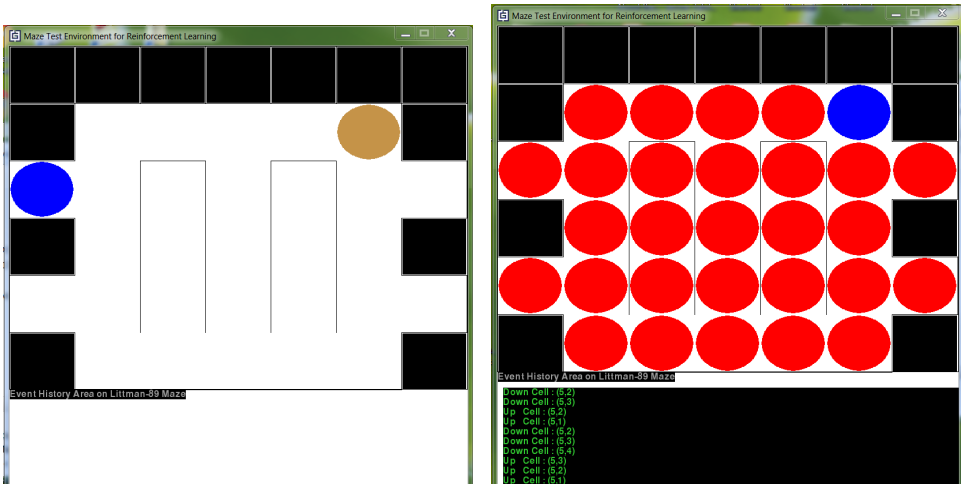
5.4 Replacement Rules with Temporal-Difference Learning

5.4.1 Applying to Temporal-Difference Learning

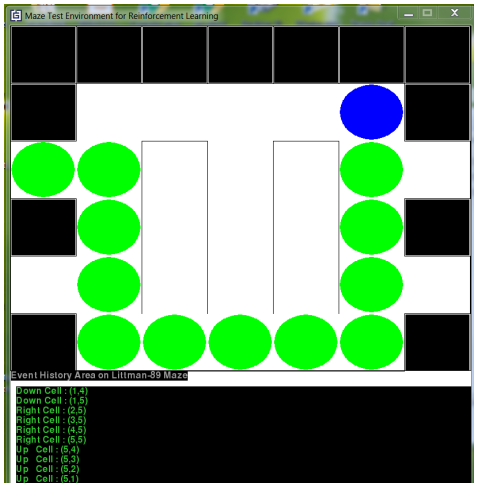
Here we combine Replacement Rules with the RL technique, Q-Learning, for the maze task. The agent initially explores the environment by Q-Learning instead of the wall follower algorithms in section 5.3. Here, Q-Learning is implemented with the greedy policy that next action is selected according to $a \leftarrow arg\max_{a} Q(s', a)$ described in section 4.9.3. After achieving the target point, the agent explores the task again until the pre-determined termination episode. Therefore, this section gives initial indications about performance improvements before fully combine Replacement Rules with Q-Learning in section 5.4.2.

In Figure 5.13, all Replacement Rules which are extracted from LSR and RSL (ExtRepLSR and ExtRepRSL) are applied along with Q-Learning. Here, the number of exploration stages are determined as 5, the agent explores the task using Q-Learning through 5 random walks. In the final session (5^{th}), Replacement Rules are applied. If we apply the Replacement Rules at the second stage (similar to ExtRepLSR and ExtRepRSL), we cannot benefit from Q-Learning updates because in the first run only one neighbouring cell is updated, and the second run only two neighbouring cells are updated. This issue can be improved by taking into account TD(λ). This case is also analysed in section 5.4.2; however, here main focus is the improvement affect of Replacement Rules through Temporal-Difference Learning.

On the other hand, the reason to select 4 is the shortest distance between source and target rooms has 5 rooms. However, as seen in Figure 5.13c, although it does not branch any dead-ends, it cannot find the shortest path. This might be related with the number of Q-Learning sessions executed before Replacement Rules are applied. We expect to achieve the shortest path, if we apply Replacement Rules after more than 5 episodes. Also, the task in Figure 5.13 cannot be solved with wall follower algorithms, if the target is selected in deadlock areas. However, this is not the case in Q-Learning explorations.



(a) At the start of exploration. (b) 5^{th} episode with only Q-Learning.



(c) 5^{th} episode with Q-Learning and Replacement Rules

Figure 5.13: Replacement Rules with Q-Learning. Red circles represent more than one time visited rooms, green circles illustrate only one time visited rooms, blue circle demonstrates current room and finally brown circle shows the target point (a). (b) This is captured from 5^{th} episode applied only Q-Learning. (c) This is captured from 5^{th} episode applied Q-Learning and Replacement Rules. First 4 episodes are performed with Q-Learning and the final episode is executed with Replacement Rules by tuning the experience taken from first 4 episodes with Q-Learning.

5.4.2 Comparison of Proposed Algorithms

The TD(0) technique does not take into account past states. The only affected states in TD(0) are the current state s_t and the following state s_{t+1}. To speed up of the learning convergence, gained information at $t+1$ can also be extended to past states. This objective is implemented with a short-term memory mechanism to show the degree to which it has been visited in the

recent past activities.

The agent starts out knowing nothing; the Q table is initialized to zero. Each episode, is equivalent to one training session, consists of the agent moving from the starting state to the target state. Each time the agent arrives at the goal state, the program goes to the following episode. The actions that lead immediately to the goal have an instant reward of 100. Other actions not directly connected to the target have -1 reward. The parameters used in Q-learning and Replacement based Q-Learning experiments are the same: $\gamma = 0.9$ the exploitation/exploration rate, $\alpha = 0.1$ learning rate, $\lambda = 0.9$ the discount factor. There is an additional parameter of eligibility decay $\tau = 0.7$ for the case of eligibility trace. Performances are averaged over 10 randomly generated 5x5 maze environments through 100 episodes.

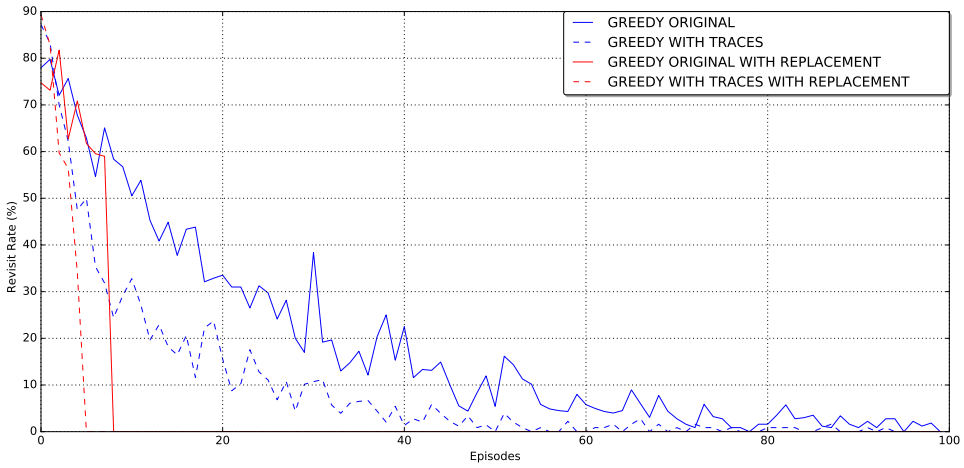


Figure 5.14: The comparison of revisit rates. The performance of four algorithms described in the text. See text for a detailed explanation of the figure and the notation used.

Figure 5.14 and Figure 5.15 compare the performance of four different cases in terms of revisit rates and total discounted rewards, respectively. Greedy original: Q-Learning or in other words TD(0). Greedy original refers that next action is selected according to the state-action with the maximum Q-value from available state-action pairs described in section 4.9.3. Greedy with traces: the the version of greedy original by adding eligibility trace. Greedy original with replacement: We apply proposed list of Replacement Rules (BscRepLSR and BscRepRSL) into greedy original. Greedy with traces with replacement: We apply proposed list of Replacement Rules (BscRepLSR and BscRepRSL) into greedy with traces. All of listed methods are proven to converge illustrated in Figure 5.14. Although the agent in each algorithm begins with

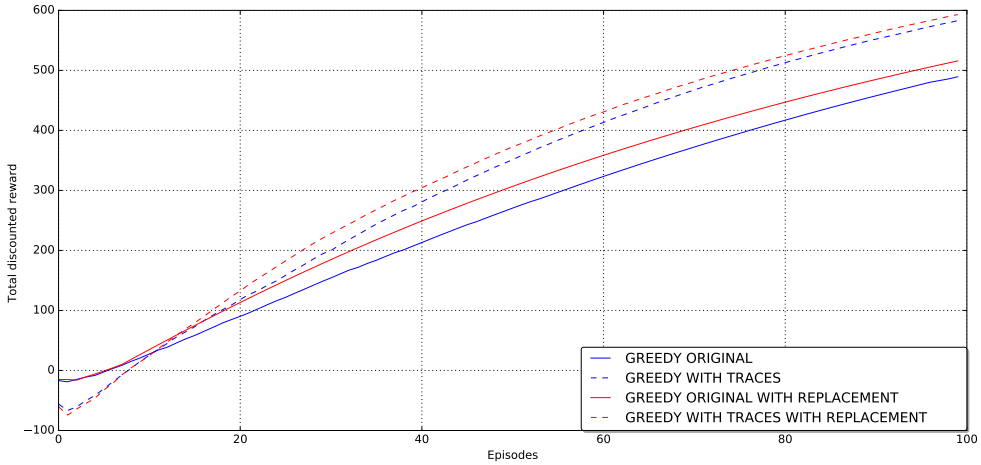


Figure 5.15: The comparison of total taken discounted reward. See text for a detailed explanation of the figure and the notation used.

intense exploration to find a route towards the goal, eligibility traces help to speed learning compared to non-eligibilities.

Figure 5.15 demonstrates total discounted reward during each episode for each cases. As expected from Figure 5.14, better convergence cases get better reward collection. Although the case of greedy with traces with replacement starts from minimum reward total, it becomes maximum after roughly 15 episodes.

5.5 Summary

In this chapter, the performance of TD learning for the maze task is successfully improved by producing new set of extraction rules. We introduce a set of novel Replacement Rules as ExtRepLSR and ExtRepRSL in section 5.3 for maze environments with a remarkable performance compared to LSR rule or RSL rule itself, also to previously offered set of rules from Venkata et al. (2011) (BscRepLSR). The set of extracted rules is also applied with TD(0) and TD(λ) techniques. We demonstrate that Replacement Rules of ExtRepLSR and ExtRepRSL have important improvements on both techniques as well. Although Replacement Rules are combined with TD-Learning here, it can also be extendible to any maze solving algorithm or maze navigation problems in order to enhance the learning speed using proposed algorithmic generation in section 5.3.1.

The simulation platform for a maze task is initially developed for wall follower algorithms (LSR and RSL) and Q-Learning algorithm. Then those algorithms are combined with previously offered list of rules (Venkata et al., 2011) BscRepLSR and currently proposed Replacement Rules (ExtRepLSR and ExtRepRSL) to get better learning performance in section 5.3. This provides a conceptual simulation framework for further path navigation models. Also, we experimentally investigate the underlying mechanism of eligibility traces and TD learning used later on.

In the thesis, we aim to apply Reinforcement Learning techniques into Spiking Neural Networks. Therefore, one of motivation for exploring implementation challenges of RL components throughout the experiments in this chapter is to prepare for learning algorithms using RL components in following chapters. The work in this chapter gives an overview and experimental stage about RL side. For instance, the performance advantage of eligibility traces is examined by combining basic Temporal-Difference learning with eligibility traces. Also, the mechanism of discounted reward with different policy types is used here. To sum up, the conceptual mechanisms of Reinforcement Learning side through experimental tasks in this chapter will be the basis for further experiments use the modulation of the reward for synaptic strengths detailed in chapter 9.

Chapter 6

Spiking Neural Network: Background

Contents

6.1 Introduction . . . 139
6.2 Spiking Neural Network Topologies . . . 140
6.2.1 Feed-forward Architectures . . . 141
6.2.2 Actor-Critic Networks . . . 142
6.2.3 Recurrent Networks . . . 143
6.3 Spike Generation . . . 143
6.3.1 Spike Train . . . 144
6.3.2 Poisson Processes . . . 144
6.3.3 Homogeneous Poisson Processes . . . 144
6.4 Neural Coding . . . 145
6.4.1 Rate Coding . . . 145
6.4.2 Population Coding . . . 146
6.4.3 Temporal Coding . . . 147
6.4.4 Summary . . . 148
6.5 Measures of Spike Train Difference and Spike Train Synchrony . . . 149
6.5.1 Victor & Purpura Distance (VP) . . . 149
6.5.2 Coincidence Factor (CF) . . . 150
6.5.3 Schreiber Distance (ScD) . . . 150
6.5.4 van Rossum Distance (vRD) . . . 151
6.5.5 Discussion . . . 151
6.6 Summary . . . 152

6.1 Introduction

Previous generations of Neural Networks use analogue signals to carry information between neurons. However, Spiking Neural Networks (SNNs) use spikes similar to biological neurons. SNNs are often referred to as the third generation of Neural Networks, which have potential

to handle complex tasks with the accurate modelling of natural computing in the brain (Bohte et al., 2002b, a). Therefore, SNNs are biologically plausible (Bohte et al., 2002b; Gruning & Sporea, 2012; Kasabov, 2012) and offer some basis to represent time, frequency, phase and other features of the information being processed.

After introducing the elements of Neural Networks, this chapter describes common topologies of Neural Networks in section 6.2. One of the most common network structures is the feed-forward topology, which is discussed in detail in subsection 6.2.1. Then, the structure of actor-critic networks is summarized in subsection 6.2.2. Also, more complex structures than the feed-forward topology, i.e. recurrent networks, are described in subsection 6.2.3. The details of spike generation is discussed in section 6.3. Initially, we introduce the description and notation for spike trains in subsection 6.3.1. Then, we introduce Poisson process, especially Homogeneous Poisson process, in subsection 6.3.3.

One of the fundamental topics of neuroinformatics is to analyse the way neurons encode information through spike trains. So various information encoding schemes such as rate coding in subsection 6.4.1, population coding in subsection 6.4.2, and temporal coding in subsection 6.4.3 are reviewed in section 6.4. We also need to measure the performance of Spiking Neural Networks through generated spike trains. There are various metrics in order to measure the (dis)similarity between spike trains such as Victor & Purpura Distance described in subsection 6.5.1, Coincidence Factor described in subsection 6.5.2, Schreiber Distance described in subsection 6.5.3 and van Rossum Distance (vRD) described in subsection 6.5.4.

6.2 Spiking Neural Network Topologies

Two or more neurons can be combined in a layer and a particular network can contain one or more such layers. One of the most commonly used Neural Network structures can be considered with three layers of neurons as input layer, computational layer and output layer as illustrated in Figure 6.1. Each layer consists of one or more **nodes** which represents the neurons shown in the diagram by the circles. Spiking neurons in a Neural Network can be structured in various different ways in terms of neuron connectivity. Broadly, there are two main categories: feed-forward structure where signals flow in single direction only and recurrent networks where neural signals flow in both directions.

6.2.1 Feed-forward Architectures

Networks without backward connections are called Feed-forward Networks (FNNs) because of the clear direction in signal propagation. In this type of Neural Network, the information flows only forwards from input to output (Kriesel, 2014). There is not any feedback signal from output to input or hidden layers. From one node to the next, arrows indicate the flow of information over the network in Figure 6.1 .

More generally, a Feed-forward Network structure consists of one input layer, n hidden computational layers and finally one output layer. A typical example of a Feed-forward Network mechanism with a single hidden layer is illustrated in Figure 6.1. Neurons in the input layer collect stimuli from the actual world similar to the sensory neurons in humans which are responsible for converting external stimuli from the environment into corresponding internal stimuli. The computational layer is called the hidden layer. Finally, the output layer prepares the response of the network similar to the motor neurons in humans. The output of the entire network itself is just the output from the designated output neurons.

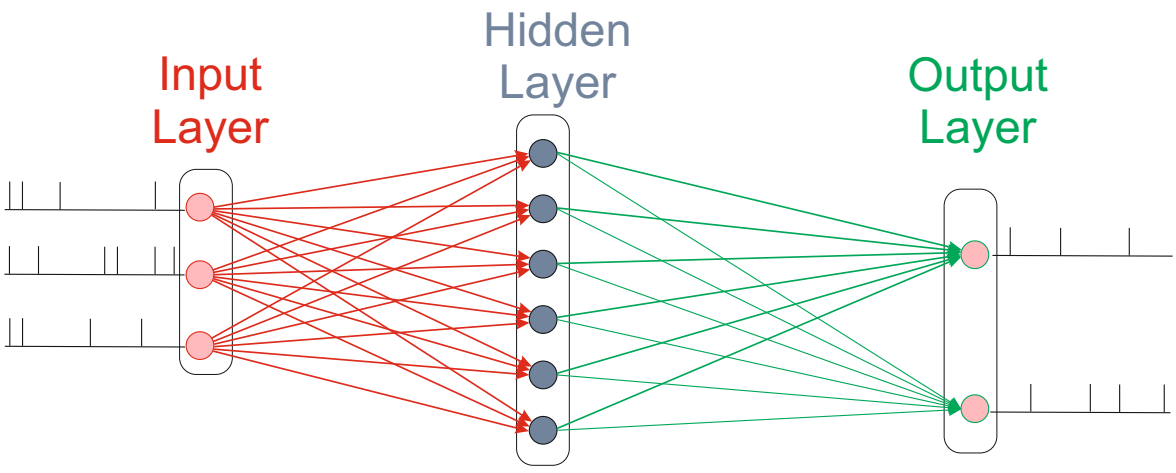


Figure 6.1: Illustration of a typical feed-forward SNN structure. Input layer neurons (three neurons-red circled) are fully connected to neurons in the hidden layer (six neurons). Hidden layer (drawn gray) neurons process the receiving neural signals to propagate neurons in the output layer (two neurons). Each node/neuron is modelled as a spiking neuron in SNNs as described in section 2.4.

6.2.2 Actor-Critic Networks

An actor-critic architecture is an instantiation of the Temporal-Difference (TD) learning algorithm (detailed in section 4.9.1) (Witten, 1977; Sutton & Barto, 1998). It contains two different units: actor and critic. Current synaptic weights cause actions through the output neuron similar to the actor unit. The critique is a TD error between the actual and desired spike patterns. The scalar signal derived from those two signals as the output of the critic module drives all learning. This parameter determines the reward in order to modulate synaptic plasticity.

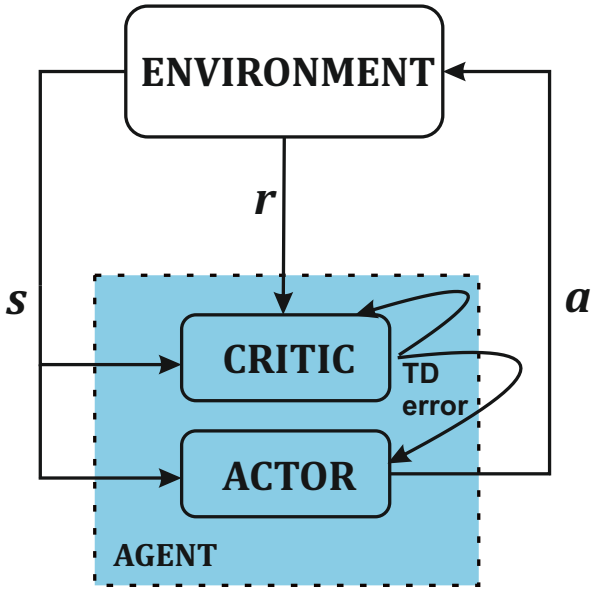


Figure 6.2: Illustration of a typical actor-critic architecture. Once an action a is performed from the selection of actor unit, a new state is entered. The new state information s is transmitted actor and critic units. Also the reward r associated with the new state is transmitted to the critic unit. The critic module generates an error signal (TD error) based on the disparity between new state and expected state. Figure is adapted from Sutton & Barto (1998). See text for a detailed explanation of the figure and the notation used.

The two main units of this architecture are called actor and critic in Figure 6.2. The actor selects an action a based on the policy in each state s. As a result of the performed action, a new state is entered. The environment informs the critic and the actor units about the current state s. Also, the current reward information r is transmitted to the critic unit associated with the state. The critic unit calculates an error signal, TD error, based on the evaluation of the new state's consequences are better/worse than expected.

Actor-critic networks for hippocampal place cells using population coding (see section 6.4.2 for population coding) are discussed in Senn & Pfister (2014). Similar to the non-spiking

actor-critic architecture, the form of actor-critic architecture in Spiking Neural Networks is demonstrated in Figure 7.4 inspired from Potjans et al. (2009); Fremaux et al. (2013). The details of proposed structure using actor-critic architecture is extended in section 7.5.

6.2.3 Recurrent Networks

Recurrent Neural Networks (RNNs) are more complex structures of NNs compared with feed-forward topologies. In addition to forward signal propagation to subsequent layers, they have also recurrent connections which are feedback channels from the following layer(s) to the previous layer(s) or connections between neurons of the same layer. Various other types of NN architectures can be reviewed by (Lipton et al., 2015; Mulder et al., 2015; Diehl et al., 2016).

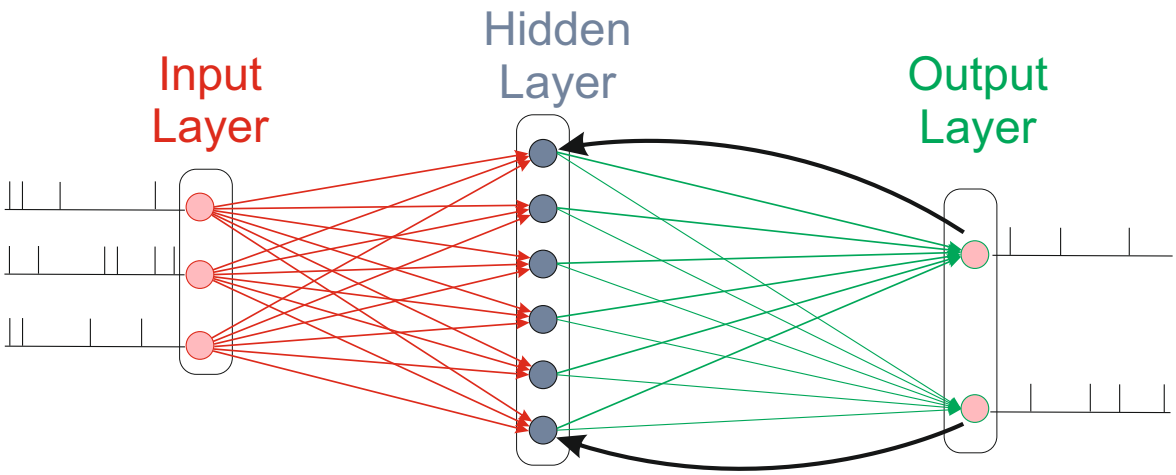


Figure 6.3: Illustration of a recurrent SNN structure. There is not any restrictions in terms of flow directions unlike feed-forward mechanisms. For instance, there are 2 recurrent connections from output layer to hidden layer. There could be more recurrent connections like feeding back the activity of the hidden-neurons to themselves in order to serve as a memory of previous spiking activities. Each node/neuron is modelled as a spiking neuron in SNNs as described in section 2.4. The figure is drawn by the inspiration of Lipton et al. (2015).

6.3 Spike Generation

This section describes spike train notation and generation of artificial spike trains for input neurons and output neurons. Input spike patterns are applied to input neurons. Output ones are generated by output neurons in SNN.

6.3.1 Spike Train

The activity of neuron j, n_j, is described by a sequence of discrete events at which the neuron emits a spike,

$$t_j = \{t_j^{(f1)}, t_j^{(f2)}, t_j^{(f3)}, ..., t_j^{(fN_j)}\} \tag{6.1}$$

where all $t_j^{(fn)} > 0$, $t_j^{(fn)} < t_j^{(f(n+1))}$, N_j for neuron j is the total number of spikes and t_j^{fn} is the time of the n^{th} spike of neuron j. This representation of a spike sequence is referred to as a spike train. They are impulses that can be written as a sum of Dirac Delta form (see section A.1),

$$S_j(t) = \sum_{n=1}^{N_j} \delta(t - t_j^{(fn)}) \tag{6.2}$$

Based on the independence of spike train generation, the spike train can be described using a particular kind of random process. This process is often described as a Poisson process to reproduce the presented experiments (Gerstner et al., 1996; Kempter et al., 1999; Song et al., 2000; Gutig et al., 2003).

6.3.2 Poisson Processes

Assuming that the timing of successive action potentials depends only on an underlying continuous driving signal, $r(t)$ as the instantaneous firing rate. Therefore, the generation of spikes is independent from all the others and this is called the independent spike hypothesis. The spike train can be described as a particular kind of random process called a Poisson process with two types: the homogeneous and inhomogeneous Poisson process. We use homogeneous Poisson processes detailed in 6.3.3.

6.3.3 Homogeneous Poisson Processes

In our experiments, the generation of sets of spike trains for inputs/outputs is controlled by the distribution of Poisson process. The homogeneous Poisson process, the average neural firing rate $r(spikes/sec)$ is assumed as a constant, defined as:

$$r(T) = r = N_{sp}/T \tag{6.3}$$

where N_{sp} is the total number of spikes that occur within the full time length of T. Although $r(T)$ varies over time in an inhomogeneous Poisson processes, it is not used here. Assuming a long time interval $(0, T)$ and picking a sub interval (t_1, t_2) of length $\Delta t = t_2 - t_1$, the density or probability of having k spikes during the time interval Δt can be derived from the binomial formula (Dayan & Abbott, 2005) as:

$$P\{N(\Delta t) = n\} = e^{-r\Delta t}\frac{(r\Delta t)^n}{n!} \quad (k \geq 0). \tag{6.4}$$

where $N(\Delta t)$ is the number of spikes in a finite time interval of length Δt, r is the mean firing rate as the average number of spikes per second in a given spike train. This formula is called the Poisson probability density function.

6.4 Neural Coding

One of the fundamental questions in neurophysiology is how neurons encode information between sensory inputs and motor actions through their spike trains. Neural coding is a transformation from physical space to neural space. Although neural information can be encoded in diverse ways, there are three main encoding schemes in SNNs: rate coding, population coding, and temporal coding. In rate coding, the number of spikes within an encoding window is considered regardless of their temporal pattern, while for temporal coding the precise timings of spikes are considered to describe a stimulus (Panzeri et al., 2010). In population coding, the information is encoded in the activity of neuron populations. From a biological perspective, precise spike timing of individual spikes is common for neuronal information processing (Bohte, 2004; Thorpe et al., 2001).

In our discussion of neural coding, s refers to sensory stimuli and r refers to neural activity. The mapping of sensory information into spike trains is a stochastic transformation from s to r. This is the job of the brain. This section briefly reviews each of the aforementioned coding schemes.

6.4.1 Rate Coding

Rate-based coding models use either counts of the total number of spikes as neuronal firing rate or measures of the Inter-spike Interval (ISI). The idea is established in experiments

performed by Adrian (1926) where the firing rate of neurons in the muscle depends on the applied force. This mechanism is improved by further physiological observations that neurons tend to fire more for stronger stimuli (Gawne et al., 1996). It does not need precise timing of each spike to convey information. Rate encoding is also called frequency encoding and can been used to interpret the outputs of Spiking Neural Networks.

Assume that the firing rate of a neuron r represents a stimulus value s:

$$r = f(s) \tag{6.5}$$

where r and s are constant over a short time Δ. The function f is known as the tuning curve with some common forms of Gaussian, Cosine, Wrapped Gaussian and circular Gaussian. The number of spikes N produced within the time interval Δ is around $r\Delta$ according to the Poisson distribution described in section 6.3.

6.4.2 Population Coding

In population coding, information is conveyed by the distributed and coordinated activity of different pools of neurons treated as populations. So the information accuracy is determined not only by the individual firing activities, but also by the correlations between neurons (Pouget et al., 2000; Trappenberg, 2010).

Neural responses are denoted as a vector r which describes the numbers of spikes of the different neurons. Encoding of the task as a relationship between the stimulus and the response of neuron populations is expressed by the conditional probability distribution:

$$p(r|s) = p(r_1^s, r_2^s, r_3^s, ...|s) \tag{6.6}$$

where s is a certain stimulus and $r_1^s, r_2^s, r_3^s, ...$ stand for the stimulus-specific response in the population (Trappenberg, 2010). Therefore, the statistical dependency between population activities and neurons' stimuli is related with statistical parameter estimation which is in the field of information theory.

6.4.3 Temporal Coding

Experimental studies show that neural systems use the precise time of individual spiking activities, and not just their firing rates, to encode information. Evidence for the importance of the first spike timing is observed in the human visual system (Thorpe et al., 1996). As an idealization of temporal coding, the timing of the first spike after the reference signal contains the entire information about the incoming signal in this scheme named time-to-first-spike coding. Temporal coding models use the timing of the individual spikes to convey information. Additionally, recent studies demonstrate that the temporal coding paradigm offers significant computational benefits compared with the rate coding (Kempter et al., 1999).

Spatio-temporal spike sequences as a form of temporal encoding are localized in space and in time (see Figure 6.4). The spatial characteristics of the input stimuli would depend on the organization and order of input neurons. Temporal properties are conveyed through the group of spike times for each individual input neuron.

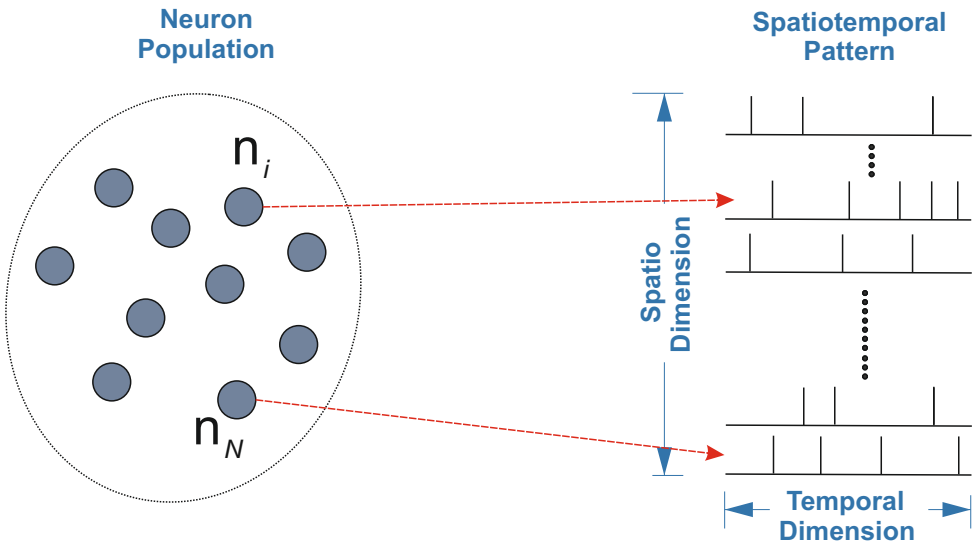


Figure 6.4: A typical spatiotemporal spike pattern. A group of neurons transfers the encoded information. Each neuron's spike train in time (vertical lines) represents the temporal dimension of information. All spike trains from the neuron group (horizontal) represent spatial dimension.

According to the number of spikes, temporal coding schemes can be divided into two forms: single-spike and multi-spike learning (spike sequence learning). Spike coding which takes into account only a single-spike can be classified under single-spike schemes such as time-to-first-spike coding (Thorpe, 1990). This coding scheme assumes the major portion of neural

information in biology is carried with the first spikes; therefore, neurons perform with only one spike per neuron in this scheme. This reduction technique considering the time to the first spike ignores the spatial extent of the neurons. However, the encoding scheme with multiple spikes take into account all precise spike timings (Legenstein et al., 2005; Pfister et al., 2006; Ponulak, 2008; Ponulak & Kasinski, 2010).

A range of spike codings for learning systems using SNNs are important, not just time-to-first-spike, so the ability to learn multiple spike sequences is relevant for a range of real-world applications described in Esser et al. (2013) using an advanced SNN neuromorphic platform of IBM TrueNorth. Despite the capability of single-spike coding compared to rate coding, it limits the diversity and capacity of the transmitted information through SNNs. However, multi-spike coding can carry more variation of information. Hence, it can significantly extend the richness of the information representation in SNNs (Ponulak & Kasinski, 2010). Furthermore, multi-spike coding is more biologically realistic compared to single-spike coding (Xu et al., 2013). However, the difficulty of learning in multi-spike coding is increased computationally compared to single-spike coding due to the interference of the local learning processes (Ponulak & Kasinski, 2010).

6.4.4 Summary

This section outlines different neuronal coding schemes used to represent sensory information. Although there is growing evidence that the brain uses all these encoding types or the combination of them, precise timing of spikes allows neurons to carry more information than random spike timings with the paradigm of rate-coding schemes. Therefore, temporal synchrony might play an important role in neural computation especially with the importance of fast information processing. For the optimal efficiency of information computation, temporal synchrony paradigm is used throughout chapter 8 and chapter 9 using a simple Poisson process for generating input and output spike patterns.

In order to transform information from the actual world into spike trains during our simulations, the temporal synchrony paradigm is implemented using a simple Poisson process for generating input and output spike patterns. Because of previously mentioned reasons as biological plausibility, the diversity and capacity of information transmission, we focus on encoding

inputs and outputs using multi-spike coding rather than single-spike coding through our experiments. In the following section, we describe different metrics for spike train differences.

6.5 Measures of Spike Train Difference and Spike Train Synchrony

Measuring the level of synchrony between two or more spike trains are important for many tasks in order to measure the deviation between actual and desired outputs during learning, or to measure the performance of the proposed method. Let S_1 and S_2 be two spike trains, each of them consists of a sequence of spike times using Equation 6.1 and Equation 6.2 as follows:

$$\begin{aligned} S_1(t) &= \sum_{n=1}^{N_1} \delta(t - t_1^{(fn)}) \\ S_2(t) &= \sum_{n=1}^{N_2} \delta(t - t_2^{(fn)}) \end{aligned} \tag{6.7}$$

where $t_1^{(fn)}$ and $t_2^{(fn)}$ denote the time of the n^{th} spike in the spike train of indicated neuron 1 and 2, respectively. N_1 and N_2 are he total number of spikes in S_1 and S_2, respectively.

There is no unique way to measure the dissimilarity between S_1 and S_2. Therefore, several different approaches for the performance statistics are considered below.

6.5.1 Victor & Purpura Distance (VP)

The Victor & Purpura Distance metric D_{VP} introduced in (Victor & Purpura, 1996, 1997) defines the dissimilarity between two spike trains as the minimum cost of transforming one spike train into the other. The transformation combines three operations: *a)* spike insertion, *b)* spike deletion, and *c)* spike shifting in time. The cost of both deleting and inserting a spike is fixed to 1, whereas the cost of shifting a spike in time $q\delta t$ is proportional to the distance moved δt where $\delta t = |t^1 - t^2|$, t^1 and t^2 are time in actual spike train S_1 and reference spike train S_2, respectively. The cost of the distance is determined by the parameter q, the cost per distance of moving a spike. For instance, if q increases, the metric becomes increasingly sensitive to spike time. Also the distance between two spike trains with N_1 and N_2 spikes is no greater than $N_1 + N_2$.

If and only if the spike trains S_1 and S_2 are identical, the distance metric $D_{VP} = 0$.

The distance calculation can be done as:

$$G_{i,j} = \begin{cases} c_{in}j, & \text{if } j = 0 \\ c_{in}i, & \text{if } i = 0 \\ min\{G_{i-1,j-1} + q\,|t_{2i} - S_{1j}|\,, G_{i-1,j} + c_{in}, G_{i,j-1} + c_{in}\} & \text{if } i > 0 \text{ and } j > 0 \end{cases} \tag{6.8}$$

where c_{in} is the spike insertion/removal cost, $G_{i,j}$ is the distance between the truncated spike trains formed by the first i spikes of S_1 and the first j spikes of S_2. Three possibilities inside the case of $i > 0$ and $j > 0$ corresponding to (a), (b) and (c) described above. G_{N_1,N_2} yields $D_{VP}(S_1, S_2)_q$ as the minimum cost of a transformation from S_1 to S_2.

6.5.2 Coincidence Factor (CF)

The Coincidence Factor (CF) as described by Kistler et al. (1997) between two spike trains determines the rate of similarity/dissimilarity between spike trains. The bound limits of this correlation measure are [-1, 1] where 1 is exactly the same, 0 is not correlated, and -1 is perfectly anti-correlated. The Coincidence Factor Γ is formulated as:

$$\Gamma = \frac{N_{coinc} - E\,(N_{coinc})}{\frac{1}{2}\,(N_{des} + N_{act}) - E\,(N_{coinc})} \tag{6.9}$$

where N_{des} are the number of spikes in the desired train as a reference, N_{act} is the number of spikes in the actual output as a comparing train, N_{coinc} is the number of coincident spikes within a time window $\Delta, E\,(N_{coinc}) = 2r_{out}\Delta N_{ref}$ is the expected number of coincident spikes generated by a homogeneous Poisson process with its rate r_{out}.

6.5.3 Schreiber Distance (ScD)

This correlation-based measure is first proposed by Haas & White (2002), and detailed by Schreiber et al. (2003). The two point processes S_1 and S_2 are convolved with a kernel filter, resulting in time series $S_1'(t)$ and $S_2'(t)$. Then the pairwise correlation between them is formulated:

$$S_S = \frac{\int_t S_1'(t) S_2'(t) dt}{\sqrt{\int_t S_1'^2(t)}\sqrt{\int_t S_2'^2(t)}} \tag{6.10}$$

The kernel filter can be an exponential or Gaussian kernel with the fixed width τ_S which defines the time scale of interaction between the two spike trains. If spike trains are identical, $S_S = 1$.

6.5.4 van Rossum Distance (vRD)

One of the important metrics in order to compute dissimilarity between spike trains is the van Rossum Distance (vRD) method which is a dimensionless distance (van Rossum, 2001). It takes into account additional and missing spikes. The distance measure maps the two spike trains as an actual spike train with S_a and a desired spike train S_d onto a profile $S_a, S_d \rightarrow R(t)$ with $0 \leq R(t) \leq 1$. The overall spike distance value can be calculated by integration as $D_R = \int R(t)dt$.

In this spike distance, the discrete spike trains S_a and S_d (see Equation 6.2) are transformed into a continuous function:

$$f(t) = e^{-t/\tau_R} H(t) \tag{6.11}$$

where τ_R is the time decay constant and $f(t)$ is the convolution of each spike t_i with an exponential kernel with the Heaviside step function as $H(t)$ (described in section A.4) using a discrete convolution as:

$$(f * S)(t) = \sum_{s:0 \leq t-s < T_R} f(s)S(t-s), \quad 0 < t < T_R \tag{6.12}$$

where T_R is the duration of spike trains. Then the van Rossum Distance D_R between S_a and S_d can be calculated as:

$$D_R(S_a, S_d) = \sum_{0 \leq t < T_R} \Big((f * S_a)(t) - (f * S_d)(t) \Big)^2 \tag{6.13}$$

where τ_R is the time decay constant and D_R is the overall value of van Rossum Distance between two spike trains of S_a and S_d.

6.5.5 Discussion

It is not a straightforward task to compare spike trains, particularly for stochastic spike trains where there is variability. Relying on counting the spike number within a spike train is the most basic approach to calculate the similarity or dissimilarity of spike trains. However, the temporal structure of spike trains is ignored here such as the Coincidence Factor described in subsection 6.5.2. Schreiber Distance as a correlation-based measure described in subsection 6.5.3 uses Gaussian filter. In Victor & Purpura Distance, described in subsection 6.5.1,

unlike van Rossum Distance, it is difficult to determine which spike is missed or extra activity if the number of spikes in the desired and actual spike trains is unequal (van Rossum, 2001). Moreover, in Schreiber Distance and Victor & Purpura Distance, required computation time grows more than linearly with the number of spikes. However, this is not the case in van Rossum Distance described in subsection 6.5.4 (Rusu & Florian, 2014). Therefore, van Rossum Distance is used throughout the experiments it takes into account the temporal encoding and computational efficiency.

6.6 Summary

This chapter describes common topologies of Neural Networks in Spiking Neural Networks: the Feed-forward Network topology, actor-critic structure, and Recurrent Neural Network. Then, we describe artificial spike generation based on Poisson process with the spike notation used throughout the thesis. Also, we discuss the encoding mechanisms: rate coding, temporal coding and population coding. Finally, in order to measure difference between spike trains, spike distance metrics are discussed: Victor & Purpura Distance, Coincidence Factor, Schreiber Distance, and van Rossum Distance.

In our further experiments, actor-critic architecture is used as it is detailed in section 7.5. Furthermore, the advantages of temporal over rate-based schemes are discussed in section 6.4.4. Therefore, we use spatio-temporal encoding mechanism for our experiments detailed in section 7.6. Temporal code schemes use multiple input and output spikes for all tasks in order to realise the maximum potential of proposed learning mechanisms. From distance metrics discussed in section 6.5, we use Coincidence Factor and especially van Rossum Distance through the following chapters.

Chapter 7

Spiking Neural Network: Implementation

Contents

7.1 Introduction . . . 154
7.2 Neuron Models . . . 154
7.3 Noise . . . 155
7.4 Delay Mechanisms . . . 158
7.5 Network Architectures . . . 160
7.5.1 Training . . . 160
7.5.2 Testing . . . 162
7.5.3 Bias Neuron . . . 162
7.6 Neural Coding . . . 164
7.6.1 Spike Generation . . . 164
7.6.2 Encoding . . . 165
7.7 Error Analysis . . . 168
7.7.1 Measures of Spike Train Difference . . . 168
7.7.2 Network Classification . . . 169
7.8 Training and Testing Mechanisms . . . 171
7.8.1 Training Mechanism . . . 171
7.8.2 Testing Mechanism . . . 173
7.9 Benchmark Descriptions . . . 174
7.9.1 Mapping . . . 174
7.9.2 Logical Operations . . . 175
7.10 Summary . . . 175

7.1 Introduction

This chapter describes the common mechanisms and techniques during experiment setups used in chapter 8 and chapter 9. The types of neuron models used in proposed SNNs under the noiseless conditions are summarized in section 7.2. In addition to training under noiseless conditions, the scenario in which noise is present is also considered in the experiments detailed in section 7.3. Then the mechanism of multiple delay connections between input and output neurons is described in section 7.4. Followingly, the network architectures of spiking neurons for obtaining spatio-temporal experiments during training and testing are illustrated in general.

In section 7.6, the implementation of the spike-generator-based Poisson process described earlier in section 6.3.2 and section 6.3 is explained. Also, temporal encoding mechanism as a neural coding is detailed here. Then, the adoption of van Rossum Distance (vRD) in order to measure spike-train similarity and a (mis)classification error metric in order to evaluate task performance of the network are detailed in section 7.7. Also, the pseudocodes of training and testing mechanisms throughout the experiments in chapter 8 and chapter 9 are described in section 7.8. Finally, benchmarks used during experiments are introduced in section 7.9.

7.2 Neuron Models

Although there are several kinds of spiking neuron models such as Leaky-Integrate-and-Fire, Hodgkin-Huxley model, Spike Response Model, and Izhikevich model described in section 2.5, the Leaky-Integrate-and-Fire model is used for the sake of simplicity. We use two sets of parameters for the LIF neurons.

The first set of LIF neuron parameters summarized in Table 2.4 are used here. These types of neurons are used for active readout nodes responsible for the learning of network. This learning neuron is driven by synaptic currents generated by its synaptic afferents. Similar to the summation of all incoming currents described in Equation 2.20, the membrane voltage V_j is a weighted sum of Post-synaptic Potentials (PSPs) of all incoming spikes as:

$$\Delta V_j(t) = \sum_i \sum_k^K \sum_f w_{ij,k} \kappa(t - t_i^f - d_{ij,k}) + V_{rest} \tag{7.1}$$

where ΔV_j is the total additional affect from all afferent neurons to the LIF neuron j (see subsection 2.5.4), w_{ij}^k is the synaptic efficacy (weight) of terminal k between neuron i, n_i, and neuron j, n_j. The f^{th} firing time of the i^{th} afferent neuron is denoted by t_i^f. $d_{ij,k}$ is the delay associated with synaptic terminal k between n_i and n_j. The function $\kappa(t)$ describes the form of post-synaptic response with $\kappa(t) = 0$ for $t < 0$. For $\kappa(t)$, we use the Dirac Delta function δ described in section A.1 in order to define PSPs. V_{rest} is the rest potential of the neuron. Each afferent spike causes a change in the Post-synaptic Potential. The amplitude of the PSP is modulated by the synaptic efficacy w_{ij}. At each time step along with the simulation time, these PSP values are added to form a membrane potential at the output neuron. Once this sum exceeds a predefined threshold potential V_{th}, the output neuron emits a spike.

In addition, another parameter set of LIF neurons transfers the input from pre-synaptic to post-synaptic spikes. They are named dummy LIF neurons. Whenever those neurons have any incoming spikes, they directly emit output spikes without integration. This is achieved by increasing V_m by $V_{th} - V_{reset}$ when a spike is received.

In the following experiments, the output layer of proposed networks has at least one active readout neuron which actively learns the desired spike pattern. On the other hand, dummy LIF neurons are used in the input layer. There is also a passive readout neuron which mimics the desired output spike train without any learning. If there is any different setup, it is described in the overall setup of the network.

7.3 Noise

Dealing with noise in the applications of Spiking Neural Networks is an important challenge (Destexhe et al., 2003). The reliability of the neural responses can be significantly impacted in the face of noise interference. However, experimental observations show that the effects of noise can be handled to produce accurate and reliable activities in the central nervous system (Shmiel et al., 2005; Tiesinga et al., 2008). Therefore, in addition to the training under

the noiseless conditions, the scenario in which noise is present is also investigated in order to verify the reliability and the robustness of the plasticity mechanisms throughout further experiments in chapter 8 and chapter 9.

Adding diffusive current noise into the deterministic LIF neuron in Equation 2.17, the membrane potential of a LIF neuron with external noise current evolves according to the following equation:

$$I_{noisy}(t) = I_{det}(t) - \mu + \sigma\xi(t) \tag{7.2}$$

where $\xi(t)$ is the Gaussian white noise process with mean μ and strength 2σ. $I_{det}(t)$ is the current supplied by the synapses from the deterministic model in Equation 2.17, $I_{noisy}(t)$ is the noise injected version. This process with constant input μ is also known as the Ornstein-Uhlenbeck process (Risken, 1989).

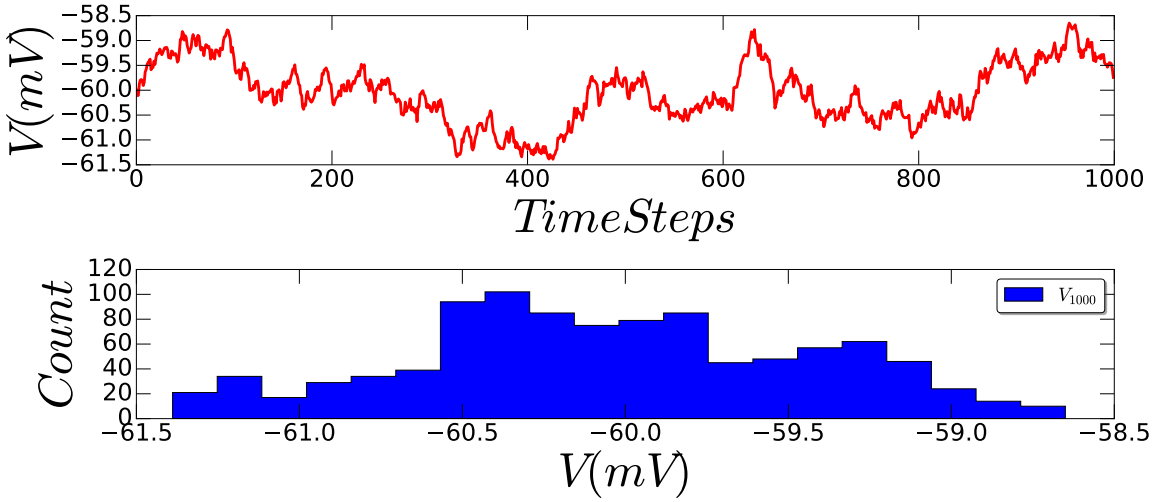


Figure 7.1: The characteristics of the low noise and how it affects neuronal activity: For $\sigma = 1.0$, the standard deviation of the membrane potential across 1000 time steps is 0.604 mV. In the top graph, the fluctuations of the membrane potential under the presence of relatively low noise is measured during 100 ms, with the time resolution $dt = 0.1$ ms. In the bottom graph, the histogram of the measured membrane potentials across 1000 time steps is illustrated. Neuron dynamics are summarized in Table 2.4.

The noise is included as additive term in synaptic input current. Two different levels of the standard deviation (SD) of I_{noisy} are used as relatively high and low noise through experiments in chapter 8 and chapter 9. The mean value of noisy current I_{noisy} described in Equation 7.2

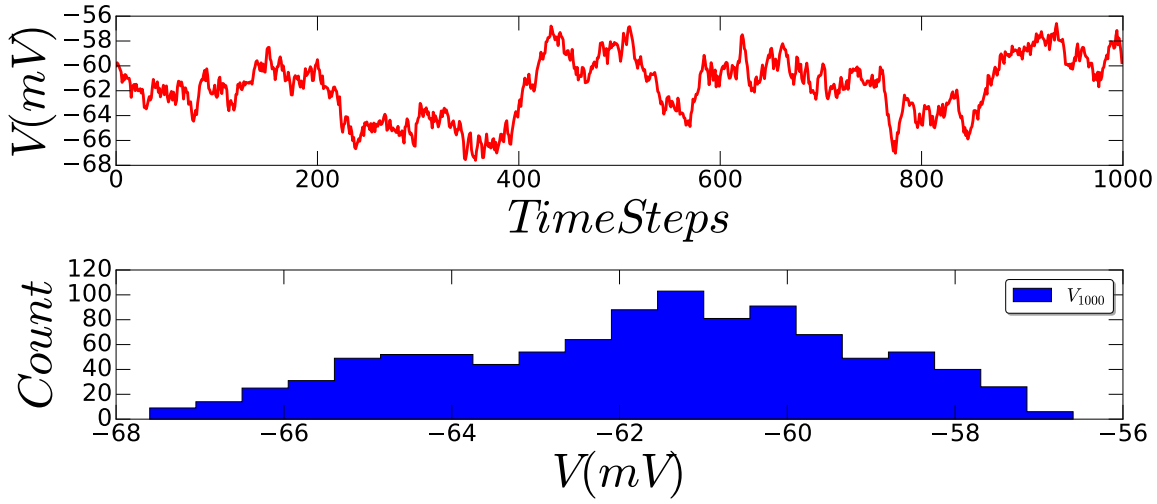


Figure 7.2: The characteristics of the high noise and how it affects neuronal activity: For $\sigma = 5.0$, the standard deviation of the membrane potential across 1000 time steps is 2.408 mV. In the top graph, the fluctuations of the membrane potential under the presence of relatively high noise is measured during 100 ms, with the time resolution $dt = 0.1$ ms. In the bottom graph, the histogram of the measured membrane potentials across 1000 time steps is illustrated. Neuron dynamics are summarized in Table 2.4.

is assumed zero as $\mu = 0$. In the case of $\sigma = 1.0$ and $\sigma = 5.0$ are assumed as relatively low noise with $SD = 0.604$ mV and relatively high noise with $SD = 2.408$ mV for the performance comparisons of plasticity under different noise levels, respectively. The characteristics of the low and high noise and how they affects neuronal activity are shown in Figure 7.1 and in Figure 7.2, respectively. The values of the membrane decay time constant, resistance and capacitance are $\tau_m = 10$ ms, $R_m = 1$ MΩ and $C_m = 10$ nF, respectively detailed in Table 2.4 and subsection 2.5.4. Selected noise levels are quite realistic according to in-vivo recordings (Destexhe et al., 2003).

To sum up, in order to inject noise to the deterministic LIF model, Ornstein-Uhlenbeck process is used based on Equation 7.2 throughout further experiments. The Ornstein-Uhlenbeck process that is often used to model a LIF neuron with a stochastic current. The noise is included as additive term in synaptic input current during further experiments. Two different noise levels are considered to compare performance of plasticities.

7.4 Delay Mechanisms

In a biological system, a post-synaptic neuron is affected from a pre-synaptic element of the synapse by the incorporation of different synaptic delays in addition to different synaptic weights (Natschlager & Ruf, 1998; Bohte et al., 2002a; Booij & Nguyen, 2005). The presence of synaptic delays reflects the time take for the Post-synaptic Potential to propagate from the dendrites to the soma, the release of neurotransmitters and the molecules of neurotransmitters to diffuse across the synaptic clefts (Sabatini & Regehr, 1999). This delay depends on the length of axon and dendrites and the conduction velocity of the action potential. The conduction delays in the axons can dominate the overall delay in some cases in neural circuit (Natschlager & Ruf, 1998; Sabatini & Regehr, 1999; Bohte et al., 2002a; Booij & Nguyen, 2005).

A synaptic connection in a Spiking Neural Network can have a single delay or multiples between the arrival of an input spike and the other end of the synaptic terminal (Natschlager & Ruf, 1998; Sabatini & Regehr, 1999; Bohte et al., 2002a; Booij & Nguyen, 2005). Also, each axonal propagation between the pre-synaptic neurons and post-synaptic neurons can be constant or plastic (Natschlager & Ruf, 1998; Bohte et al., 2002a) inspired by biological synapses. Those structures for synaptic delays are considered in the proposed architecture for Spiking Neural Networks.

For the multi-synaptic structure in the proposed topology, each individual connection from a pre-synaptic neuron i to a post-synaptic neuron j has a fixed number of K synaptic terminals, $N_{sub} = K$ with $K > 1$, where each terminal serves as a sub-connection that is associated with a delay $d_{ij,k}$ inspired from Bohte et al. (2002a). For the single-synaptic structure, there is only a single synaptic connection with $K = 1$.

In the proposed network architectures, $d_{ij,k}$ with $k \in [1, K]$ indicates the sub-terminal where i is the pre-synaptic neuron index, and k is the sub-terminal index as illustrated in Figure 7.3. The number of synapses to a learning neuron is equal to the number of input layer neurons $N_{in} = M$ times the number of synaptic terminals $N_{sub} = K$ through selected connection topologies.

For the plastic delays in the proposed architecture, the synaptic travelling time between two

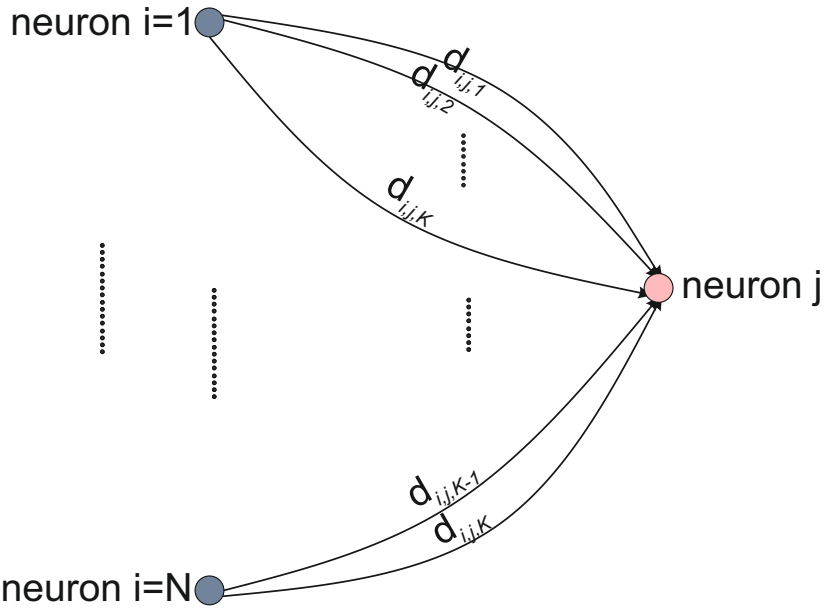


Figure 7.3: Proposed delay mechanism. K is the total number of sub-connections between each input (blue filled circles) and output neuron (pink filled circle), N is the total number of input layer neurons. i:index for input neurons with $1 \leq i \leq N$, j:index for output neuron, k:index for sub-connection with $1 \leq k \leq K$. Delays with non-negative values are $d_{ij,k} \in R^{+}$.

neurons with the specified synapse is defined by:

$$\Delta t_{ij} = t - t_i^f - d_{ij,k} \tag{7.3}$$

where t is the current time, t_i^f is the firing time of pre-synaptic neuron i, and $d_{ij,k}$ is the delay value of synaptic terminal k. Each synaptic terminal k has a delay $d_{ij,k}$. The plasticity mechanism of the axonal propagation delay is extended in section 8.9 using Equation 7.3.

Synaptic connections and delays are considered in three different ways throughout further experiments: First structure as multiple synaptic connections with constant axonal propagation speed is referred to as multi-constant-delay. The architecture of multi-constant-delay has predetermined constant delay values ($d_{ij,K}$ with $K > 1$). Another used structure as multiple synaptic connections with plastic axonal propagations is referred to multi-plastic-delay. The multi-plastic-delay has varied values of delays during learning ($d_{ij,K}$ with $K > 1$). Final used mechanism as a single synaptic connection without axonal delay is referred to single-connection. The single-connection has no delay ($d_{ij,K} = 0$ ms with $K = 1$).

The single-connection is used to compare the performance of plasticity mechanism in chapter 9 with the multi-constant-delay. On the other hand, the mechanism of the multi-constant-delay

is compared with the multi-plastic-delay architecture in section 8.7 and section 8.10.

7.5 Network Architectures

In this section, we describe the architecture of spiking neurons for obtaining spatio-temporal experiments. A fully connected two layered feed-forward Spiking Neural Network architecture without hidden layer is used in experiments. The network structure is inspired by the actor-critic architecture detailed in section 6.2.2 illustrated in Figure 7.6. All input neurons are connected to the learning neuron in the output layer. Bias neuron detailed in section 7.5.3 is only connected to the desired neuron in the output layer in order to generate desired spiking activities $S_d(t)$.

Table 7.1 gives the common structural parameters for the experiments in section 8.7 and section 8.10. The proposed structure for the SNN contains $N_{in} = M = 10$ neurons in the input layer (see Table 7.1). Each neuron in the input layer is fed with a different spike train pattern, $S_1^{in}(t)$, $S_2^{in}(t)$, ..., $S_M^{in}(t)$, driven by external Poisson spike trains which are outlined in section 7.6. In the output layer, there are two neurons, $N_{out} = 2$, one is generating the actual output patterns $S_a^{out}(t)$. The other neuron produces desired spiking activities $S_d^{out}(t)$.

Parameter Type	Parameter Names	Values
The number of input layer neurons	$N_{in} = M$	20
The number of synaptic terminals (sub-connections)	$N_{sub} = K$	10
The number of output layer neurons	$N_{out} = O$	2

Table 7.1: Parameters of the network architecture for mapping and logical operation benchmarks.

7.5.1 Training

For training structures, we use two different structures illustrated in Figure 7.5 used in Remote Supervised Method and Delayed Remote Supervised Method; and Figure 7.6 used in Reward-modulated Spike-timing Dependent Plasticity.

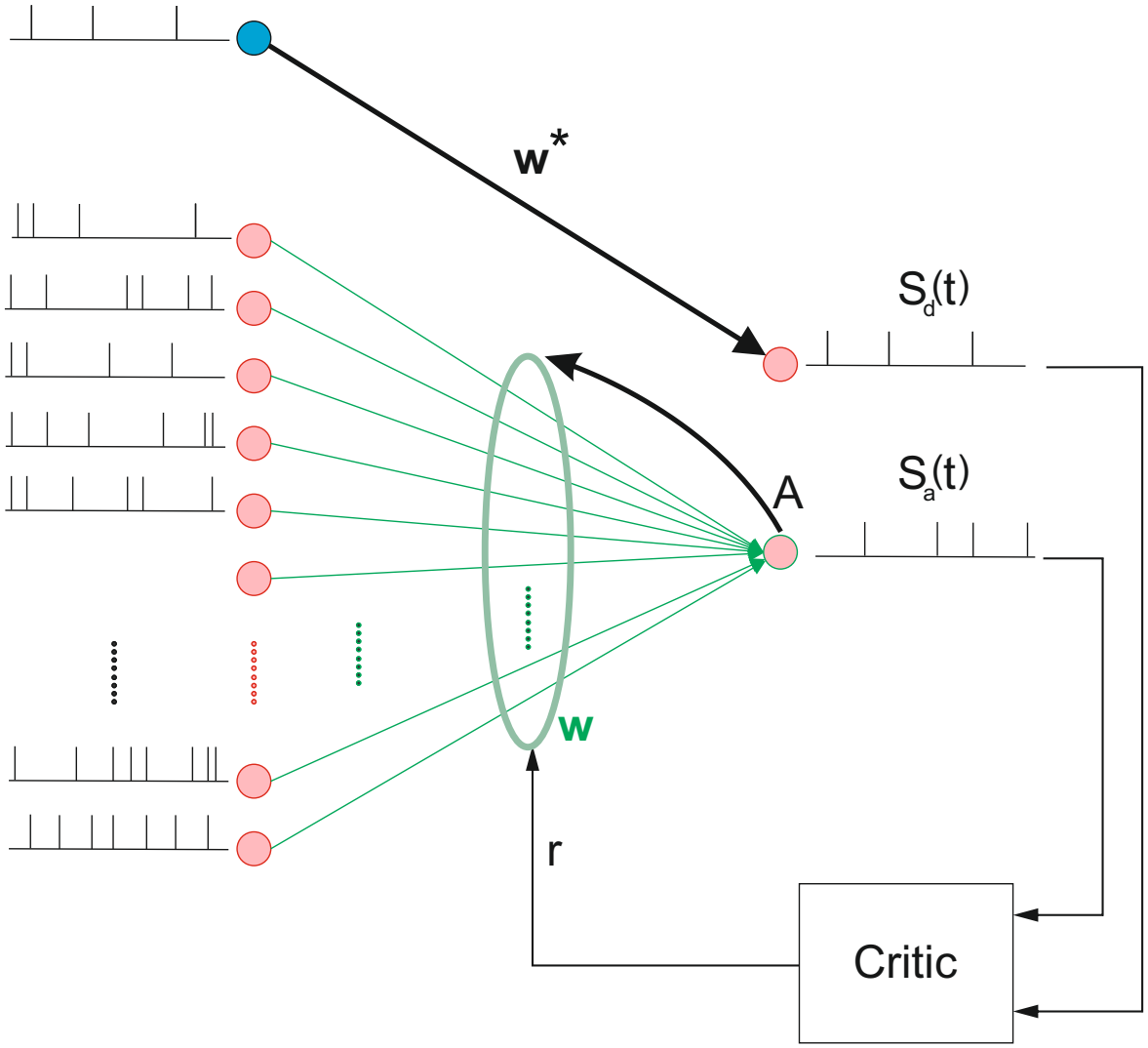


Figure 7.4: Illustration of an actor-critic network architecture. In this architecture, the synaptic strengths w are adapted with the effects of the pre-synaptic activity, the post-synaptic activity, and the reward signal r received in response to the action.

In Figure 7.5, all input neurons are connected to all output neurons according to a many-to-many projection layout. However, in Figure 7.6, all input neurons are connected to only learning output neuron. In both figures, the output neuron designated as desired output $S_d(t)$ is not responsible for active learning. It just mimics the bias neuron's behaviour. However, the actual neuron $S_a(t)$ is responsible for active learning in the SNN. The bias neuron is only connected to the desired output neuron to generate $S_d(t)$ according to a one-to-one connection scheme.

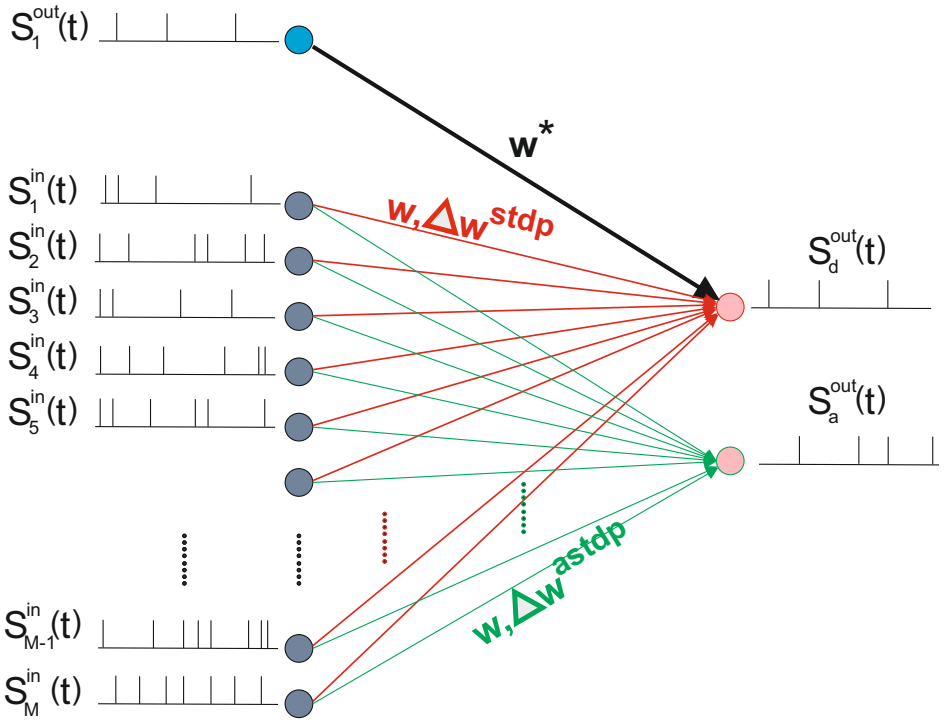


Figure 7.5: Proposed network architecture for ReSuMe training of Spiking Neural Network. All afferent neurons in input layer (grey circles) and bias neuron (blue circle) are dummy neurons. Active output LIF neuron (green shell circle), which generates actual output $S_a(t)$, is connected to all input neurons (grey circles) with weights ($\hat{w}$) equipped with anti-Hebbian-STDP (green lines). All afferent neurons are also connected to desired output neuron (red shell circle) with Hebbian-STDP dynamics (red lines). Bias neuron (red circle) is connected to passive LIF comparator neuron with fixed weight (w^*) in order to generate desired spike train $S_d(t)$. The role of bias neuron is detailed in section 7.5.3. See text for a detailed explanation of the figure and the notation used.

7.5.2 Testing

Testing the structure is illustrated in Figure 7.7, the overall structure from training does not changed apart from the bias connection and the connections between input layers to desired output. During testing, there is no any active learning; any part related with learning is removed from the network architecture. Desired neuron is kept only to monitor the performance of the network during testing cycles. It is not involved in any kind of weight/delay modifications or updates.

7.5.3 Bias Neuron

The mechanism of bias neuron is introduced for different network topologies in Figure 7.5, Figure 7.6, Figure 7.7, Figure 8.2, and Figure 8.3. The bias neuron is only connected to the

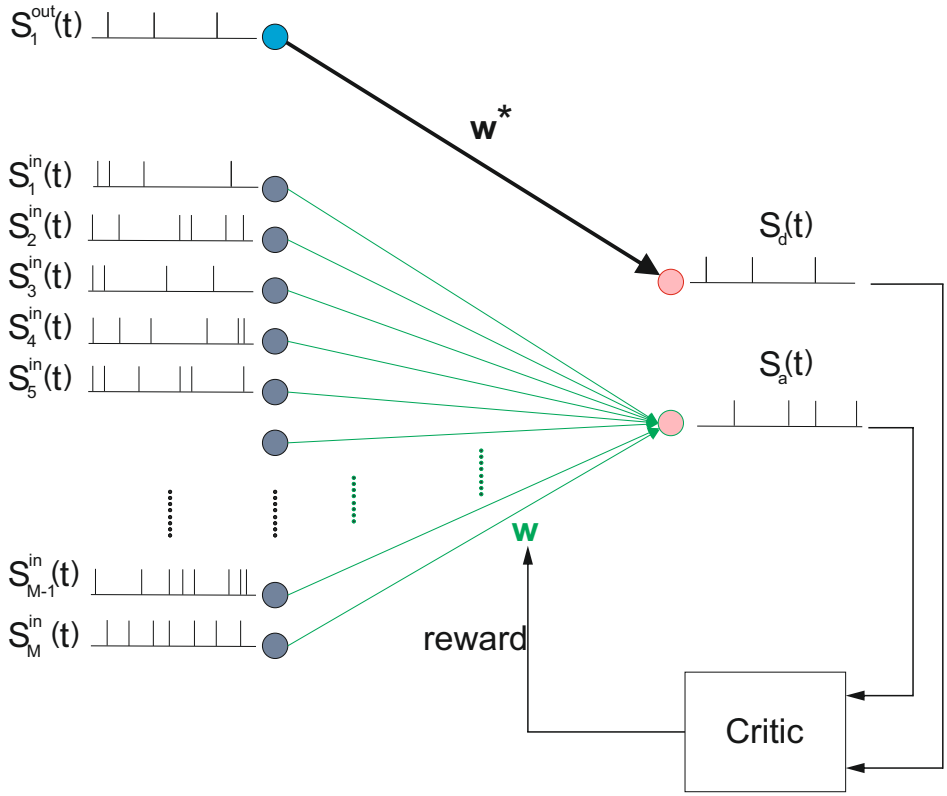


Figure 7.6: The network architecture during training for Reinforcement Learning in Spiking Neural Network. All afferent neurons in input layer and bias neuron (blue circle) are dummy neurons. Active output LIF neuron (red shell circle), which generates actual output $S_a(t)$, is connected to all input neurons (grey circles) with weights $\hat{w}$ equipped with STDP. Bias neuron (blue circle) is connected to passive LIF comparator neuron with fixed weight (w^*) in order to generate desired spike train $S_d(t)$. The role of bias neuron is detailed in section 7.5.3. See text for a detailed explanation of the figure and notation used.

desired neuron with fixed weight in the output layer of the network in order to generate desired spiking activities $S_d(t)$ illustrated in above listed figures. This mechanism is introduced as a solution for two issues: Firstly, it helps to introduce a new topology detailed in section 8.4 in order to handle the heterosynaptic plasticity mechanism for Remote Supervised Method and Delayed Remote Supervised Method. Bias neuron is a part of the proposed network topology in section 8.4 which is the alternative interpretation of synaptic remote supervision.

Another issue is to avoid the shifted version of desired patterns in the output layer. If we remove the synaptic connection between the bias input neuron and the desired output neuron by replacing it with a direct connection from bias input to output neuron, the desired output spiking activities are not generated precisely by the network during training cycles. In the direct connection scenario (without the bias neuron), the simulation delay once the actual output neuron receives incoming spikes is less than the simulation delay once the desired

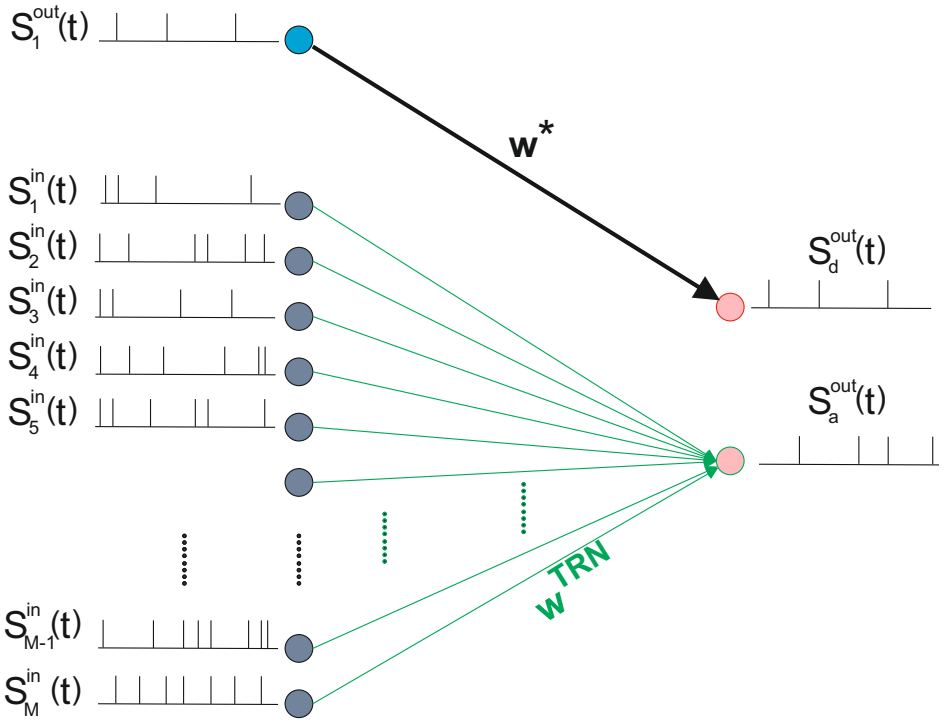


Figure 7.7: The network architecture during testing for all proposed learning techniques in Spiking Neural Network. All afferent neurons in input layer (grey circles) and bias neuron (blue circle) are dummy neurons. The role of bias neuron is detailed in section 7.5.3. Learning output LIF neuron (green shell circle), which generates actual output $S_a(t)$, is connected to all input neurons (grey circles) with trained weights $w^{\hat{TRN}}$ loaded from training cycles. Bias neuron (blue circle) is connected to passive LIF neuron with fixed weight (w^*) in order to measure the performance of each learning cycle. See text for a detailed explanation of the figure and the notation used.

output neuron receives incoming spikes from the bias neuron. This delay difference causes a shifting of desired activities at the output with the direct connection. Therefore, there is a fixed synaptic connection between the bias input and the desired output neurons.

7.6 Neural Coding

7.6.1 Spike Generation

In order to design a spike generator to provide inputs for computer simulations, the Inter-spike Interval (ISI) from a Poisson process are generated. The term of Inter-spike Interval refers to the time interval between two successive spikes within a spike train. For instance, the average time interval between two spikes is 100 ms as ISI for a neuron firing at 10 spikes/sec. The program through each time step of size dt (typically set to 1 ms) generates a random number

x_{rand} chosen uniformly in the range between 0 and 1.

$$r\delta(t) = \begin{cases} > x_{rand}, & \text{spike is fired,} \\ \leq x_{rand}, & \text{it is not fired.} \end{cases} \tag{7.4}$$

The Poisson model ignores the existence of a refractory period of neuron dynamics. However, it is not a problem during our experiments because of min-ISI considerations. After adding min-ISI, although it is not strictly Poisson any more, min-ISI mimics the refractory period in terms of biological plausibility. Spike trains generated using this mechanism feed our input neurons through our experiments in chapter 8 and chapter 9. The following section describes how those generated sets represent external information.

7.6.2 Encoding

We use spatio-temporal encoding for our experiments in chapter 8 and in chapter 9. The details of the used encoding mechanism is introduced here. Pattern selection (or stimulus) determines the number of blocks for the network. The total of N different blocks to the input can represent a maximum number of 2^N classes of patterns. For example, assume that we would like to implement a binary logical operation task which has 2 inputs with single digit each. Two features are sufficient for a group of patterns containing 4 classes “00”, “01”, “10”, “11”. The spatio-temporal mapping block acts as a converter function which translates the input features (binary strings in this case) into spike timings within the encoding window. In the example, the stimulus with binary values (“0” or “1”) are converted into a set of temporal patterns of discrete spikes. The encoding time window is determined as hundreds of milliseconds to be consistent with biological observations (Joris et al., 2004; DiCarlo et al., 2012; Salvioni et al., 2013). As a result, each input variable is represented by a group of neurons through the input layer.

The encoding part aims to generate corresponding spiking patterns that represent the input selections. Each input representation (can be single pattern as in mapping experiments or can be logical values true or false as in logic operation experiments) is converted into corresponding spatio-temporal patterns in order to feed the input layer of the SNN. We use the homogeneous Poisson spike generator discussed previously in section 6.3. Encoding generates a set of

specific activity patterns that represent attributes of external stimuli. A temporal coding scheme which uses the timing of the individual spikes to carry the information is detailed in subsection 6.4.3. Spatio-temporal encoding which is a type of temporal encoding is used here (see subsection 6.4.3).

The number of input patterns per experiment is one for the mapping benchmarks (detailed in section 7.9), the same as the number of output patterns. Therefore, the spatio-temporal mapping unit generates a single group of spike trains for the input layer. This pattern is an N_{in}-dimensional spike train and each spike train in the pattern refers to firing activities in the input layer. Each element of the pattern is connected to only one encoding neuron. Similarly, the number of input patterns per experiment is two for the two-bit logical operation benchmarks with a single output pattern.

Through ReSuMe experiments in section 8.3, DelReSuMe experiments in section 8.9 and R-STDP experiments in section 9.5, a spike train $S^{in}(t)$ for input neurons consists of a homogeneous Poisson spike train with constant spike probability $r_{in} = 0.4$ Hz for mapping and $r_{in} = 0.8$ Hz for logical operations with the minimum Inter-Spike Interval $ISI = 10$ ms (see Table 7.2). In this manner, each input neuron emits multiple spikes with firing rate r_{in}. $S^{in}(t)$ is partitioned into M subgroups of equal length, to give spike trains $S^{in}_1(t)$, $S^{in}_2(t)$, ..., $S^{in}_M(t)$. M indicates the number of neurons in the input layer (see Figure 7.5). The duration of each spike train is $T_p = 100$ ms (see Table 7.2). Each extracted spike train $S^{in}_1(t)$, $S^{in}_2(t)$, ..., $S^{in}_M(t)$ from $S^{in}(t)$ feeds each neuron in the input layer over each episode.

Parameter Type	Parameter Names	Values(Mapping)	Values(LogicalOp)
Input Firing Rate	r_{in}	0.40 Hz	0.80 Hz
Output Firing Rate	r_{out}	0.03 Hz	0.03 Hz
Spike Train Length	T_p	100 ms	100 ms
Minimum Inter-Spike Interval	ISI	10 ms	10 ms

Table 7.2: Encoding parameters for mapping and logical operation benchmarks based on ReSuMe, DelReSuMe and R-STDP.

For the output neuron, a spike train $S^{out}_d(t)$ consists of a homogeneous Poisson spike train with constant spike probability $r_{out} = 0.03$ Hz and minimum Inter-Spike Interval $ISI = 10$ ms (see Table 7.2). In this manner, each output neuron emits multiple spikes with firing rate

r_{out}. The extraction applied to the input layer does not need to be applied to $S_d^{out}(t)$ because the network contains a single training output neuron as its readout. The only restriction for the output spike extraction is no spiking activity in the first $2 * \tau_m = 20$ ms of the period, $T_p = 100$ ms. Generated output spike activities are assigned into one of two binary classes as either 0 or 1 in each time bin.

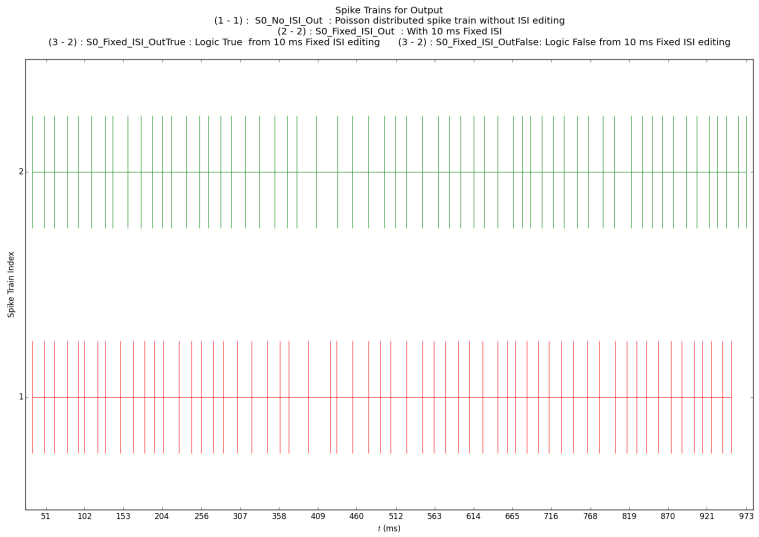


Figure 7.8: Prepared spike trains for the output.

In Figure 7.8 and the Figure 7.9, we can see a schematic of input and output spike trains, respectively. Every vertical line represents a spike train before separation. A spike train for a single presentation has a time duration of $T_p = 100ms$. Hence, during the experiments of logical operations, we generate 10 trains for input and each with a total duration of $T_p * N_P = 100ms * 10 = 1000ms$.

Similar to encoding mechanism under ReSuMe and DelReSuMe experiments, spatio-temporal encoding as a type of temporal encoding scheme which uses the timing of the individual spikes to carry the information is used for R-STDP experiments in chapter 9 as well.

Selected Poisson spike probability for both inputs and outputs allows neurons to fire multiple times within the encoding window (see Figure 7.8 and Figure 7.9). This is a more realistic approach in terms of biologically plausibility. In other words, information transmitted from pre-synaptic neurons to post-synaptic neurons is encoded in the form of a spike train instead of a single-spike as described in section 6.4.

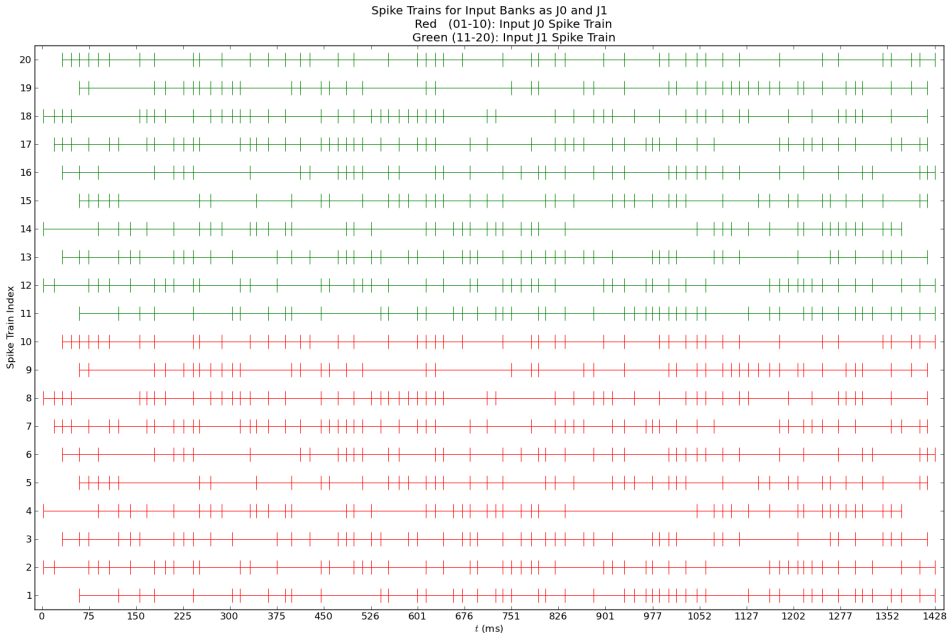


Figure 7.9: Prepared spike trains for the input. y-axis shows the spike train index. From 1 to 10 indexes with red lines belong to bank P1. From 10 to 20 indexes with green lines belong to bank P2. The total spike train time is divided into the duration of T_p to feed each neuron in the input layer during each presentation time T_{pe}.

7.7 Error Analysis

7.7.1 Measures of Spike Train Difference

The decay constant τ_R parametrizes the metric and should be selected in order to be sensitive to temporal difference between spikes. Otherwise it is sensitive to any rate relationship of spike trains. Taking into account the ISI of input and desired output trains, we use $\tau_R = 10$ ms, in Table 8.1.

In order to remove the dependence of the van Rossum Distance on the number of desired spikes, we use a normalized version of the measure D_R which is calculated as:

$$D_R^{norm} = D_R / D_R^d \tag{7.5}$$

where normalized vRD, current vRD and vRD of desired spike train, are respectively, D_R^{norm}, D_R and D_R^d. The measure is illustrated in Figure 7.10.

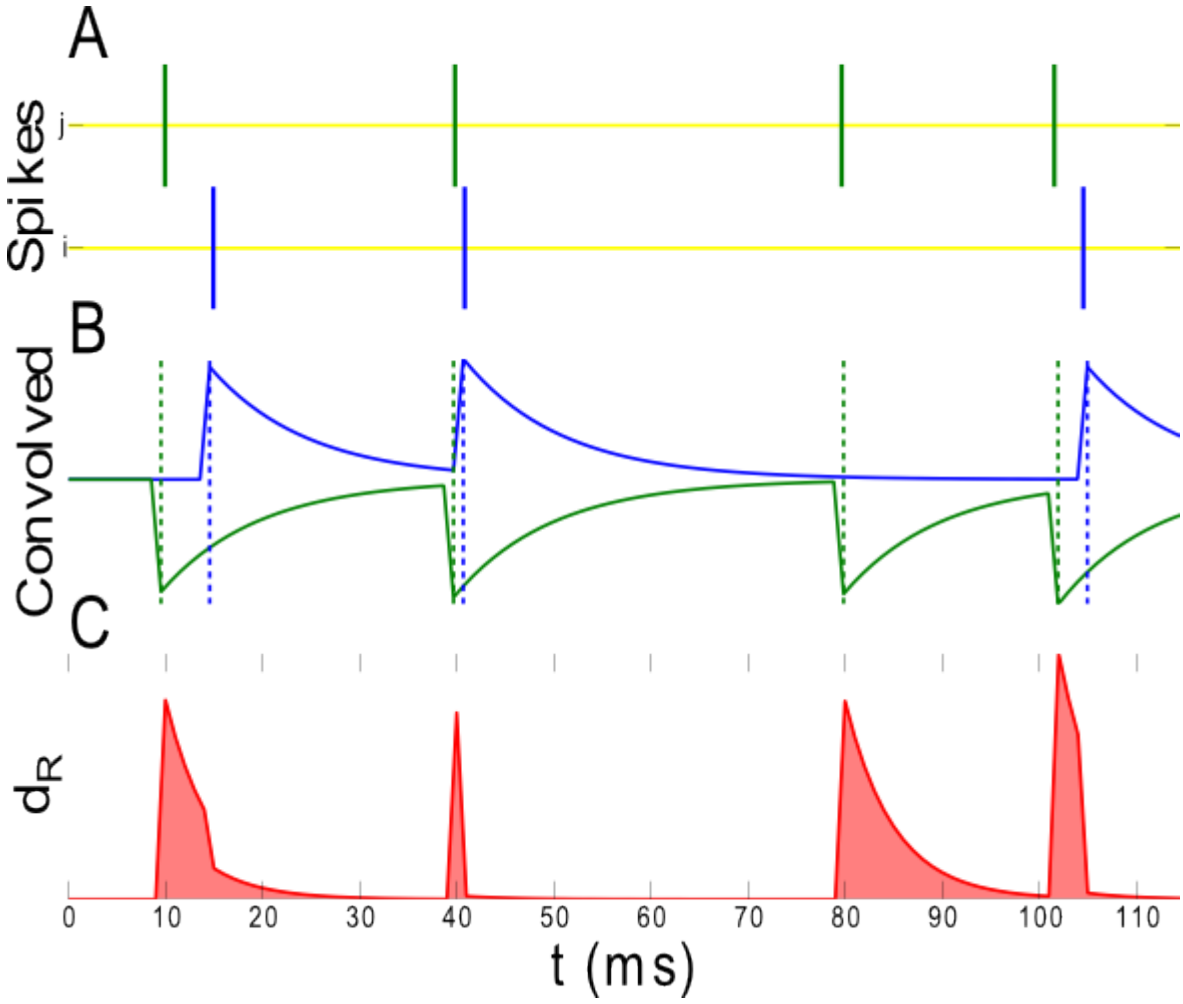


Figure 7.10: Illustration of van Rossum Distance measure. A) Two example spike trains illustrated. In the top graph, the first spike train (shown as green coloured with the spike train index j) is desired spike activity and the following train (shown as blue coloured with spike train index i) is the actual spikes over a period. B) Definition of the van Rossum Distance: Above two spike trains (actual train flipped) are convolved with Heaviside function with time constant τ_c. C) The van Rossum Distance is measured by the squared difference of the two convolved spike trains. The spike train distance is determined by the area under the curve through integration.

7.7.2 Network Classification

The details of measuring of spike train differences are detailed in section 6.5. van Rossum Distance (vRD) described in subsection 6.5.4 is one of those metrics. In addition to the vRE based on vRD, we also define a (mis)classification error for logical operation task detailed in section 7.9. It is reasonable to assume that the current network output has errors both in False-Positives (FPs) and False-Negatives (FNs) scenarios by assuming there are two valid (binary) outputs. A False-Positive indicates a training error that a network output is TRUE (1) and the desired response to the same input pattern is FALSE (0). Similarly, a False-Negative

is also a training error that a network output is FALSE (0) and the desired response to the same input pattern is TRUE (1).

Differentiating the error made by the network between False-Positives and False-Negatives is not necessary for our experiments. Therefore, we define a straightforward (mis)classification rate as *Logic Classification Error (LCE)* which counts the number of false outputs by combining False-Positive and False-Negative errors.

We describe the desired output as S_d and anti-desired output as $S_d^{'}$. $S_d^{'}$ is the inverse of the desired output in terms of the logic equivalent. If the spike distance between actual and desired trains, $D_R(S_a, S_d)$, is more than the distance between actual and anti-desired trains, $D_R(S_a, S_{d'})$, the network output (actual) is labelled $Q^{'}$ which is a misclassification. However, if the spike distance between actual and anti-desired trains, $D_R(S_a, S_{d'})$, is more than the distance between actual and desired trains, $D_R(S_a, S_d)$, the network output (actual) is labelled as the same as Q which is a correct classification. This explanation is summarized as:

$$LCE_P = \begin{cases} \text{if } D_R(S_a, S_d) >= D_R(S_a, S_{d'}) & S_a \Rightarrow Q & \rightarrow \text{Correct Classification,} \\ \text{otherwise} & S_a \Rightarrow Q^{'} & \rightarrow \text{Incorrect Classification.} \end{cases} \tag{7.6}$$

where LCE_P is the Logic Classification Error through a presentation P. For an example presentation, if inputs are ($P1$, $P2$) = (1, 0), the outputs are (Q, $Q^{'}$) = (0, 1) for the operation of AND. Details of all possible scenarios for all operations can be seen in section A.5. The full formula with the designed testing loop can be seen in Equation A.7.

During each testing epoch, an epoch based LCE_E with the percentage error is calculated as:

$$LCE_E = \frac{\text{Number of misclassification}}{N_P^{TST} N_{Sp}^{EXP}} * 100\% \tag{7.7}$$

where N_P^{TST} is the number of presentations in a single testing epoch. N_P^{TST} corresponds to the total number of classifications during an epoch. N_{Sp}^{EXP} is the number of spike sets performed during each experiment. Therefore, LCE is measured as a percentage of the total error count without separating False-Positives and False-Negatives.

To sum up, we use two types of error metrics in order to validate the results, vRE and LCE. Although vRE has more details about error trajectories, proposed error measure, LCE, gives another aspect about the network performance.

7.8 Training and Testing Mechanisms

To simulate Spiking Neural Networks in software, the Brian package (Goodman & Brette, 2008) based on Python language has been used. The Python code for all simulations including neurons, synapses and network models based on Brian architecture is available on the author's web page [1].

Each experiment has at least two network structures for separate phases of training and testing described in subsection C.6.1. Training and testing phases also have slightly different flows throughout their iterative processes.

7.8.1 Training Mechanism

Neurons and synapses are monitored during simulation. The spiking activity of each neuron is recorded throughout the simulation (using monitor objects in Brian). Because of modularity, different neuron models can easily be selected in the proposed framework.

In order to simulate experiments and analyse results, we use the same framework Appendix C. The pseudo-code of training procedure is illustrated in Algorithm 5. Because of memory usage, we record values onto disk at the end of each presentation run (see section C.2). Recorded parameters during this experiment can be listed as output firing times, current reward value and error.

During the training process, we feed the inputs with randomly generated spike patterns based on the specified parameters (see section C.10). Then, the resulting spikes are propagated through the network at each presentation. Adjustable parameters (such as weights and/or delays) associated with the input patterns are updated with each presentation. The details of the training procedure in the proposed framework is outlined in subsection C.6.1.

[1] http://www.ozturkibrahim.com

The pseudo-code of the training procedure is illustrated in Algorithm 4. Once a network has been constructed at the beginning of the training process, the initial weights/delays (described later) are chosen from initialization sets based on the initialization type. The network is then stimulated for a specific time period, T_{pe} at each presentation. The network processes the input spike patterns based on the distribution of input types (described in section C.9) using the weights and/or delays and synaptic functions. Spiking activities from all neurons in the network can be monitored and recorded if it is desired.

Algorithm 4 Pseudocode of Typical Training Process for ReSuMe/DelReSuMe

procedure TRAINING(*LearningAlg*, *Op*, N_E, N_P, *SpSetId*, *InitId*)
    Create the feed-forward SNN based on structural parameters (neurons, synapses, layers, initializations, connections, monitors).
    Load Spike set based on selected *SpSetId*.
    Load Initialization set based on selected *InitId*.
    Assign all initial values such as weights and delays based on selected methods.
    for $Epoch \leftarrow 0, N_E$ **do**
        Generate inputs based on the distribution of input types.
        Prepare corresponding outputs based on the selected task.
        for $Presentation \leftarrow 0, N_P$ **do**
            Choose corresponding spike patterns for input and outputs from loaded spike set for the current presentation.
            Apply spike patterns for the input and output.
            Run the network for the time T_{pe}.
            Use learning algorithm (*LearningAlg*) to calculate next value of adjustable parameter.
            Update adjustable parameters such as the weights as w $\leftarrow$ w + Δ w.
            Update adjustable parameters such as the delays as s $\leftarrow$ s + Δ s.
            Store recorded parameters (weight, reward, firing activities) onto disk.
            Set presentation $\leftarrow$ presentation + 1.
        end for
        Apply weight scaling if it is active based on the scaling type.
        Reset the state of all neurons and traces.
        Set epoch $\leftarrow$ epoch + 1.
    end for
end procedure

Synaptic weights/delays are continually refined during training using the difference between the actual outputs and the desired outputs. At the end of each epoch, we reset the state of all neurons and traces (such as pre- and post-synaptic traces, eligibility traces) to prepare them for the following epoch. The stop criterion adopted in this algorithm is the maximum epoch number of training N_E^{TRN}.

The reward is calculated based on the timing difference between action potentials of the actual neuron and desired neuron. The error measure is calculated in the critic unit. The reward signal is delayed by T_{pe} then presented at the end of each trial. The generated reward modulates the synaptic weights based on Spike-timing Dependent Plasticity rule. All synaptic strengths are updated based on the performed action with the modulation of the reward factor at the end of each current learning cycle as described in Equation 9.7.

Algorithm 5 Pseudocode of Training Process during Reward-modulated Spike-timing Dependent Plasticity

procedure TRAINING(*LearningAlg*, *Op*, N_E, N_P, *SpSetId*, *InitId*)
    Create the feed-forward SNN based on structural parameters (neurons, synapses, layers, initializations, connections, monitors).
    Load Spike set based on selected *SpSetId*.
    Load Initialization set based on selected *InitId*.
    Assign all initial values such as weights, delays, rewards based on selected methods.
    for $Epoch \leftarrow 0, N_E$ **do**
        Generate inputs based on the distribution of input types.
        Prepare corresponding outputs based on the selected task.
        for $Presentation \leftarrow 0, N_P$ **do**
            Choose corresponding spike patterns for input and outputs from loaded spike set for the current presentation.
            Apply spike patterns for the input and output.
            Run the network with the duration T_{pe}.
            Use learning algorithm (*LearningAlg*) to calculate next value of adjustable parameter.
            Update adjustable parameters such as the weights as w $\leftarrow$ w + Δ w.
            Store recorded parameters (weight, reward, firing activities) onto disk.
            Set presentation $\leftarrow$ presentation + 1
        end for
        Apply weight scaling if it is active based on the scaling type.
        Reset the state of all neurons and traces.
        Set epoch $\leftarrow$ epoch + 1.
    end for
end procedure

7.8.2 Testing Mechanism

At the end of the training simulations, we analyse results by reading from the stored training file. During testing for all types of experiments in chapter 8 and in chapter 9, the followed procedure is described in Algorithm 6.

Algorithm 6 Pseudocode of Typical Testing Process for ReSuMe/DelReSuMe

```
1: procedure TESTING(TrainingSessId)
2:     Reload parameters as LearningAlg, Op, N_E based on TrainingSessId.
3:     Create the feed-forward SNN based on given parameters and the current training
   session.
4:     Reload all trained values with trajectories to test each training step (such as weights
   and/or delays).
5:     Generate all possible testing inputs and corresponding outputs (number of possible
   testing pairs = N_P).
6:     Reload the spiking set (used in the dependent training session).
7:     for Epoch ← 0, N_E do
8:         Apply suitable values from training trajectories (such as weights and/or delays)
   based on the epoch number.
9:         for Presentation ← 0, N_P do
10:            Choose input values (such as pairs of (0,0), (0,1), (1,0), (1,1)).
11:            Choose corresponding spike patterns for the current inputs.
12:            Apply spike patterns for input.
13:            Run the network with the duration T_pe.
14:            Record firing activities of output neurons.
15:            Set presentation ← presentation + 1.
16:        end for
17:        Reset the state of all neurons.
18:        Set epoch ← epoch + 1.
19:    end for
20: end procedure
```

At the beginning of the testing cycles, all trained parameters are reloaded from the dependent training session. The performance of each recorded training step is measured with all possible input combinations. There are no adjustments related to any configurable parameters (weights/delays). At the end of each testing epoch, the states of all neurons are reset. The stop criterion adopted in this algorithm is the maximum epoch number of testing N_E^{TST}, this is the same as the dependent training epoch number N_E^{TRN} (see Table C.2). The pseudocode of testing procedure is shown in Algorithm 6. Details of the testing procedure in the proposed framework is outlined in subsection C.6.1.

7.9 Benchmark Descriptions

7.9.1 Mapping

The first benchmark implementation is to map the timings of input spike patterns into the target output patterns precisely. This experiment gives initial indications about the

performance of the proposed method. The main learning phenomena relied on here is the framework of ReSuMe described in section 8.3.

Figure 7.11: Block diagram of mapping benchmarks with spike trains. Input spike trains are $S^{in}_{P1,1}(t)$, $S^{in}_{P1,2}(t)$, ..., $S^{in}_{P1,N_{P1}}(t)$ with single output spike train $S^{out}_{Q,1}(t)$. $P1$ and Q are the label for input and output, respectively.

Figure 7.11 illustrates the block diagram of the benchmark. There is a single input bank named $P1$. A spatio-temporally encoded spike pattern is applied to the input bank ($P1$). The spiking network is trained to produce a desired spike train which corresponds to the input pattern at the output (Q).

7.9.2 Logical Operations

Figure 7.12 demonstrates the block diagram of the logical operations benchmark. There are two input banks named $P1$ and $P2$ in the input layer. A spatio-temporally encoded spike pattern is applied to each bank for a logical value FALSE (0) or TRUE (1). The spiking network is trained to perform a logical operation and produces a spike train which corresponds to each proper logical value at the output named Q.

This benchmark can be executed in order to validate the network's performance in the case of multiple input-output patterns. We test some of the learning algorithms with a number of logical operations P1, AND, OR, XOR. The truth tables of operations can be seen in section A.5.

7.10 Summary

Common concepts throughout performed simulations in chapter 8 and chapter 9 are introduced in this chapter. Firstly, types of used neuron models considering noise are introduced. Used

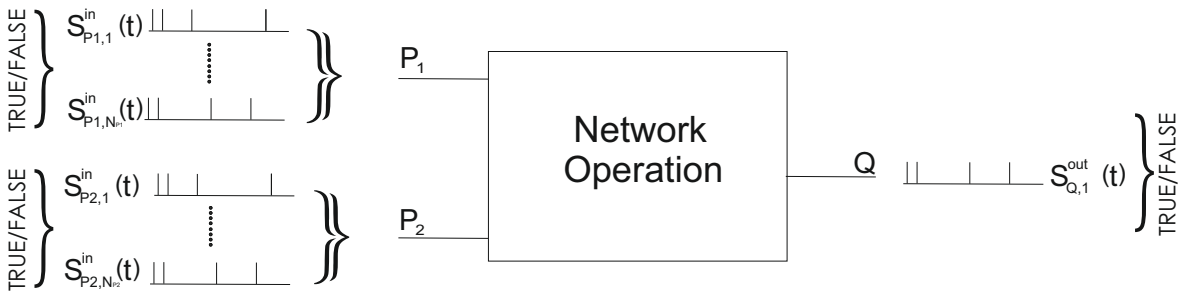


Figure 7.12: Block diagram of logical operation benchmarks with spike trains. Input spike trains are $S^{in}_{P1,1}(t)$, $S^{in}_{P1,2}(t)$, ..., $S^{in}_{P1,N_{P1}}(t)$ for input bank $P1$ and $S^{in}_{P2,1}(t)$, $S^{in}_{P2,2}(t)$, ..., $S^{in}_{P2,N_{P2}}(t)$ for input bank $P2$ with single output spike train $S^{out}_{Q,1}(t)$. $P1$, $P2$ and Q are the name of banks for first input, second input and output, respectively. Bank refers to each subsection in the layer. For instance, here input layer has two banks $P1$ and $P2$, corresponding to 2-bits.

synaptic delay mechanism is also detailed. In this mechanism, multiple synapses transmitting multiple spikes from a pre-synaptic neuron to a post-synaptic neuron. Then, the details of network architecture for training and testing cases are described. The details of proposed encoding mechanism of spatio-temporal patterns is summarized with Poisson spike generator. Then, error metrics are detailed in order to evaluate spike-train (dis)similarity and the performance of the network. Also, the pseudocodes of training and testing mechanisms are described. Finally, used benchmarks during experiments in chapter 8 and chapter 9 are briefly introduced.

Chapter 8

ReSuMe and DelReSuMe

Contents

8.1 Introduction . . . 177
8.2 SpikeProp . . . 179
8.3 Remote Supervised Method . . . 181
8.4 Architecture for Proposed Topology . . . 184
8.5 Error Analysis . . . 189
8.6 ReSuMe Overall Setup Experiments . . . 190
8.7 Experiments: Mapping for Remote Supervised Method . . . 191
8.7.1 Results of Noiseless Simulations . . . 191
8.7.2 Results of Noisy Simulations . . . 195
8.7.3 Discussion of Mapping . . . 198
8.8 Experiments: Logical Operations for Remote Supervised Method . . . 199
8.8.1 Results of Noiseless Simulations . . . 200
8.8.2 Discussion . . . 202
8.9 Delayed Remote Supervised Method . . . 206
8.10 Experiments: Mapping for Delayed Remote Supervised Method . . . 209
8.10.1 Results of Noiseless Simulations . . . 210
8.10.2 Results of Noisy Simulations . . . 214
8.10.3 Discussion of Mapping . . . 215
8.11 Summary . . . 217

8.1 Introduction

One important topic in Spiking Neural Networks (SNNs) is supervised learning based on temporal coding, since spiking neurons have a structure more like real neurons (Maass, 1996). However, the exact mechanisms of supervised learning in biological systems remains unclear (Thorpe & Gautrais, 1998; Ponulak & Kasinski, 2010). Today one of the main challenges is

to discover efficient learning rules without losing the biological realism of actual spiking neurons.

In neurons, the firing time of input and output spikes encodes the input and output information (Gerstner & Kistler, 2002). According to the number of spikes per presentation during learning, supervised learning based on temporal coding schemes can be divided into two forms: single-spike and multi-spike learning (Booij & Nguyen, 2005; Schrauwen & Campenhout, 2004; Gruning & Sporea, 2012; Sporea & Gruning, 2013). In single-spike learning, neurons are allowed to fire only once at the desired firing time; this restriction makes it suitable only for time-to-first-spike coding described in section 6.4. However, in multi-spike learning there is no restriction on spiking activities. There is much research based on single-spike learning (Bohte et al., 2002b; Schrauwen & Campenhout, 2004; Booij & Nguyen, 2005; Kasinski & Ponulak, 2006; Ponulak & Kasinski, 2010). Although spiking neurons have greater representational capacity than Multi-Layer Perceptrons (MLPs) (Maass, 1996), single-spike learning is not consistent biologically, and it restricts the ability of SNNs to express information. For biological plausibility, it is necessary to use multi-spike learning instead of single-spike coding (Booij & Nguyen, 2005; Kasinski & Ponulak, 2006; Ponulak & Kasinski, 2010). Also, increasingly large networks must be used in order to handle big problems with single-spike codings. Therefore, we focus on multi-spike coding throughout the thesis.

Richer dynamics of SNNs encourage development of new learning rules inspired from biological mechanisms. A Back Propagation like algorithm named SpikeProp is the first general Machine Learning rule which applied the principle of Back Propagation to SNNs (Bohte et al., 2002b). Instead of adjusting the average firing rate of neurons in Back Propagation, SpikeProp adjusts the precise spike time of each neuron, which is restricted to single-spike encoding. In order to present further algorithms for SNNs, it is important to understand the derivation of SpikeProp from Back Propagation, so the mathematics of SpikeProp is reproduced in section 8.2.

The Remote Supervised Method (ReSuMe) learning rule described by Ponulak & Kasinski (2010) is a supervised learning method in which the trained neuron can fire at desired times in response to spatio-temporal input patterns inputs. Unlike SpikeProp, ReSuMe is capable of generating a desired spike train with multiple spikes. The main contributions in this chapter are a architecture concept for existing heterosynapticities such as ReSuMe and a novel extension of ReSuMe with better performance. Through the proposed ReSuMe architecture

described in section 8.3, the problem of getting multiple spikes into and out of the Spiking Neural Network is solved, and the task of mapping is fulfilled. Through mapping experiments in ReSuMe, the SNN learns the desired firing times of the output neurons by adapting the synaptic weights. Then, more challenging experiments with logical operations are carried out with ReSuMe. We test the proposed network structure and learning with logical tasks such as TRUE, P1, AND, OR and XOR.

A novel learning rule, Delayed Remote Supervised Method (DelReSuMe), is introduced, that extends ReSuMe from only weight tuning into both weight and delay learning as described in section 8.9. This hypothesis is tested with mapping benchmarks in section 8.9 which examines their results. The mapping benchmark, previously introduced in subsection 7.9.1, is to map the timings of input spike patterns into the target output patterns precisely. Through these tests, learning the desired firing times of the output neurons has faster convergence than ReSuMe because synaptic delays are also plastic in addition to the plasticity of weights.

This chapter presents the mathematical models with following experimental results for ReSuMe in section 8.3 and DelReSuMe in section 8.9 after introducing SpikeProp in section 8.2, which is one of the first supervised learning techniques for SNNs. We also propose an alternative topology for ReSuMe and DelReSuMe in section 8.4 with experimental evidences which can mimic exactly the same behaviour of ReSuMe. In section 8.5, we summarize the selected parameters of error metrics in order to validate the results of the learning procedures on the basis of previously introduced metrics in section 7.7. In section 8.6, we summarize common details for both benchmarks: mapping experiments in section 8.7 and logical operation experiments in section 8.8. In order to validate robustness of the network, we also execute the mapping benchmarks under low noise and high noise conditions: for ReSuMe in subsection 8.7.1 and in subsection 8.7.2, for DelReSuMe in subsection 8.10.1 and in subsection 8.10.2, respectively. Also, ReSuMe tasks are performed for logical operation benchmark introduced in subsection 8.8.1.

8.2 SpikeProp

One of the first supervised learning techniques for SNNs is SpikeProp, analogous to the classical Back Propagations (BPs) in traditional Neural Networks (Bohte et al., 2002b). SpikeProp

is a gradient-descent learning algorithm similar to Back Propagation for rate coding. The euclidean distance between target and actual spiking activity determines the synaptic weights to minimize the error. However, applications of SpikeProp are mainly based on the timing of a single output spike (Booij & Nguyen, 2005; Schrauwen & Campenhout, 2004). Although Booij & Nguyen (2005) claims that it can be applicable for multiple spikes, there is limited evidence. Also, Gruning & Sporea (2012); Sporea & Gruning (2013) explain that SpikeProp fails during their preliminary experiments for multiple output spikes.

The proposed network topology for SpikeProp is a fully connected Feed-forward Network of spiking neurons that use exact spike time temporal coding (Bohte et al., 2002b). Each connection between neurons has multiple synaptic terminals with fixed multiple delays. Only the synaptic weights can be adjusted during training. The neuron activity is characterized by the Spike Response Models (SRMs) discussed in subsection 2.5.5. The neuron model in the SpikeProp assumes that each neuron can only fire once during a given period.

The goal of the algorithm is to learn the desired spiking times, denoted t_j^d, at the output neurons $j \in J$. The algorithm performs in much the same way as Back Propagation works. The results of a learning mechanism are applied into network layers labelled with I (input), H (hidden) and J (output). The error function is chosen as the mean square error defined by:

$$E = \frac{1}{2} \sum_{j \in J} (t_j^a - t_j^d)^2 \tag{8.1}$$

where t_j^a and t_j^d are actual and target output spike timings of post-synaptic neuron j, respectively. Learning is the process of modifying the synaptic weights according to the time difference between pre-synaptic firing times and the post-synaptic firing times. In each learning step, all weights are gradually adjusted in the direction minimizing E according to the gradient descent method as:

$$\Delta w_{ij,k} = -\eta \frac{\partial E}{\partial w_{ij,k}} \tag{8.2}$$

where η is the learning rate, $\eta > 0$ and $w_{ij,k}$ is the weight of sub-connection k from neuron i to neuron j. As t_j is a function of x_j, which depends on the weights $w_{ij,k}$, the gradient term in Equation 8.2 is expanded using the chain rule to:

$$\frac{\partial E}{\partial w_{ij,k}} = \frac{\partial E}{\partial t_j^a}(t_j^a) \frac{\partial t_j^a}{\partial x_j(t)}(t_j^a) \frac{\partial x_j(t)}{\partial w_{ij,k}}(t_j^a) \tag{8.3}$$

where t_j^a is the actual spiking time of the post-synaptic neuron j. SpikeProp starts with the neurons at the output layer of the network. Then it looks back at the neurons that connect to it from the input layer.

In the learning paradigm, the set of temporal input data is memorized into the weights. However, one of the main drawbacks of SpikeProp is that it is only capable of learning the first spike per input neuron, so subsequent activities are ignored. It cannot learn to reproduce a spike train with multiple spikes. Also, the SpikeProp algorithm is designed for training the weights only. To address these weaknesses, we have made improvements with ReSuMe in section 8.3 and then proposed a novel technique in section 8.9 with further improvements.

8.3 Remote Supervised Method

The Remote Supervised Method (ReSuMe) learning rule described by Ponulak & Kasinski (2010) is a supervised learning method in which the trained neuron can fire at desired times in response to spatio-temporal input patterns inputs. Unlike SpikeProp, ReSuMe is capable of generating a desired spike train with multiple spikes. The ReSuMe is derived from the Widrow-Hoff (WH) rule (Widrow, 1990) and is based on the interaction of two Spike-timing Dependent Plasticity learning windows. Here, we demonstrate the mathematics of the ReSuMe, using an alternative mathematical notation to have consistency throughout the thesis.

Applying the Hebbian Asymmetric-STDP process (see section 3.4) in Equation 3.6 into the difference of firing times between input spike i and the desired output spike j, $\Delta t = t_j^d - t_i$, synaptic weights of those synapses can be modified according to:

$$\Delta w_{ij}^d(t_i, t_j^d) = a + \begin{cases} +A_{pre} e^{-\frac{t_j^d - t_i}{\tau_{pre}}}, & \text{if} \quad t_j^d - t_i \geq 0, \\ -A_{post} e^{+\frac{t_j^d - t_i}{\tau_{post}}}, & \text{if} \quad t_j^d - t_i < 0. \end{cases} \tag{8.4}$$

where t_j^d and t_i are the desired and pre-synaptic firing times, respectively. The constant parameter $a \geq 0$ is non-Hebbian factor which defines the amplitude of the noncorrelation contribution to the total weight update. The non-Hebbian factor helps to set the instantaneous firing rate of the post-synaptic neuron without taking into account a precise timing of the particular spikes (Ponulak, 2008). A_{pre} and A_{post} are the maximum change for weight

potentiation and depression, respectively. τ_{pre} and τ_{post} are the width of learning windows for potentiation and depression, respectively. The shape of the learning window $Ae^{-\frac{\Delta t}{\tau}}$ is exponential here because of better performance of convergence but it does not have to be exponential (Ponulak, 2008).

The anti-Hebbian Asymmetric-STDP process (see section 3.4) which weakens distal synapses, and strengthens proximal synapses is discussed in section 3.4. This opposition is handled by reversing the sign of amplitudes for LTP and LTD. Applying the anti-Hebbian Asymmetric-STDP process in Equation 3.6 into the difference of firing times between input spike i and the actual output spike j, $\Delta t = t_j^a - t_i$, synaptic weights of those synapses can be updated according to:

$$\Delta w_{ij}^a(t_i, t_j^a) = -a + \begin{cases} -A_{pre} e^{-\frac{t_j^a - t_i}{\tau_{pre}}}, & \text{if} \quad t_j^a - t_i \geq 0, \\ +A_{post} e^{+\frac{t_j^a - t_i}{\tau_{post}}}, & \text{if} \quad t_j^a - t_i < 0. \end{cases} \tag{8.5}$$

where t_j^a and t_i are actual and pre-synaptic firing times, respectively.

If actual and desired spiking times are the same, the Hebbian rule and anti-Hebbian rule for Asymmetric-STDP processes are balanced by $\Delta w_{ij}^a(t_i, t_j^a) = -\Delta w_{ij}^d(t_i, t_j^d)$. As a result, there are no weight changes because of those spikes.

According to ReSuMe, the total change of synaptic weight Δw_{ij} is modified by combining the Hebbian term in Equation 8.4 and anti-Hebbian based Asymmetric-STDP term in Equation 8.5 with contributions from input-output spike pairs according to the following equation:

$$\Delta w_{ij} = \sum_{t_i \in S_i} \Big(\sum_{t^d \in S_j^d} \Delta w_{ij}^d(t_i, t_j^d) + \sum_{t^a \in S_j^a} \Delta w_{ij}^a(t_i, t_j^a) \Big) \tag{8.6}$$

For our experiments, amplitudes of synaptic efficacies A_{pre}, $A_{post} \geq 0$ are set as $A_{pre} = A_{post} = 0.005$ summarized in Table 8.1. Decay time constants τ_{pre}, $\tau_{post} \geq 0$ and they are set as $\tau_{pre} = \tau_{post} = 5ms$ here. Also, non-Hebbian factor is set to 0 because we already use activity-dependent synaptic scaling (see subsection 3.5.1) in order to get desired activation level at the post-synaptic neuron, $a = 0$.

Parameter Type	Parameter Names	Values
Plasticity amplitudes	A_{pre}, A_{post}	0.005, 0.005
Decay constants	τ_{pre}, τ_{post}	5ms, 5ms
non-Hebbian Factor	a	0

Table 8.1: Model parameters used for ReSuMe used in Figure 8.1.

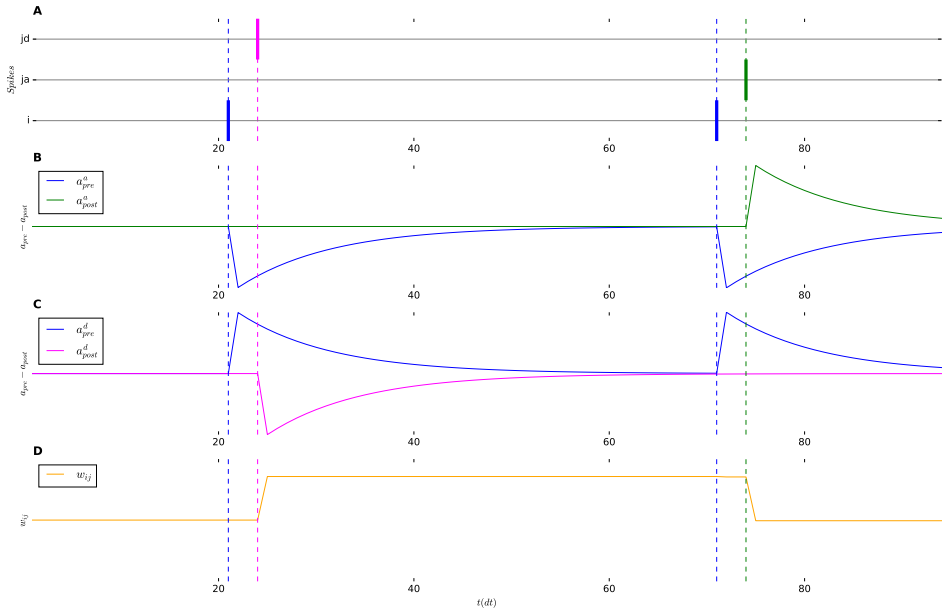


Figure 8.1: Illustration of concepts underlying the ReSuMe learning from Ponulak (2008). **A)** The row of i at the bottom of (A) illustrates pre-synaptic n_i neuron's activity coloured magenta. The row of ja and jd represent the actual (coloured green) and desired (coloured blue) activity in ReSuMe learning, respectively. **B)** LTP (in green subwindow) and LTD (in blue subwindow) for excitatory connections can bee seen with a^a_{pre} and a^a_{post} based on pre-synaptic and actual output spikes from (A). A_{pre} and A_{post} in Table 8.4 are maximum amplitude of a^a_{pre} and a^a_{post} traces, respectively. **C)** LTP (in blue subwindow) and LTD (in magenta subwindow) for excitatory connections can bee seen with a^d_{pre} and a^d_{post} based on pre-synaptic and desired spikes from (A). A_{pre} and A_{post} in Table 8.4 are maximum amplitude of a^d_{pre} and a^d_{post} traces, respectively. **D)** Changes of the synaptic efficacy w_{ij} are triggered by the teacher (desired) or post-synaptic (actual) activity at times 25 ms and 72 ms, respectively. The level of change is determined by the learning windows in (B) and (C).

Figure 8.1 demonstrates the ReSuMe learning mechanism based on the described parameters in Table 8.1. The combined synaptic efficacy depends both on the correlation between the pre-synaptic and actual post-synaptic firing activities and on the correlation between the pre-synaptic and desired firing activities.

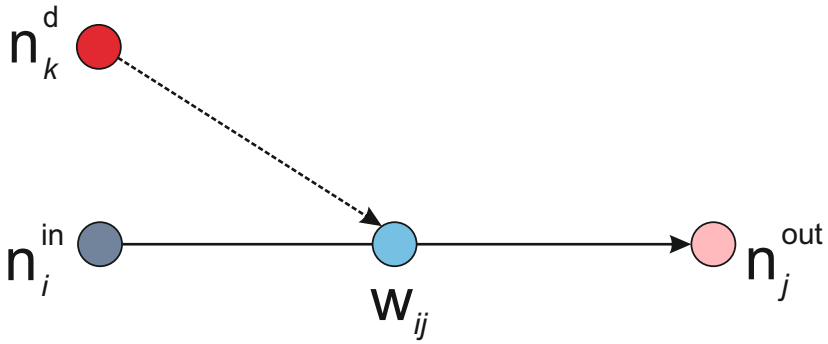
Regulation of synaptic weights and neuronal activity can only be adjusted based on pre-synaptic

stimulation (Lynch et al., 1977). Heterosynaptic plasticity is observed in the hippocampus as a result of modulatory inter-neurons (Lynch et al., 1977). Changes in synaptic strengths during plasticity can be accompanied by changes at nearby synapses through heterosynaptic modulation. Therefore, the amount of depression and potentiation in these mechanisms is balanced by a local mechanism of both normalization of synaptic weights and synaptic competition (Chistiakova et al., 2015). Various forms of heterosynaptic plasticity are observed in a variety of brain regions and organisms during associative learning, the development of neural circuits, and homoeostaticity (Bailey et al., 2000). However, they can be categorized in two forms: non-associative or associative. The non-associative form is only heterosynaptic, whereas associative, activity-dependent heterosynaptic modulation combines homosynaptic and heterosynaptic plasticity (Bailey et al., 2000). The associative heterosynaptic concept as a biologically plausible phenomenon is used through experiments in this chapter and it is described in the following section.

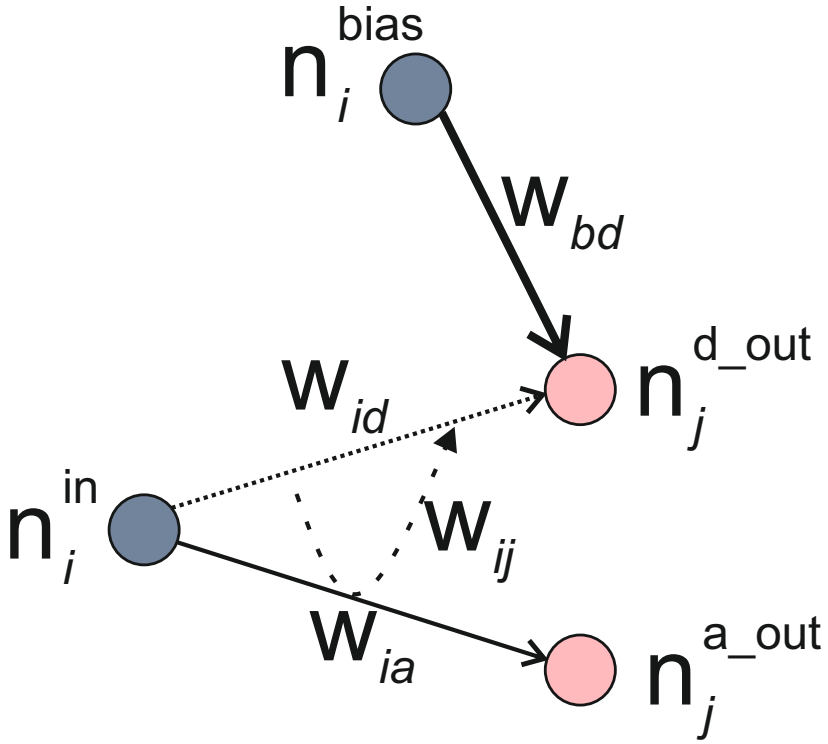
8.4 Architecture for Proposed Topology

Existing simulators, especially the one we develop using Brian (see subsection 2.7.1), do not support remote supervision for synapse models. Therefore, we present a structure that handles the ReSuMe paradigm. In this experiment, a new fully connected two layered feed-forward spiking neural network architecture without a hidden layer is proposed. The network structure is inspired by the actor-critic architecture (Potjans et al., 2009) illustrated in Figure 7.5 and Figure 7.7.

Although the ReSuMe mechanism is inspired from heterosynaptic plasticity, we propose a new concept of remote supervision. Therefore, ReSuMe requires synaptic information between neurons i and k in order to adjust synaptic weight between neurons i and j illustrated in Figure 8.2a. However, many implementation environments, including that used in our experiments (see section 2.7) do not allow this. Hence, we express this plasticity in the concept of heterosynaptic plasticity without introducing remote supervision. The mechanism in Figure 8.2b is presented using a minimalist network scenario demonstrated in Figure 8.3. It is a more plausible way biologically because of the mechanism of heterosynaptic plasticity rather than remote supervision.



(a) Current topology for ReSuMe.



(b) Proposed topology for ReSuMe.

Figure 8.2: The comparison for pre-post neuron connections for the current and proposed structure. (a) The structure of neuron connection for ReSuMe learning with the heterosynaptic connection. This is adapted from Ponulak & Kasinski (2010). Desired neuron, input neuron and output neuron are shown n_k^d, n_i^{in} and n_j^{out}, respectively. The weight between input and output is w_{ij} which is affected from desired neuron activities as well.
(b) The heterosynaptic connection for the ReSuMe in (a) is modified here. The weight between input neuron n_i^{in} and actual output neuron $n_j^{a\_out}$ is w_{ia}. The weight between input neuron n_i^{in} and desired output neuron $n_j^{d\_out}$ is w_{id}. The total ReSuMe weight between input neuron n_i^{in} and output neuron is w_{ij}. w_{ij} is summed net weight change through n_i^{in} and n_j^{out} as in (b), summarized in Equation 8.8. This summation illustrated with dashed arrow is applied at the end of each simulation period/presentation. Bias neuron n_i^{bias} detailed in section 7.5.3 is connected to the desired output neuron $n_j^{d\_out}$ with fixed weight w_{bd} in order to generate desired spiking activities at $n_j^{d\_out}$.

Anti-Hebbian Spike-timing Dependent Plasticity (anti-Hebbian-STDP) process between input neurons and the actual output neuron can modify synaptic connections. However, Hebbian-STDP (Hebbian Spike-timing Dependent Plasticity) process between input and desired output neuron cannot modify the actual synaptic dynamics drawn as dashed lines in Figure 8.2. Initially, all synaptic weights, except the fixed connection from the bias synapse (see section 7.5.3), are set sufficiently small between w_{min}^{init} and w_{max}^{init} (see Table 8.4). Inputs do not cause any spikes in the early stages of training. Once weights are increased during later stages of training, extra spikes are generated inside the desired spike patterns which is not acceptable.

In order to solve this problem, the dynamics of connections are reconfigured. Generated spikes at the desired output should only be caused from the bias input not from input neurons. For that reason, once training is performed, weight updates for those Hebbian-STDP synapses are not applied unlike anti-Hebbian-STDP synapses. All STDP synapses into the desired output neuron have minimal weight values the same as their initial strengths. Hebbian-STDP updates are only used to find the difference between anti-Hebbian-STDP in ReSuMe technique. If we do not restrict Hebbian-STDP synapses with this limitation, they will generate extra spikes in the desired output, misguiding the training sessions especially near to the end of training.

Figure 8.2 demonstrates how proposed topology in Figure 8.3 mimics ReSuMe weight change through two synapses from input to actual output neuron and desired output neuron. The aim in the proposed structure is to divide w_{ij} (the weight between input neuron (n_i^{in}) and output neuron) into two different synapses as w_{ia} (the weight between input neuron n_i^{in} and actual output neuron $n_j^{a\_out}$) and w_{id} (the weight between input neuron n_i^{in} and desired output neuron $n_j^{d\_out}$).

Minimalist network scenario for the proposed topology is displayed in Figure 8.3. In the simplest scenario, there are 4 neurons. Three of them n_i^{in}, $n_j^{a\_out}$ and $n_j^{d\_out}$, are fundamental ones in order to demonstrate the concept properly. Bias neuron n_i^{bias} allows network to generate the desired output with the correct timing (see also section 7.5.3). Considering that simulations are flowing from input layer to output layer (left to right), if the desired spike train S_d is connected directly $n_j^{d\_out}$ rather than connecting to n_i^{bias}, the desired spike pattern is always trained a simulation time step, t_{dt}, earlier which is not adequate. The weight between n_i^{bias} and $n_j^{d\_out}$ is w_{bd} which is fixed and maximum weighted value (see Table 3.1) in order

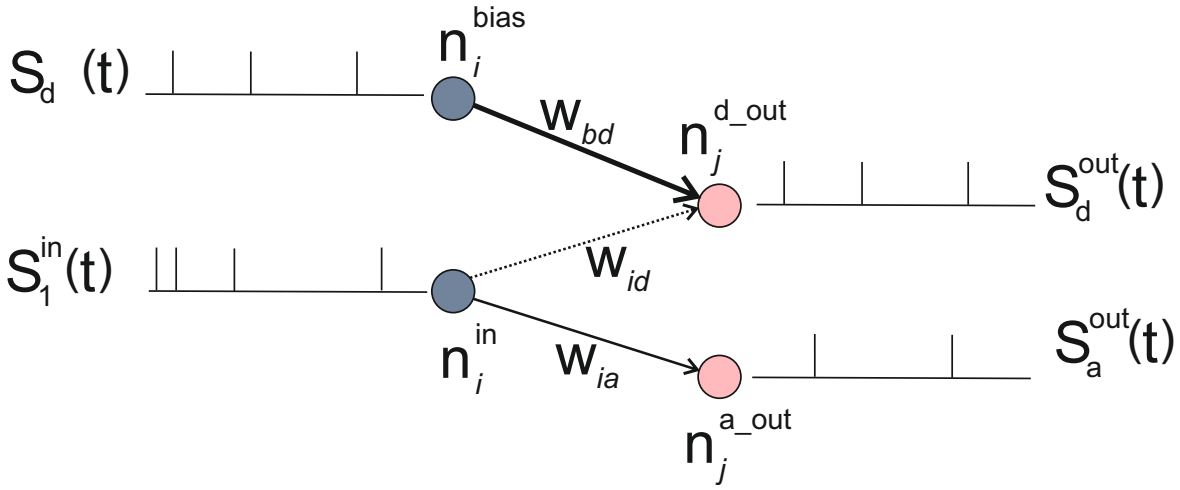


Figure 8.3: Minimalist network scenario for the proposed topology is displayed here. See the text for detailed description.

to transfer entire pre-synaptic activity S_d to the desired output activity S_d^{out}. The weight between input neuron (n_i^{in}) and actual output neuron ($n_j^{a\_out}$) is w_{ia}. The weight between input neuron (n_i^{in}) and desired output neuron ($n_j^{d\_out}$) is w_{id}. The total ReSuMe weight between input neuron (n_i^{in}) and output neuron is w_{ij}. Spike patterns for input, desired output and actual output are shown with S_1^{in}, S_d^{out} and S_a^{out}, respectively. S_d prepared externally and S_d^{out} generated from $n_j^{d\_out}$ are the same patterns with S_d in terms of spiking activities.

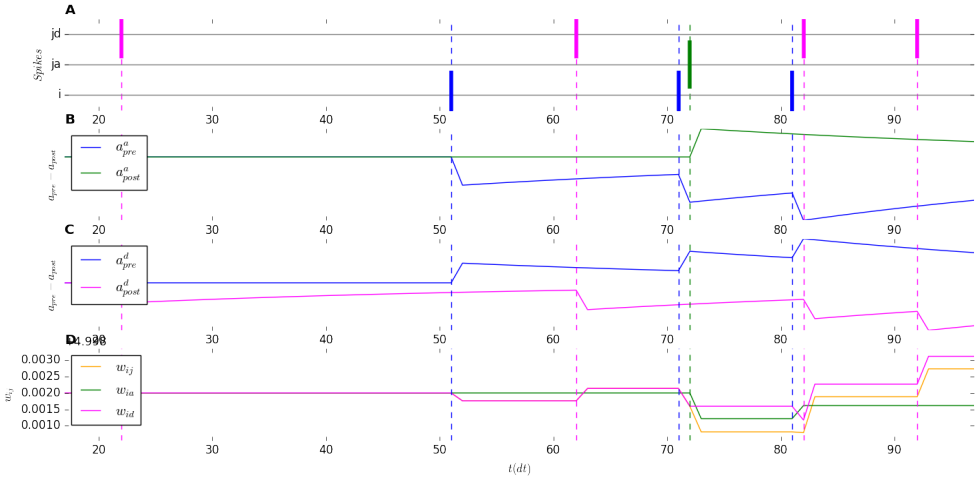
In Figure 8.4, the weight between input neuron (n_i^{in}) and actual output neuron ($n_j^{a\_out}$), the weight between input neuron (n_i^{in}) and desired output neuron ($n_j^{d\_out}$), the total ReSuMe weight between input neuron (n_i^{in}) and output neuron (same as w_{ij} in Figure 8.2) are w_{ia}, w_{id} and w_{ij}, respectively.

In Figure 8.4, a_{pre}^{a} and a_{post}^{a} are traces for anti-Hebbian learning; a_{pre}^{d} and a_{post}^{d} are traces for hebbian learning. a_{pre}^{a} and a_{pre}^{d} follow spiking activity from the input neuron with opposite sides. a_{post}^{a} and a_{post}^{d} follow actual and desired neuron spiking activities respectively.

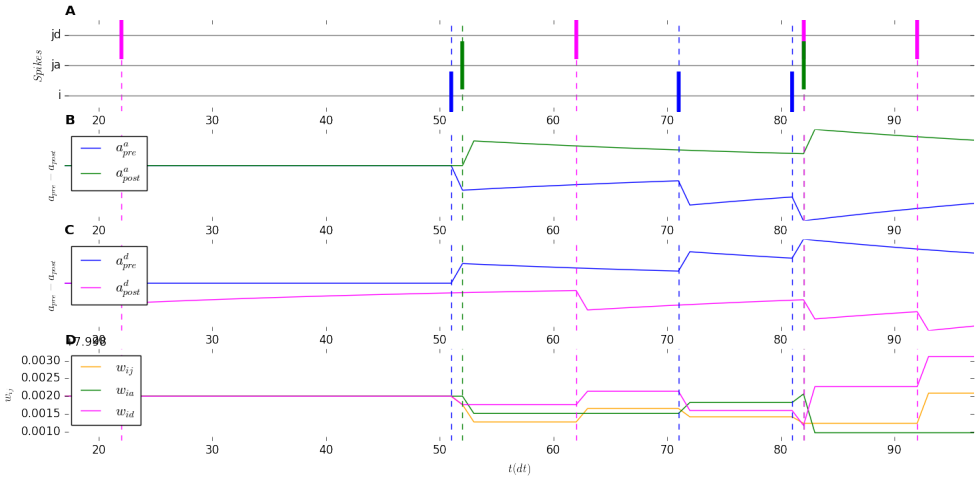
In Figure 8.4a, all weights w_{ia}, w_{id} and w_{ij} are the same at the beginning of each simulation period (see Equation 8.7).

$$w_{id} = w_{ia} = w_{ij} \rightarrow w_{ij}^{0} \tag{8.7}$$

where w_{ij}^{0} is the ReSuMe synaptic weight at the beginning of each learning cycle.



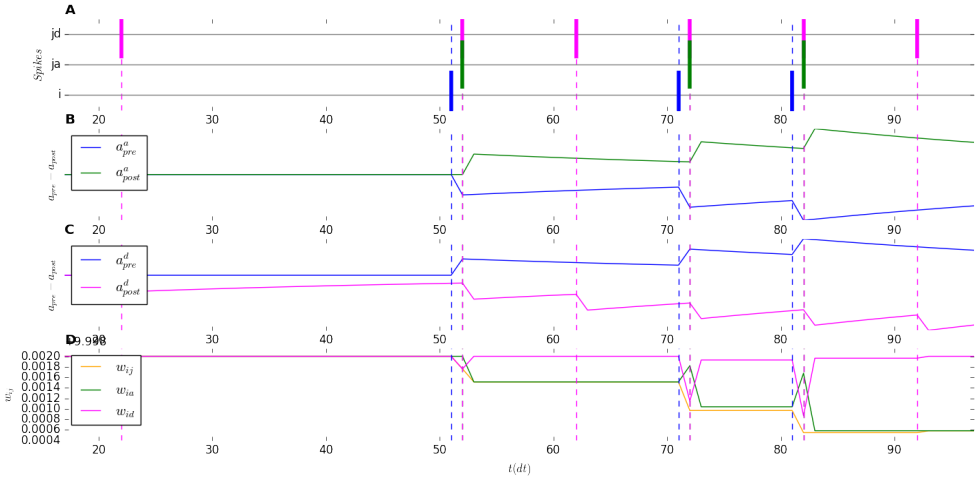
(a) Simulation scenario with single spike in desired pattern.



(b) Simulation scenario with two spikes in desired pattern.

Figure 8.4: Simulation scenario with different number of spikes in desired pattern.

In Figure 8.4a, the first weight change happens between input and desired output neuron (around 52. ms) w_{id} in the proposed simulation scenario. w_{id} follows the total ReSuMe weight w_{ij} (coloured orange line) because there are no weight change in w_{ia} so far. The next weight change is again because of input-desired output pairs (around 63. ms). w_{id} still follows w_{ij} exactly until the change on w_{ia} around 72 ms. At this point, Δw_{ia} is modified because of input-actual output spiking activity. At the end of each presentation, the main weight update rule is applied as summarized in Equation 8.8. Weight update procedures per presentation with three different actual neuron activities are illustrated in Figure 8.4a, Figure 8.4a, and Figure 8.4c.



(c) Simulation scenario with three spikes in desired pattern.

Figure 8.4: Simulation scenario with different number of spikes in desired pattern (continued) for the minimalist connection shown in Figure 8.2. A) Three spiking activities as input spike pattern (row i as S_1^{in}), actual output spiking activity (row ja as S_a^{out}) and desired spike pattern (row jd as S_d^{out}) are displayed. Each of them is coloured differently. Also activity times during other patterns are also plotted on all patterns with vertical dashed lines. B) a^a_{pre} and a^a_{post} are traces for anti-hebbian learning between input and actual output neuron. C) a^d_{pre} and a^d_{post} are traces for hebbian learning between input and desired output neuron. D) The weight between input neuron (n_i^{in}) and actual output neuron ($n_j^{a\_out}$) is w_{ia} coloured with green line. The weight between input neuron (n_i^{in}) and desired output neuron ($n_j^{d\_out}$) is w_{id} coloured with magenta line. The total ReSuMe weight between input neuron (n_i^{in}) and output neuron is w_{ij} in the text and in Figure 8.2) coloured with orange line.

$$\begin{aligned} w_{ij} &\leftarrow w_{ij}^0 + \Delta w_{ij} \\ w_{ij} &\leftarrow w_{ij}^0 + \Delta w_{id} + \Delta w_{ia} \end{aligned} \tag{8.8}$$

where Δw_{ij} is splitted into Δw_{id} and Δw_{ia}.

8.5 Error Analysis

In order to validate the results we use the van Rossum Distance (vRD), described in 6.5.4 as D_R, in order to measure the distance between separate spike trains. The distance metric is not involved in learning; however, it reflects the dissimilarity between actual and desired spike trains during testing.

The distance error is named the van Rossum Error (vRE) throughout the thesis. We use averaged vRE because error-epoch trajectories are plotted averaging vRE through a number of experiments.

Parameter Type	Parameter Names	Values
vRD Decay	τ_R	10 ms
vRD Period	T_R	120 ms

Table 8.2: Model parameters used for error analysis.

We set the decay constant τ_R as a free parameter to $\tau_R = 10$ ms (see Table 8.2). The period of metric T_R in Equation 6.13 is the same as the period of a single run for an extended presentation time $T_R = T_{pe} = 120$ ms.

8.6 ReSuMe Overall Setup Experiments

Network architecture used in ReSuMe tasks is demonstrated in section 7.5. The types of neuron models used in proposed SNNs under the noiseless conditions are summarized in section 7.2. In addition, the training and testing stages of performed experiments are already described in subsection 7.8.1 and subsection 7.8.2, respectively. The delay mechanism from section 7.4 is used here. The number of synaptic connections is $N_{sub} = K = 10$ (see Table 7.1) with non-programmable delays between $1 - 10ms$, $d_{ij,k} = \text{k}$ (ms). However, the bias input neuron $n_{i=0}$ has a single connection without a delay, $d_{ij,k=0} = 0$ ms.

Parameter Type	Parameter Names	Values
Presentation Time	T_{pe}	120 ms
Run Time Resolution	dt	0.1 ms
Presentation Number	N_P	10
Epoch Number	N_E	1000

Table 8.3: Length parameters used for the computer simulations.

Each presentation runs for the simulated $T_p = 100$ ms duration of the input spike trains also with two times the maximal synaptic delay $2 * \tau_m = 20$ ms added as $T_{pe} = 120$ ms (see Table 8.3). Because of the discrete nature of spikes, the simulation of network dynamics is

performed on a time step of $dt = 0.1$ ms. We denote the number of epochs as N_E. Each epoch consists of presentations, the number of presentations during each epoch is denoted by N_P in Table 8.3. Total simulation time per session for each spike set can be calculated using Equation C.2. For instance, it is calculated as 20 mins with the parameters described in Table 8.3.

Parameter Type	Parameter Names	Values
Plasticity amplitudes	A_{pre}, A_{post}	0.005, 0.005
Decay constants	τ_{pre}, τ_{post}	5ms, 5ms
non-Hebbian Factor	a	0
Weight Limits	$[w_{min}^{TRN}, w_{max}^{TRN}]$	[-3, +3]
Initial Weight Limits	$[w_{min}^{init}, w_{max}^{init}]$	[-0.02, 0.08]
Excitatory/Inhibitory Rate	$[r_{exc}^{init}, r_{inh}^{init}]$	(0.2, 0.8)

Table 8.4: Model parameters used for the computer simulations through ReSuMe.

In order to avoid either silencing or extreme network activity by frequent firings of neurons, all synaptic weights are bounded with a lower boundary w_{min}^{TRN} and an upper boundary w_{max}^{TRN}. Because we use Additive-STDP updates instead of Multiplicative-STDP updates (see section 3.4.2 for details), current weight values are clipped to keep them in these predefined boundaries.

8.7 Experiments: Mapping for Remote Supervised Method

Table 7.1 gives the structural parameters for the experiment in section 7.5. The proposed structure for the SNN contains $N_{in} = M = 20$ neurons in the input layer (see Table 7.1). In the output layer, there are two neurons, $N_{out} = 2$, the one generates the actual output patterns $S_a^{out}(t)$. The other neuron produces desired spiking activities $S_d^{out}(t)$.

8.7.1 Results of Noiseless Simulations

An example of desired output signal $S_d^{out}(t)$ and the convergence of the actual output $S_a^{out}(t)$ over noiseless ReSuMe training cycles are depicted in Figure 8.5. The desired patterns have 3 spikes. On the early runs, there are less spiking activities in the actual output compared

to desired spiking activities because synaptic weights are initialized as sufficiently small at the beginning. Once the training continues, the generated spikes become closer to desired activity. After less than 90-100 learning cycles, most networks in various experiments illustrated in Figure 8.5 (see also subsection D.1.1 for other runs) produce the 3 desired spikes precisely. With this configuration, it can be clearly seen that the network reliably learn the target/desired pattern within 80 epochs, as shown in Figure 8.5. Zoomed version of the experiment can be seen in Figure 8.5. Further experimental results are also illustrated in Appendix D, respectively. Furthermore, the stability preservation of the network by keeping frequencies of neuronal activity within a certain range can be seen over the entire simulation.

Figure 8.6 demonstrates the evolution of synaptic weights for the noiseless ReSuMe mapping experiments in Figure 8.5. There are two experiments with different spike sets, each spike set in an experiment contains 5 spatio-temporal pattern sets. Other weight trajectories are not shown here because they have quite similar behaviours to the demonstrated ones. Upper two figures in Figure 8.6 are from one experiment and lower two figures are from another experiment. As it is detailed in section 8.9, each input-output connection pair has 10 synaptic terminals with fixed delays. Hence, left figures show one pair's weight trajectories of those input-output pairs. Right figures illustrate entire sub-terminals (20x10=200 here). It is clearly seen that weight trajectories do not achieve the minimum/maximum weight boundaries as in STDP. Because here Hebbian-STDP and anti-Hebbian-STDP balance each other once desired pattern is exactly the same as the actual spike generation. Hence, weights can stabilize in the intermediate values unlike single Hebbian-STDP or anti-Hebbian-STDP whose synaptic weights converge to maximum/minimum boundaries. Although the network generates desired output before 80 epochs, weights are completely stabilized later around 800 epochs seen in Figure 8.6.

Histogram of weights through noiseless conditions, prepared from the lower experiment in Figure 8.6, are illustrated in Figure 8.7. Four different characteristic time points throughout learning cycles are selected in order to show the evolution of synaptic weights. w_0 for 1^{st} presentation is the case of initial weights illustrated in Figure 8.7a. w_{199} for 200^{th} presentation is the case of after limited amount of learning epochs illustrated in Figure 8.7b. w_{599} for 600^{th} presentation is the case of after more amount of learning epochs illustrated in Figure 8.7c. w_{1999} for 2000^{th} presentation is the case of almost fully stabilized weights (end of simulation

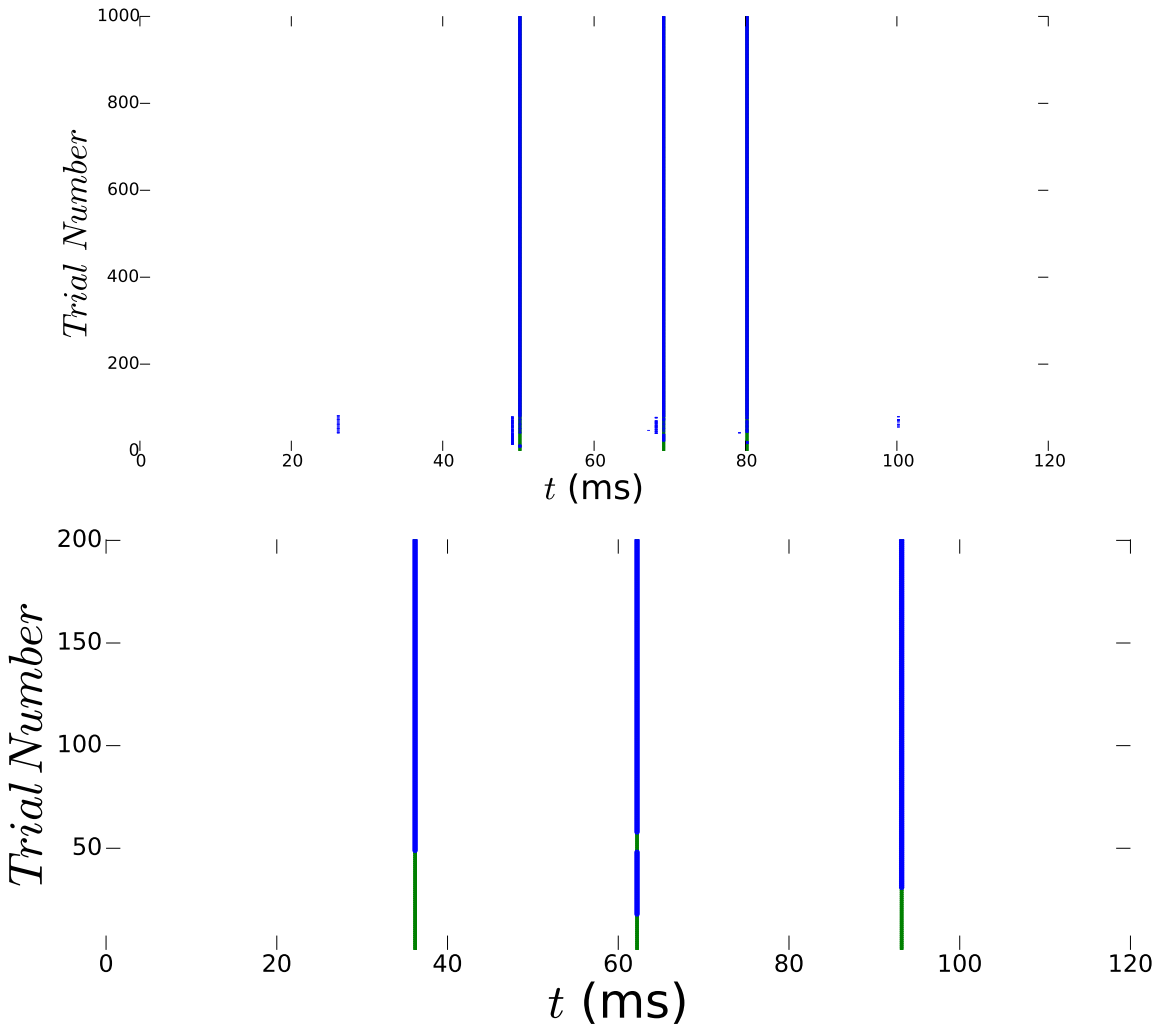


Figure 8.5: Reconstruction of the transformation from input patterns to output spike timings (noiseless condition). Trial number (y-axis) indicates the number of epoch N_E which consists of presentations N_P summarized in Table 9.1. The current network is trained to map a spatio-temporally encoded input spike trains into another spatio-temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour. Upper figure is from one experiment and below figure is from another experiment. More results can be seen in subsection D.1.1.

or close to end of simulation) illustrated in Figure 8.7d. While the learning is performed, the distribution covers the full range of available weights seen through selected time points: 1^{st}, 200^{th}, 600^{th} and 2000^{th} presentations. It is also clear that the range and distribution of the weights from 600^{th} to 2000^{th} presentation do not change significantly; because the stabilization of the learning is already achieved until the presentation of 600^{th} (see also Figure 8.5 and Figure 8.6). The unimodal steady-state distribution of synaptic weights is produced.

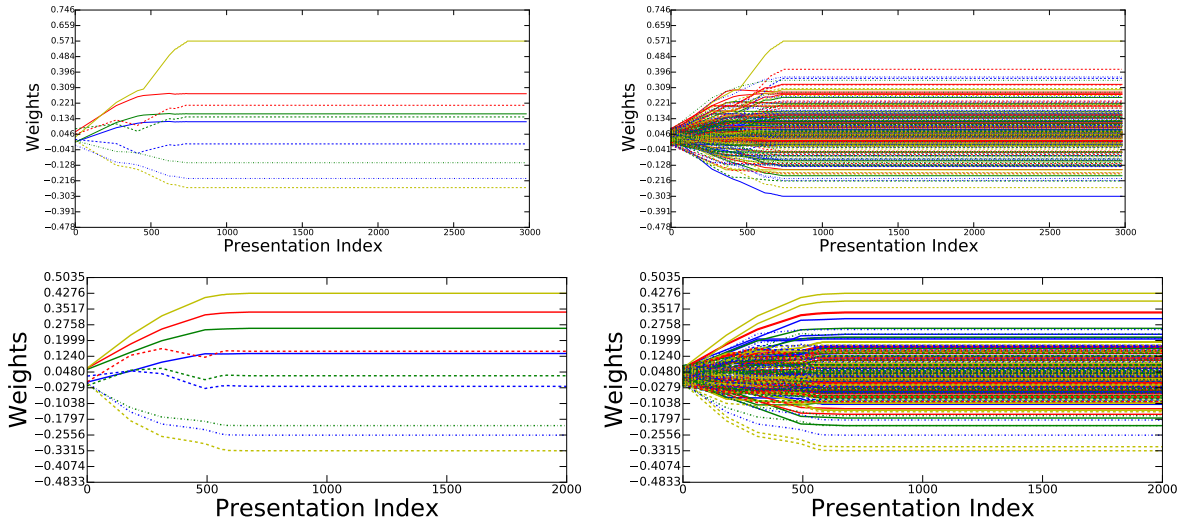


Figure 8.6: Evolution of synaptic weights for the mapping experiments (noiseless condition). In each plot individual weights are represented in a randomly selected colour. (Left) These show only neuron 0 to output neuron with 10 sub-connections during a whole training (2000 presentations). (Right) These demonstrate entire synaptic weight modifications during a whole training (2000 presentations). Upper two figures are from one experiment and lower two figures are from another experiment.

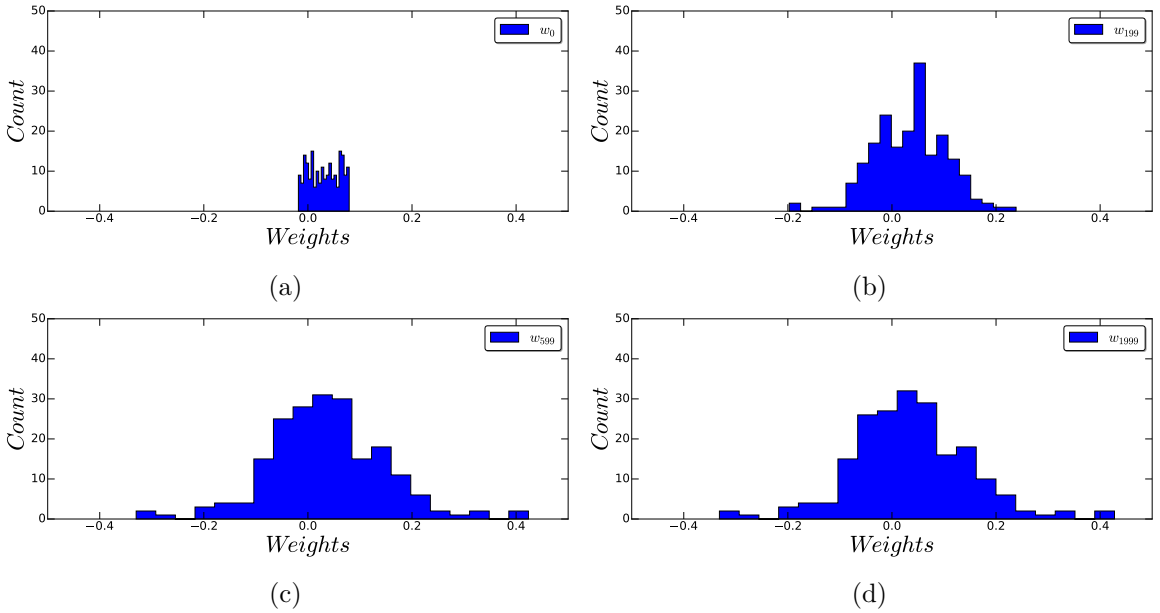


Figure 8.7: Histogram of weights is prepared from the lower experiment in Figure 8.6 (noiseless condition). Four different characteristic time points are shown in the figure with legend texts: w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{1999} for 2000^{th} presentation in (d).

In Figure 8.8, experiments are executed during 1000 epochs. There is no change or update after roughly 180 epochs even if the simulation is continued. This case is valid for all following experiments. There are 5 experiments working in parallel and independent. Each network

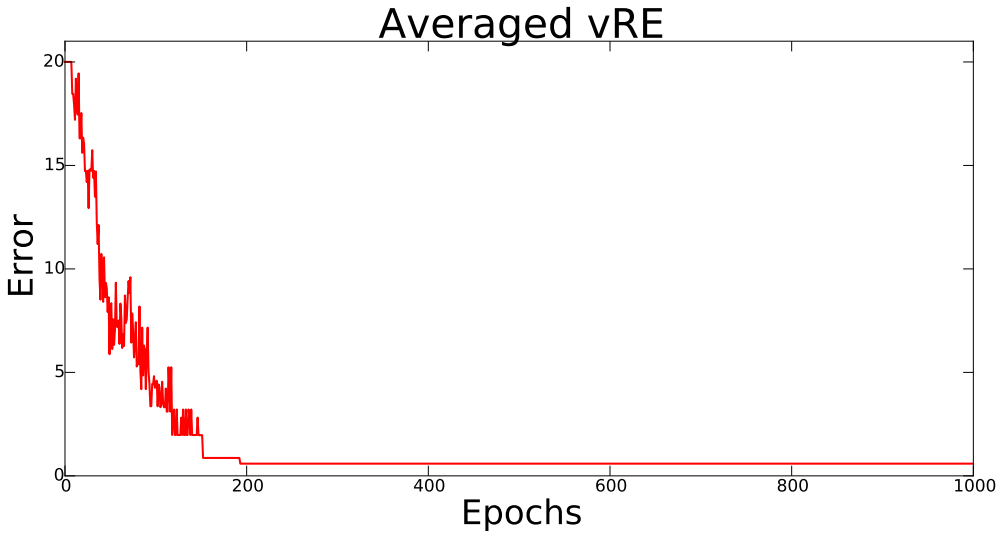


Figure 8.8: Averaged-vRE for mapping experiments. Epochs (x-axis) indicates the number of epoch N_E which consists of presentations N_P summarized in Table 9.1. See text for a detailed explanation of the figure.

in each experiment has different random weight initializations. In addition, spike sets for each experiment are also different and randomly selected from previously generated Poisson processes described in section 7.6. The figure demonstrates that averaged-vRE through 5 experiments starts with its maximum value as 20. Once training is performed, the error is converging to a minimum. As it is seen that it does not become zero although it converges. The reason is not entirely clear, but it might be related with the one of spike sets. One of the trajectories in Figure 8.5 has a spike at 20ms and is struggling to converge. It is possible that the network cannot increase the weights in 20ms. The silent period at desired output patterns can be slightly increased to cope with this issue.

In the following section, the simulations are also shown under noise to demonstrate any detrimental effect that the plasticity may have. This helps to verify the reliability and robustness of the plasticity mechanism as well.

8.7.2 Results of Noisy Simulations

In the previous experiment, we assume deterministic model of neurons with noise-free learning conditions. Under these assumptions the trained neuron can reliably generate target spike pattern whenever the corresponding inputs are presented. On the other hand, the reliability of the neural responses can be significantly impacted in the face of noise interference. Therefore,

the proposed architecture with the ReSuMe is investigated in the stochastic, noisy network in order to reliably generate target sequences of spikes. The details of the inserted noise and how they affect neuronal activity are detailed in section 7.3. The characteristics of low noise and high noise are depicted in Figure 7.1 and in Figure 7.2, respectively.

The precision and reliability of the target activity are tested in the presence of background noise during training sessions. The noise detailed in section 7.3 is simulated by a Gaussian white noise current injected to the neuron. The noise is included as additive term in synaptic input current. Two different noise levels are used as relatively high and low noise described in section 7.3.

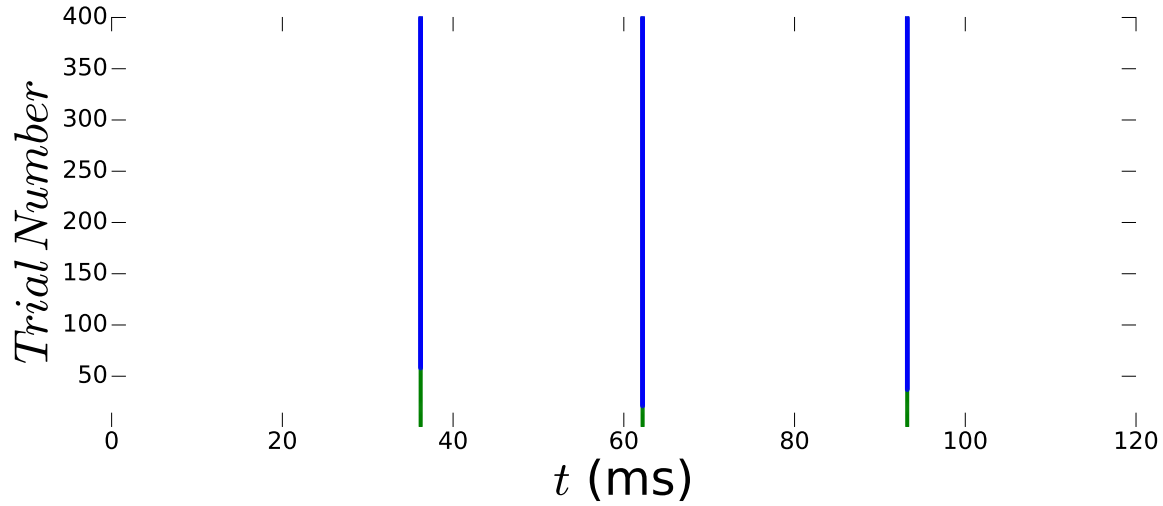


Figure 8.9: Reconstruction of the transformation from input patterns to output spike timings (low noise). Trial number (y-axis) indicates the number of epoch N_E which consists of presentations N_P summarized in Table 9.1. The current network is trained to map spatio-temporally encoded input spike trains into another spatio-temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour. Upper figure is from one experiment and below figure is from another experiment. More results can be seen in subsection D.1.1.

For the low noise, the actual and desired spiking activities through learning are illustrated in Figure 8.9. The evolution of synaptic weights is depicted in Figure 8.10. Also the histogram of weights through four different learning times is demonstrated in Figure 8.11.

For the high noise, the actual and desired spiking activities through learning is illustrated in Figure 8.12. The evolution of synaptic weights is depicted in Figure 8.13. Also the histogram of weights through four different learning times is demonstrated in Figure 8.14.

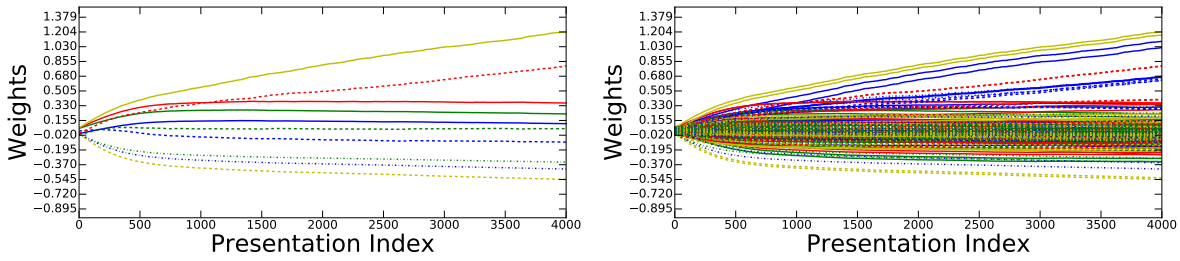


Figure 8.10: Evolution of synaptic weights for the mapping experiments (low noise). In each plot individual weights are represented in a randomly selected colour. (Left) These show only neuron 0 to output neuron with 10 sub-connections during a whole training (4000 presentations). (Right) These demonstrate entire synaptic weight modifications during a whole training (4000 presentations). Upper two figures are from one experiment and lower two figures are from another experiment.

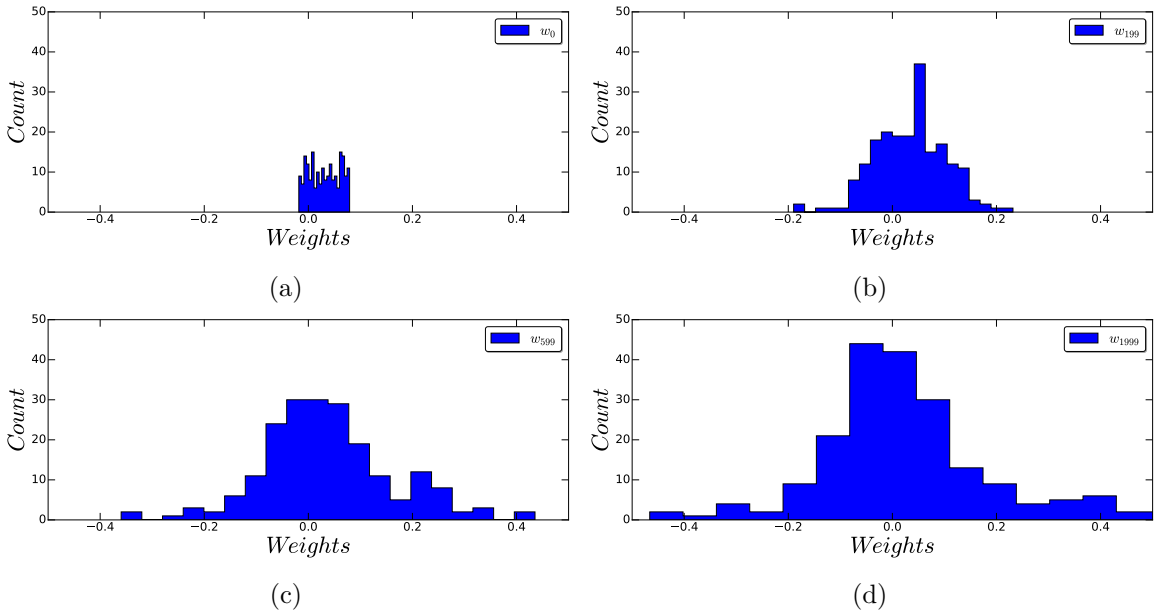


Figure 8.11: Histogram of weights through noisy conditions (low noise) (prepared from the lower experiment in Figure 8.6.) taken from four different characteristic time points shown with figure legend text : w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{1999} for 2000^{th} presentation in (d).

Inserting noise into the experiments cause shifting of actual spiking times (compared to desired spiking times) especially for the relatively high noise experiments. Although ReSuMe is robust to relatively low noise, it generates extra actual spike compared to desired activity in the further learning points. Also the precision of actual spiking activities gets worse while noise level is increased illustrated in Figure 8.9 and Figure 8.12. Also the speed of convergence for the learning gets worse once the level of noise is increased. For instance, three desired spikes are mimicked about 50^{th} epoch with noiseless conditions, about 70^{th} epoch with low

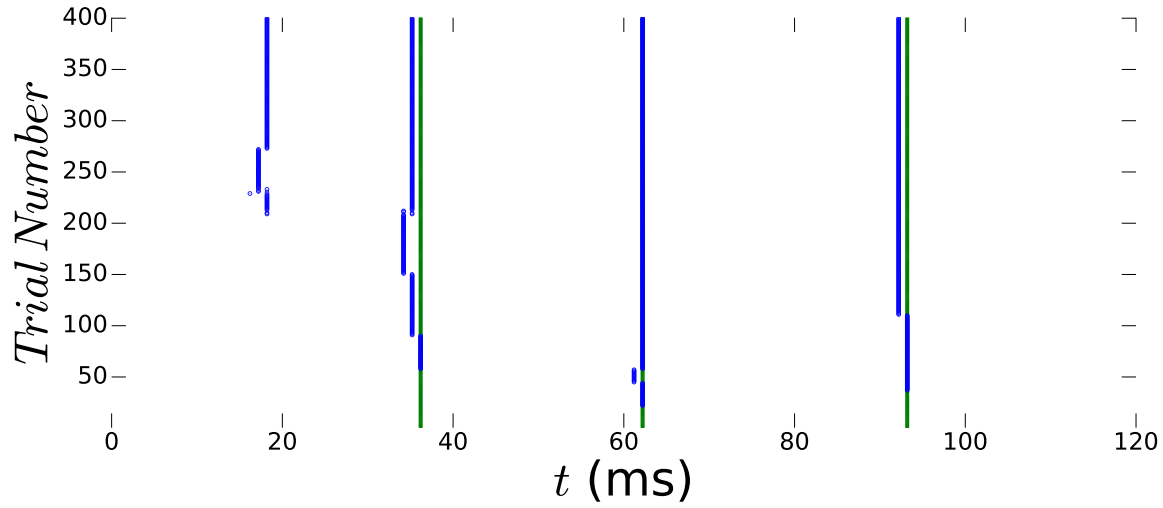


Figure 8.12: Reconstruction of the transformation from input patterns to output spike timings (high noise). Trial number (y-axis) indicates the number of epoch N_E which consists of presentations N_P summarized in Table 9.1. The current network is trained to map spatio-temporally encoded input spike trains into another spatio-temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour. Upper figure is from one experiment and below figure is from another experiment. More results can be seen in subsection D.1.1.

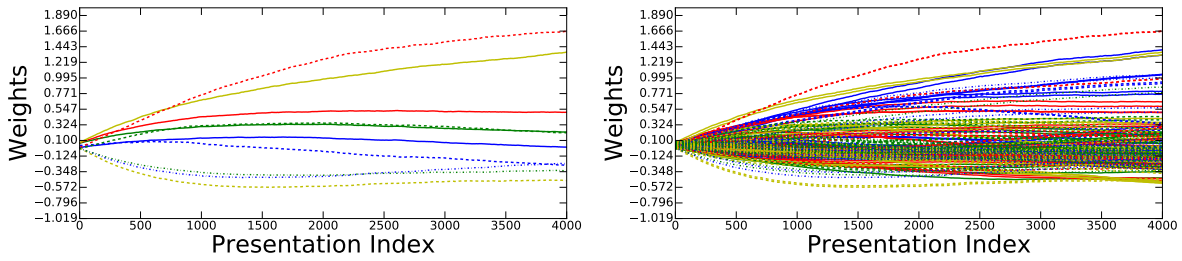


Figure 8.13: Evolution of synaptic weights for the mapping experiments (high noise). In each plot individual weights are represented in a randomly selected colour. (Left) These show only neuron 0 to output neuron with 10 sub-connections during a whole training (4000 presentations). (Right) These demonstrate entire synaptic weight modifications during a whole training (4000 presentations). Upper two figures are from one experiment and lower two figures are from another experiment.

noise conditions, about 140^{th} epoch with high noise conditions. The shape of synaptic weight distribution is the unimodal similar to noiseless conditions.

8.7.3 Discussion of Mapping

Each experiment is initialized with randomly generated weights. However, the performance of the algorithm is not affected from the initial conditions unlike SpikeProp. Therefore, we do not need to deal with optimizing the initial weights and the architecture of SNNs for this purpose.

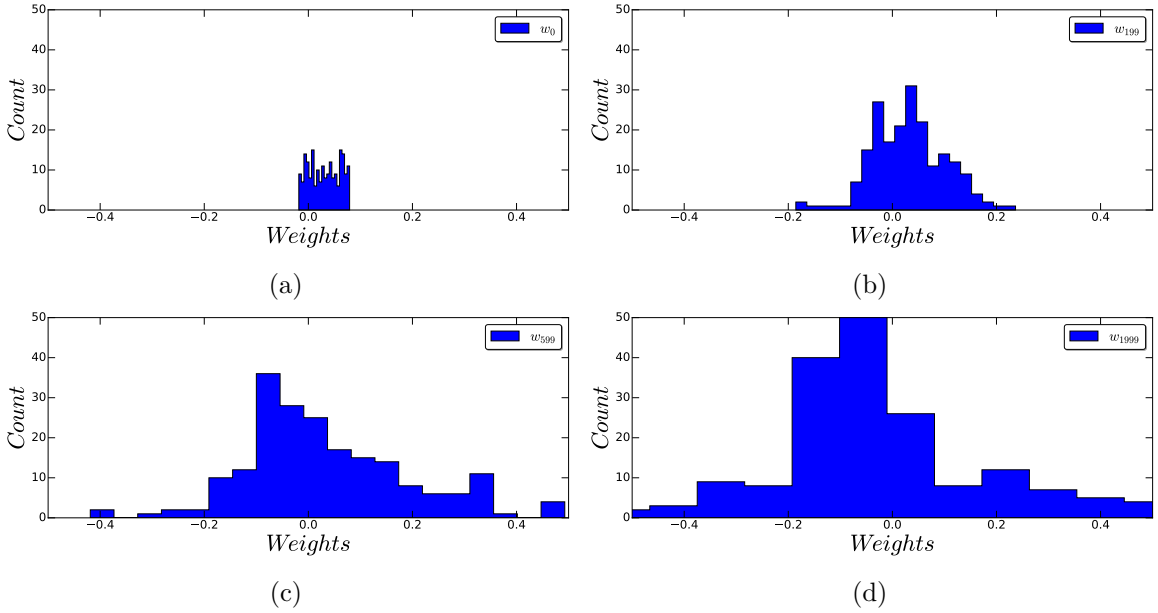


Figure 8.14: Histogram of weights through noisy conditions (high noise) (prepared from the lower experiment in Figure 8.13.) taken from four different characteristic time points shown with figure legend text : w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{1999} for 2000^{th} presentation in (d).

The task has been tested at least with 5 different spike sets for inputs and output. We have shown that input spike patterns can be successfully mapped into desired spike pattern at the output in less than 180 epochs. Entire network including all synaptic weights is stabilized around 500 epochs.

In the next section, the same infrastructure of ReSuMe is tested for more complex tasks than the mapping experiment.

8.8 Experiments: Logical Operations for Remote Supervised Method

In addition to single input-output pattern mapping in earlier sections, ReSuMe tasks are also performed for the logical operation benchmark introduced in subsection 8.8.1. Those tests are evidence for the proposed topology (introduced in section 8.4) which works in the case of multiple input-output patterns as well. Also, this benchmark can be the basis for future experiments in order to extend single input-output pattern mapping into multiple input-output patterns. Therefore, we test the infrastructure with a number of logical operations described in subsection 7.9.2.

8.8.1 Results of Noiseless Simulations

Averaged vRE trajectories during training can be seen in Figure 8.15 for operation TRUE, in Figure 8.16 for operation P1, in Figure 8.17 for operation AND, in Figure 8.18 for operation OR, and in Figure 8.19 for operation XOR. Each of those error trajectories starts from more than 20 and they converge to less than 3 except XOR task in Figure 8.19. Those graphs illustrate that each of tasks are learnt from the network except XOR.

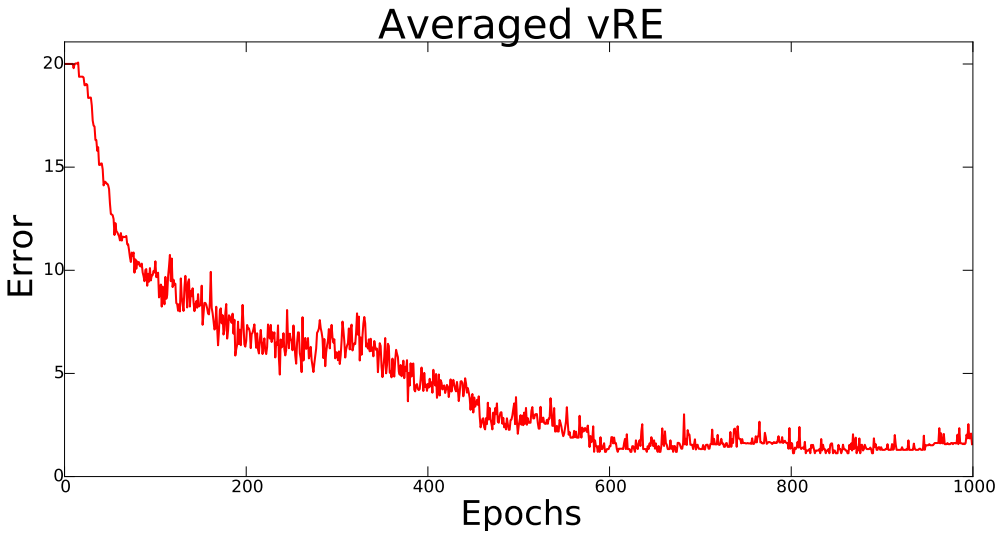


Figure 8.15: Averaged-vRE for operation TRUE.

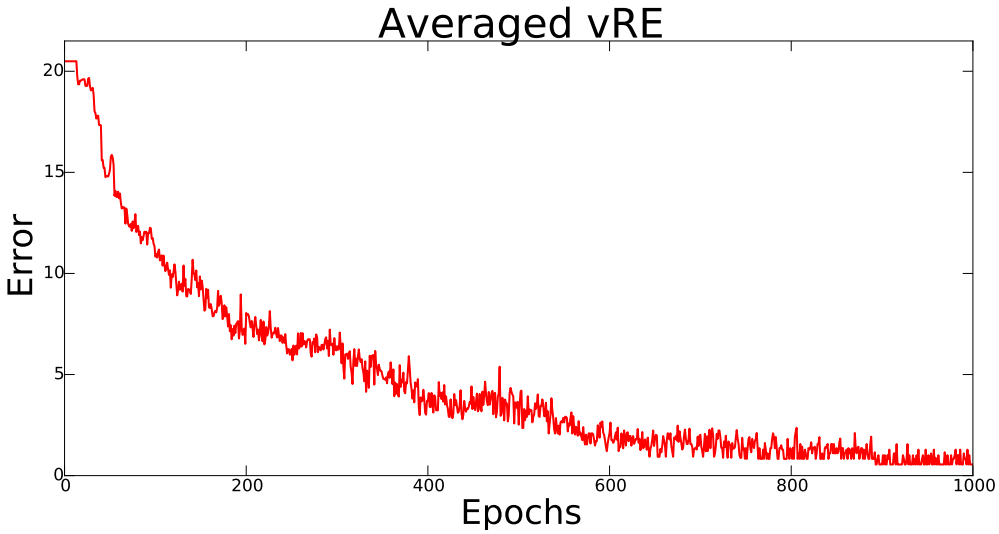


Figure 8.16: Averaged-vRE for operation P1.

The LCE trajectories of the SNN through different logical operations can be seen in Figure 8.20 for operation TRUE, in Figure 8.21 for operation P1, in Figure 8.22 for operation AND,

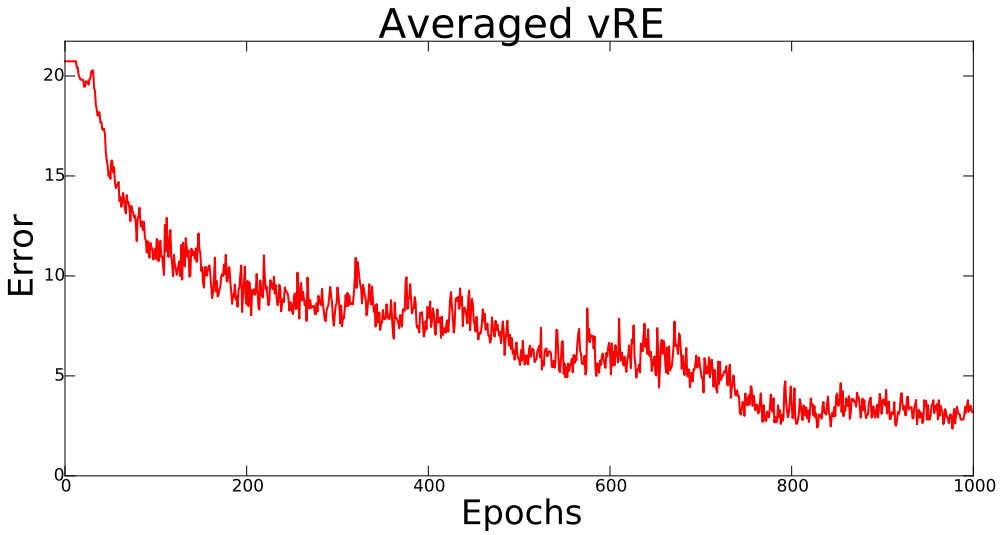


Figure 8.17: Averaged-vRE for operation AND.

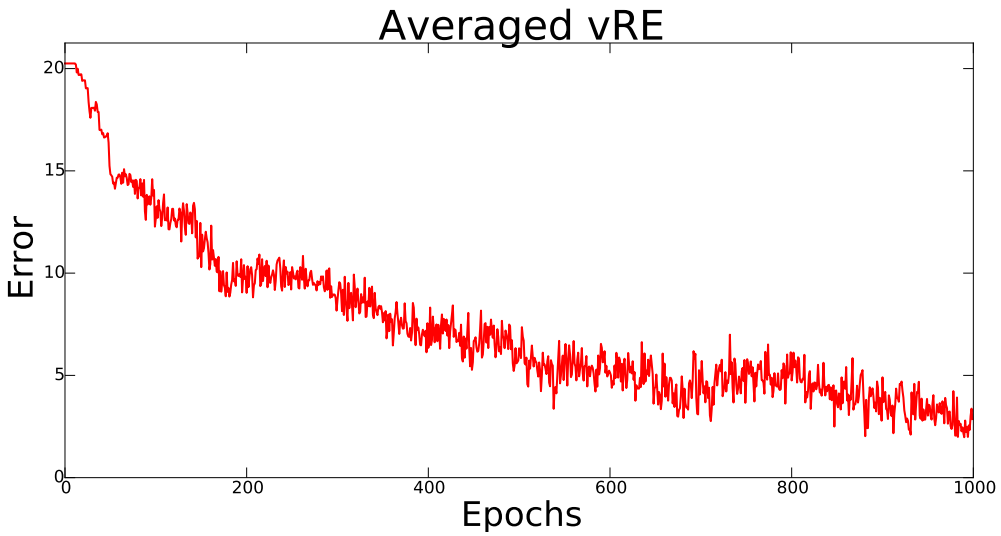


Figure 8.18: Averaged-vRE for operation OR.

in Figure 8.22 for operation OR, and in Figure 8.24 for operation XOR. Each error graph starts from roughly 50% which is random whether the output is TRUE or FALSE. Once the training is performed, Logic Classification Error approaches the minimum value through all logical tasks except XOR. Error graphs (LCE) do not look perfectly smooth. The resolution of error in graphs can be seen as 5% because each experiment is done using 5 spike sets. Also each training result is tested with all four possibilities (2-bit all possibilities for input: 0-0,0-1,1-0,1-1). Hence, during each epoch there are 20 (4x5) samples to part of LCE_P. This corresponds to the 5% resolution as percent. If the number of spike sets per experiment is increased, smoother graphs can be achieved.

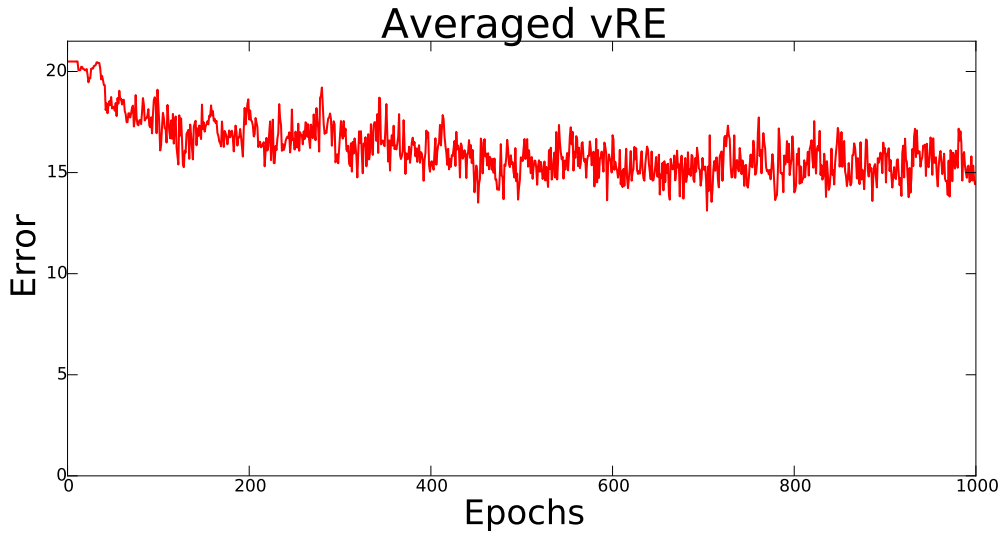


Figure 8.19: Averaged-vRE for operation XOR.

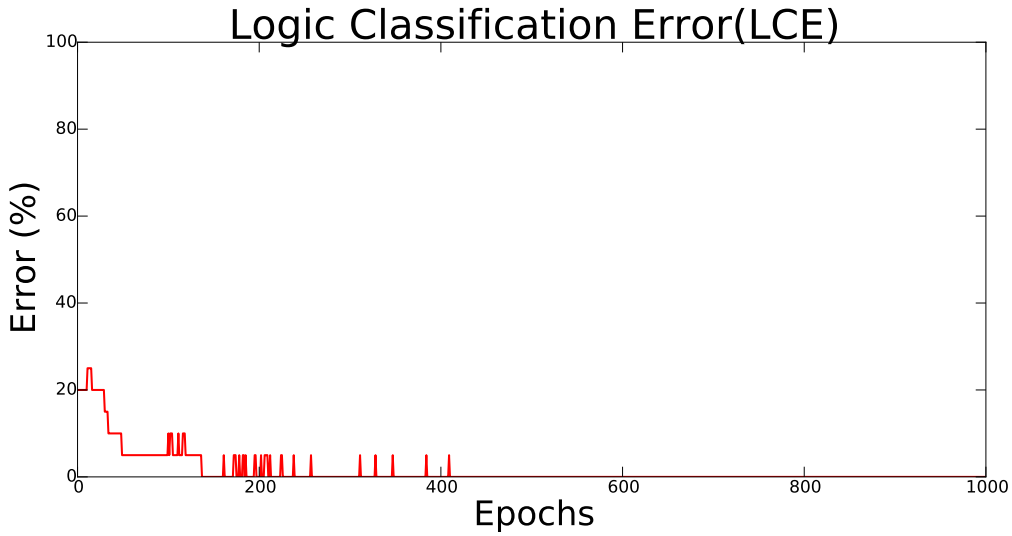


Figure 8.20: Logic Classification Error (LCE) for operation TRUE.

8.8.2 Discussion

As the mapping benchmark, each experiment has been initialized with randomly generated weights. Averaged performances of the algorithm through at least 5 different spike sets for inputs and output have been illustrated. We have shown that all logical operations except XOR can successfully be learned by the network. All networks are stabilized in less than 1000 epochs.

However, the 2-layer proposed network fails to learn XOR. In order to solve the XOR problem (truth table demonstrated in section A.5), we may need to introduce a new layer into our SNNs. This layer, called the "hidden layer", allows the network to create and maintain

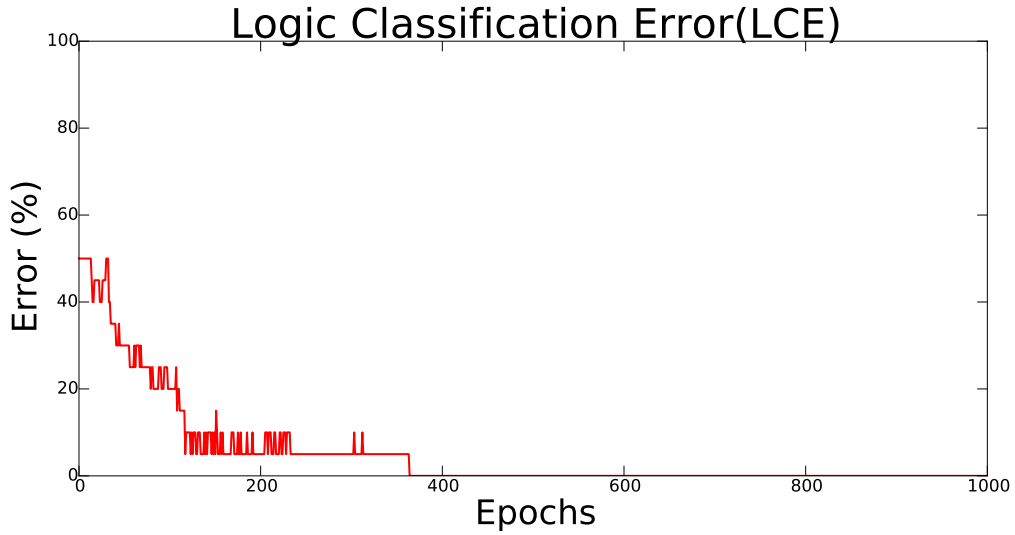


Figure 8.21: Logic Classification Error (LCE) for operation P1.

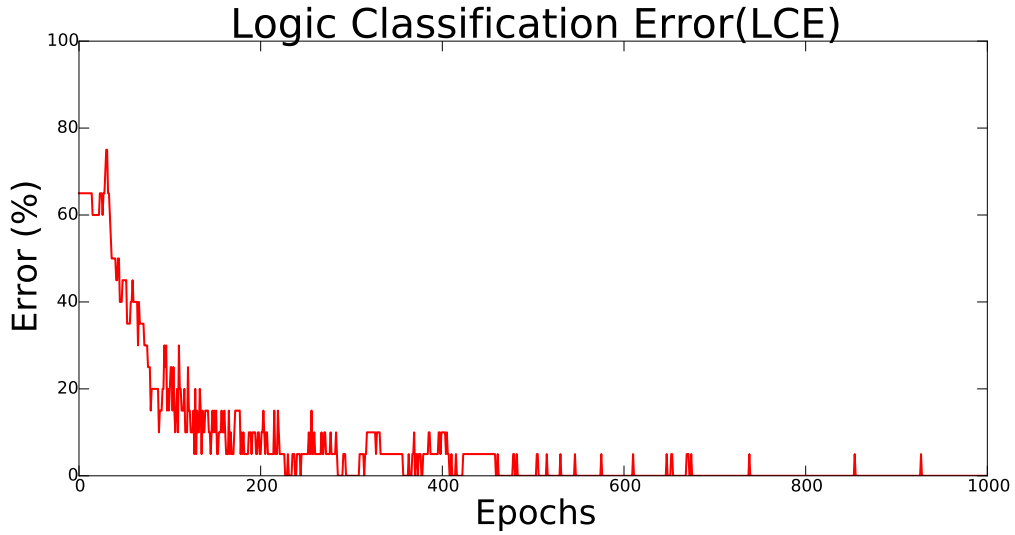


Figure 8.22: Logic Classification Error (LCE) for operation AND.

internal representations of the input. However, we have not tested this approach yet and it is appointed as a future work.

All ReSuMe results for both benchmarks are summarized in Table 8.5. In the next section, we extend the ReSuMe learning in order to have faster convergence.

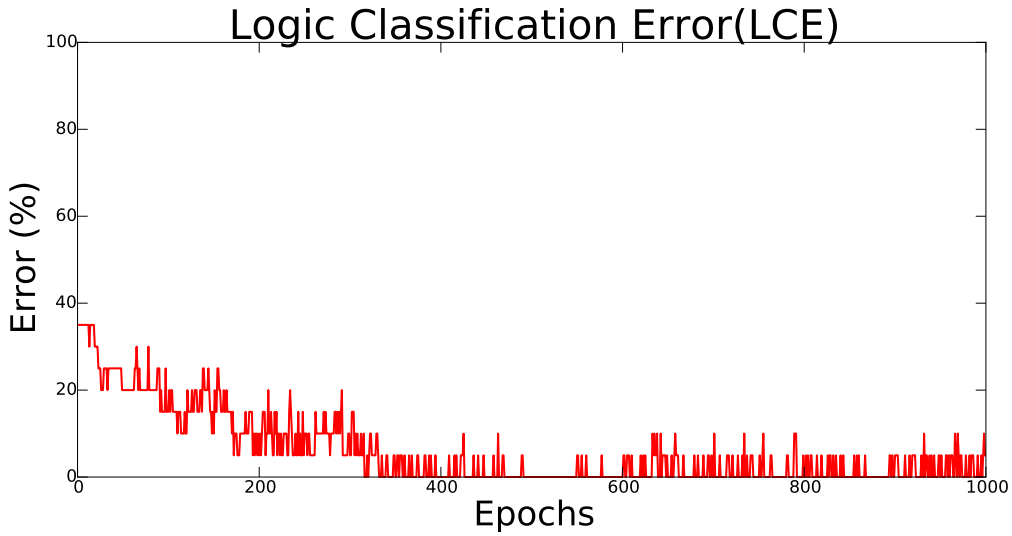


Figure 8.23: Logic Classification Error (LCE) for operation OR.

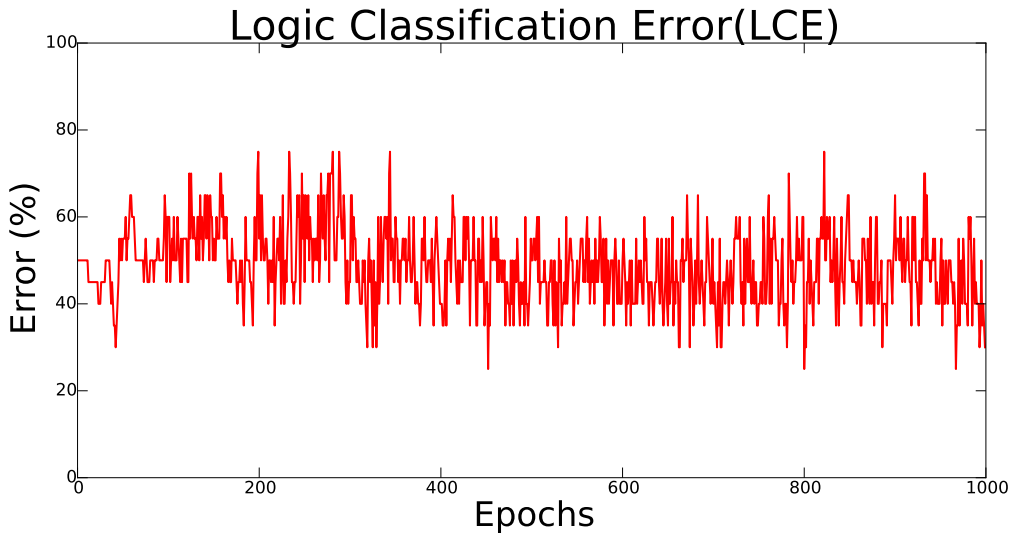


Figure 8.24: Logic Classification Error (LCE) for operation XOR.

Operation	Error Type	Epoch 400-499	Epoch 900-999
Mapping	vRE	0.5898561230	0.589856123412
	LCE	0.0000000000%	0.000000000000%
TRUE	vRE	3.6184665780	1.536079477980
	LCE	0.0505050505%	0.000000000000%
P1	vRE	3.6017840401	0.757597537969
	LCE	0.0000000000%	0.000000000000%
AND	vRE	7.6031646698	3.228147492520
	LCE	2.7777777778%	0.050505050500%
OR	vRE	6.8060406054	3.229104176230
	LCE	0.9090909091%	2.272727272730%
XOR	vRE	15.6070357980	15.459427742100
	LCE	46.8181818180%	47.070707070700%

Table 8.5: Summary of all operations through ReSuMe.

8.9 Delayed Remote Supervised Method

Delay adoption in addition to adapting synaptic weights in Spiking Neural Networks is considered in several researches (Natschlager & Ruf, 1998; Eurich et al., 2000; Adibi et al., 2005; Kerr et al., 2013). There are two methods to model computational delay: delay shift and delay selection (Eurich et al., 2000). In the delay shift approach, synaptic connection is considered as a single-connection with a single plastic delay (Eurich et al., 2000). In the delay selection approach, each pre-synaptic neuron is connected to the post-synaptic neuron with multi-plastic-delay (Natschlager & Ruf, 1998).

We propose an extended version of Remote Supervised Method, named Delayed Remote Supervised Method (DelReSuMe) using the delay selection approach. The proposed method modifies both synaptic weights and delays. Although the topology of ReSuMe in section 8.3 has fixed multi-delay mechanism, DelReSuMe architecture has plastic multi-delay architecture detailed in section 7.4. Except the plastic delays, the structure of the network is the same as for the ReSuMe experiments in section 7.5. The weight adjustment procedure of DelReSuMe is also the same as ReSuMe combining Hebbian-STDP in Equation 8.4 and anti-Hebbian-STDP in Equation 8.5.

By adding delay plasticity here, our motivation is to achieve faster convergence compared with ReSuMe. For this purpose, we add another exponential trace mechanism for each synaptic delay in addition to the weight traces in ReSuMe, and we name the trace as $c(t)$ over time t:

$$c(t) = \begin{cases} Ae^{-\frac{\Delta t}{\tau_c}} & \text{for } \Delta t \geq 0, \\ 0 & \text{for } \Delta t < 0. \end{cases} \tag{8.9}$$

where t is the current time, and t_i is firing time of pre-synaptic neuron i illustrated in Figure 8.25 with $\Delta t = t - t_i$. τ_c is the decay constant of $c(t)$ and A is the initial amplitude of the trace.

We extend the Equation 8.9 for the synapses which have their internal delays and reshape it

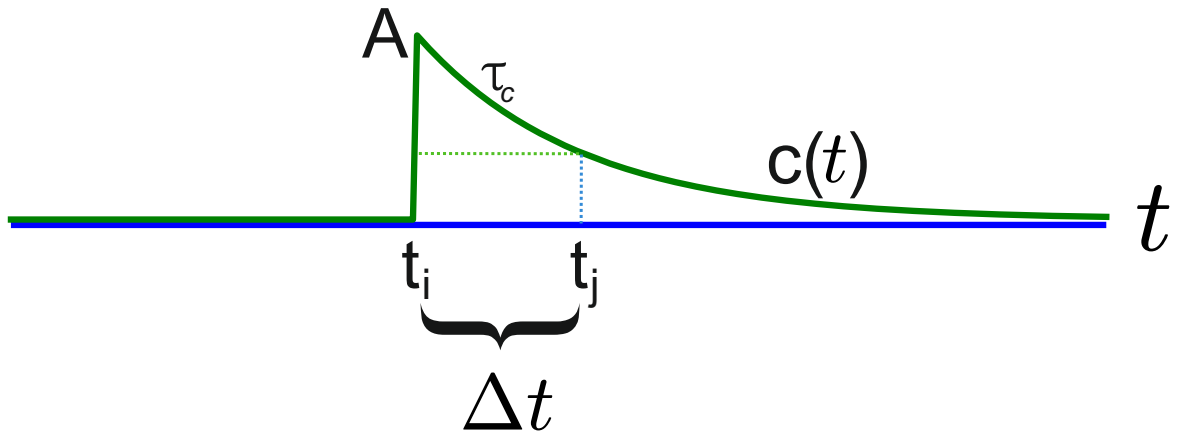


Figure 8.25: An exponential trace for synapse k with its internal delay in DelReSuMe. t is the current time, and t_i is the firing time of pre-synaptic neuron i, τ_c is the decay constant of $c(t)$ and A is the amplitude of trace. In this figure the trace $c(t)$ is evaluated at time $t = t_j$. $\Delta t = t - t_i$ with $t = t_j$.

by the activation time during only pre-synaptic firing times. It is formulated as:

$$c(t) = \begin{cases} Ae^{-\frac{t-t_i-d_{ij,k}}{\tau_c}} & \text{for } t_i^{(f_n)} < t < t_i^{(f_{n+1})}, \\ 0 & \text{otherwise.} \end{cases} \tag{8.10}$$

where t is the current time, and t_i is the firing time of pre-synaptic neuron i illustrated in Figure 8.26. $t_i^{(f_n)}, t_i^{(f_{n+1})}$ are pre-synaptic firing times. τ_c is the decay constant of synaptic delay trace $c(t)$. A is the initial amplitude increase. $d_{ij,k}$ is the delay from k^{th} synaptic terminal between pre-synaptic neuron i and post-synaptic neuron j.

Suppose that $t = t_j^a$ or $t = t_j^d$ which are the times of actual post-synaptic spikes or desired post-synaptic spikes, respectively. Using the delay trace $c(t)$, we can determine the adjustment to the delay between the active neurons. Using the inverse of Equation 8.9 to solve for Δt:

$$\begin{aligned} c_i(t) &= Ae^{-\frac{\Delta t}{\tau_c}} \\ ln(c_i(t)) &= lnA + lne^{-\frac{\Delta t}{\tau_c}} \\ ln\tfrac{c_i(t)}{A} &= -\tfrac{\Delta t}{\tau_c} \\ \Delta t &= -\tau_c ln\tfrac{c_i(t)}{A} \end{aligned} \tag{8.11}$$

where $\Delta t = t - t_i = t_j - t_{i,k}^f$. Δt is the time difference between current time t (active only post synaptic firing times t_j) and pre-synaptic spike time t_i (for k^{th} synaptic terminal, we use $t_{i,k}^f$).

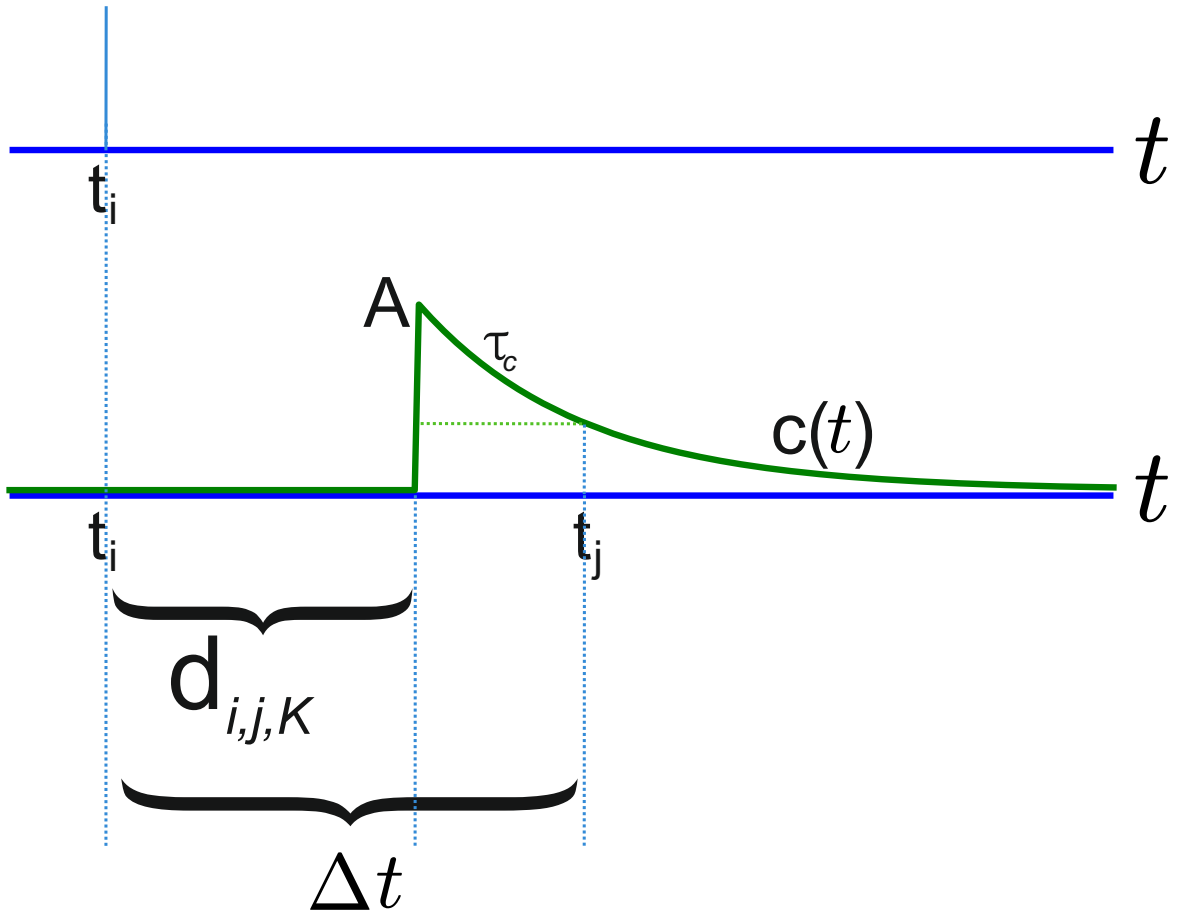


Figure 8.26: An exponential trace for each synaptic delay in DelReSuMe. t is the current time, and t_i is the firing time of pre-synaptic neuron i, τ_c is the decay constant of $c(t)$ and A is the amplitude of trace. $d_{ij,k}$ is the delay from k^{th} synaptic terminal between pre-synaptic neuron i and post-synaptic neuron j. Once $t = t_j$, the DelReSuMe is active. t_j can be either actual post-synaptic firing time or desired post-synaptic firing time. $\Delta t = t - t_i$ with $t = t_j$.

Then if we apply it to synapses with its own delay $\Delta d_{ij,k} = t - t^f_{i,k} - d_{ij,k}$, we get:

$$\Delta d_{ij,k} = t - t^f_{i,k} - d_{ij,k} \quad = -\tau_c ln \tfrac{c_i(t)}{A} \tag{8.12}$$

where $t^f_{i,k}$ is f^{th} firing time through synapse k from pre-synaptic neuron i.

The delay of synaptic terminal k is updated with:

$$\Delta d_{ij,k}(t) = \begin{cases} +sgn(w_{ijk})\Delta d_{ij,k}, & t = t^d_j, \\ -sgn(w_{ijk})\Delta d_{ij,k}, & t = t^a_j, \\ 0, \qquad otherwise. \end{cases} \tag{8.13}$$

where sgn is the signum function defined in A.5. t^d_j, t^a_j are desired and actual firing times of

post-synaptic neuron j, respectively. $\Delta d_{ij,k}(t)$ is the delay change for the k^{th} synaptic terminal between pre-synaptic neuron i and post-synaptic neuron j. w_{ijk} is the synaptic weight for the k^{th} synaptic terminal between pre-synaptic neuron i and post-synaptic neuron j at the time t. The motivation of including the signum function is to differentiate excitatory and inhibitory synapses. The effect is to shift synaptic delays in opposite directions for excitatory synapses, and the same direction for inhibitory synapses. Excitatory and inhibitory synapses have positive and negative weights, respectively.

The idea of integrating plastic synaptic delay into ReSuMe to improve the performance of learning is also considered in Taherkhani et al. (2015). However, there are several differences compared with our method. Firstly, a single-connection architecture using the delay shift approach rather than multi-plastic-delay is considered in their work. Our structure can be classified under the delay selection approach rather than the delay shift approach. On the other hand, the way to apply delay plasticity in the proposed learning rule is also dependent on the type of synapses whether they are inhibitory or excitatory in our method. This is handled with *sgn* function instead of separating two update rules for inhibitory and excitatory synapses. In addition, the presence of noise is not considered in Taherkhani et al. (2015). However, we consider two different noise levels: relatively low and relatively high noise described in section 7.3.

8.10 Experiments: Mapping for Delayed Remote Supervised Method

The parameters and network details including neuron models, neuron numbers, encoding mechanism, initialization methods except delay mechanism are the same as ReSuMe mapping experiment described in section 8.3 and section 8.7. Therefore, only delay mechanism is detailed here.

The value of minimum and maximum delays are not changed from ReSuMe experiments. However, synapses do not have fixed delays here unlike ReSuMe. All delays in DelReSuMe are initialized between of d_{min} and d_{max} same as ReSuMe. Furthermore, each synaptic delay is restricted to adjust only once in a single training in order to avoid instability of the network using Equation 8.13. Also, only a single synaptic delay can be adjusted per learning cycle.

The one that is eligible to adjust has nearest spiking activity before current update time. Synaptic weights are also continuously modified during entire training. The modification of synaptic delays once a training can also be interpreted as a proper initialization of synaptic delays for the task.

8.10.1 Results of Noiseless Simulations

An example desired output signal $S_d^{out}(t)$ and the convergence of the actual output $S_a^{out}(t)$ over noiseless DelReSuMe training cycles are depicted in Figure 8.27. The desired patterns have 3 spikes the same as ReSuMe tasks in subsection 8.7.1. After less than 20 learning cycles the 3 desired spikes are produced precisely through exactly the same task with ReSuMe. The comparison can be seen more clearly in the zoomed version in Figure 8.28.

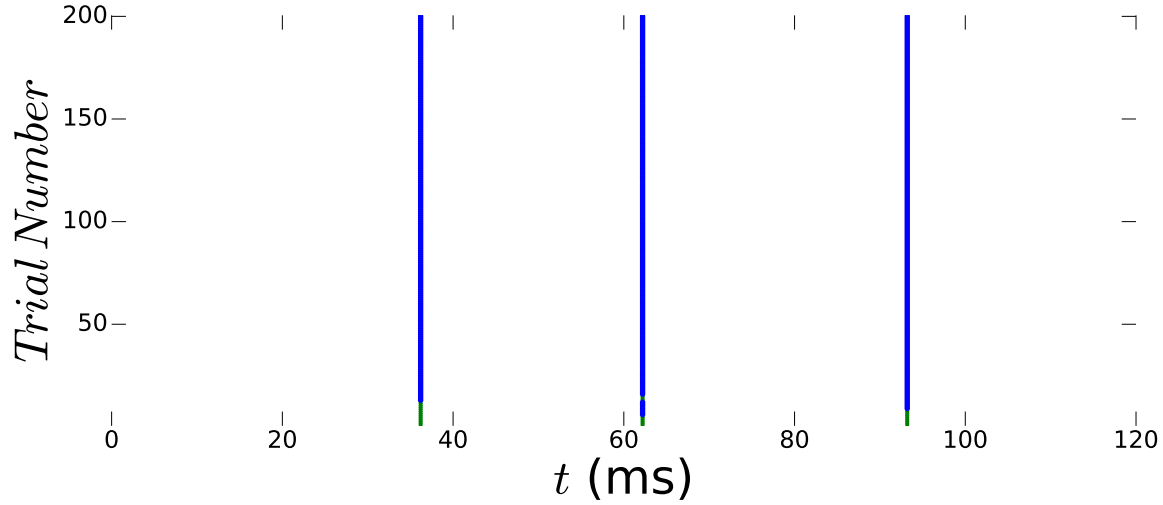


Figure 8.27: Reconstruction of the transformation from input patterns to output spike timings (noiseless condition). The current network is trained to map a spatio-temporally encoded input spike trains into another spatio-temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour.

Figure 8.29 demonstrates the evolution of synaptic weights for the noiseless DelReSuMe mapping experiments in Figure 8.27. Left figure shows single pair's weight trajectories of inputs-output. Right figure illustrates entire sub-terminals similar to subsection 8.7.1. Similar to ReSuMe experiments, it is clearly seen that weight trajectories do not achieve the minimum/maximum weight boundaries as in STDP. Once trained synaptic weights are stabilized in the intermediate values, the range of them during DelReSuMe is narrower than ReSuMe illustrated in Figure 8.6. In other words, DelReSuMe learning has smaller weight changes

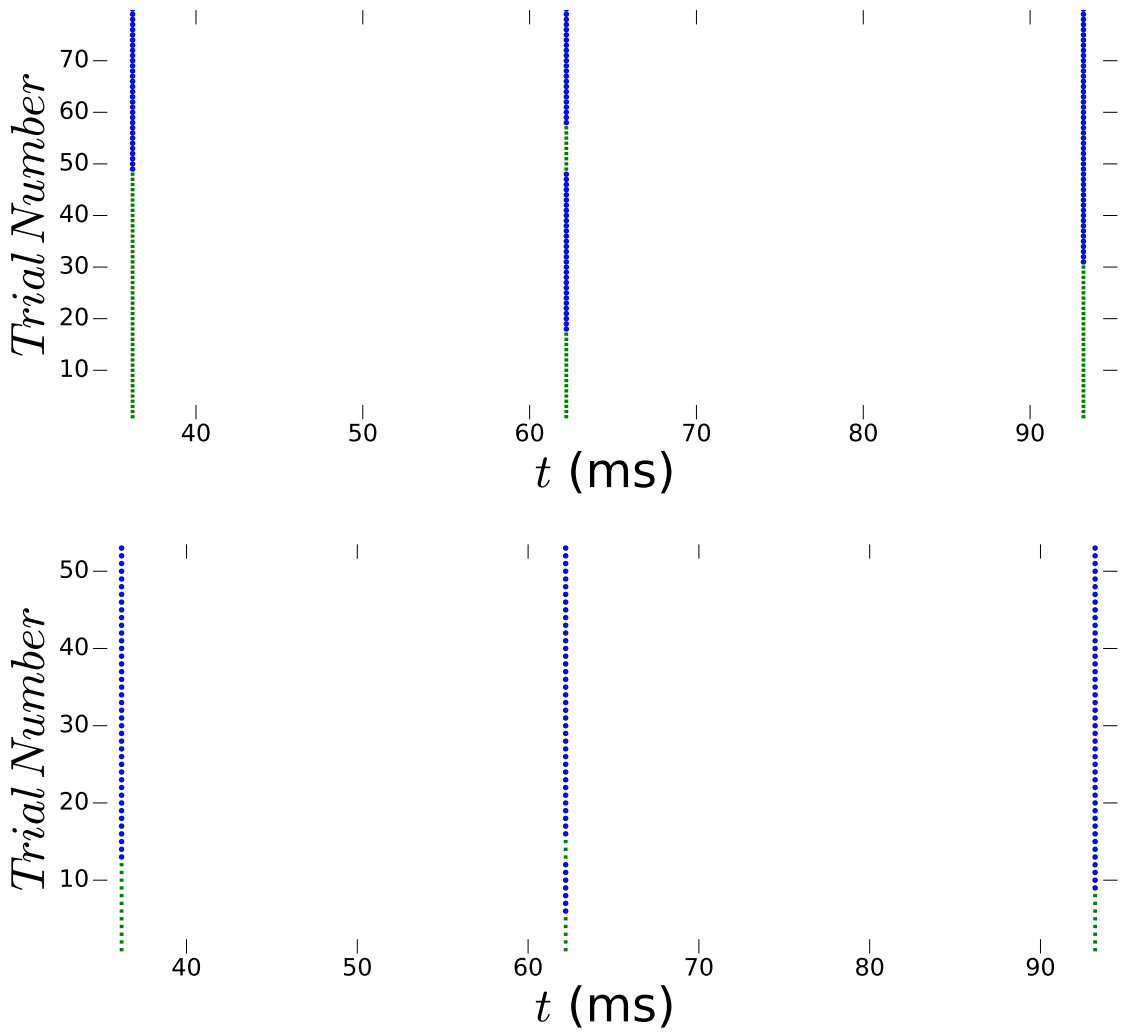


Figure 8.28: Reconstruction of the transformation from input patterns to output spike timings (noiseless condition). Zoomed version of Figure 8.5 and Figure 8.27. The current network is trained to map a spatio-temporally encoded input spike trains into another spatio-temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour.

than ReSuMe learning. Although the network generates desired output before 30 epochs, weights completely stabilize later around 200 epochs seen in Figure 8.29.

Histogram of weights through noiseless conditions, prepared from the bottom experiment in Figure 8.29, is illustrated in Figure 8.30. Similar to subsection 8.7.1, four different characteristic time points throughout learning are selected in order to show the evolution of synaptic weights. The timing of those points are the same as ReSuMe in order to illustrate comparisons clearly. w_0 for 1^{st} presentation is the case of initial weights illustrated in Figure 8.30a. w_{199} for 200^{th} presentation is the case of after limited amount of learning epochs in ReSuMe. For DelReSuMe illustrated in Figure 8.30b, it is almost the point that synaptic weights are

fully stabilized. w_{599} for 600^{th} presentation and w_{1999} for 2000^{th} presentation are the cases after more learning epochs in ReSuMe. For DelReSuMe in Figure 8.30c and in Figure 8.30d, there are almost no weight changes compared to 200^{th} presentation which has already stabilized synaptic weights. The unimodal steady-state distribution of synaptic weights is produced.

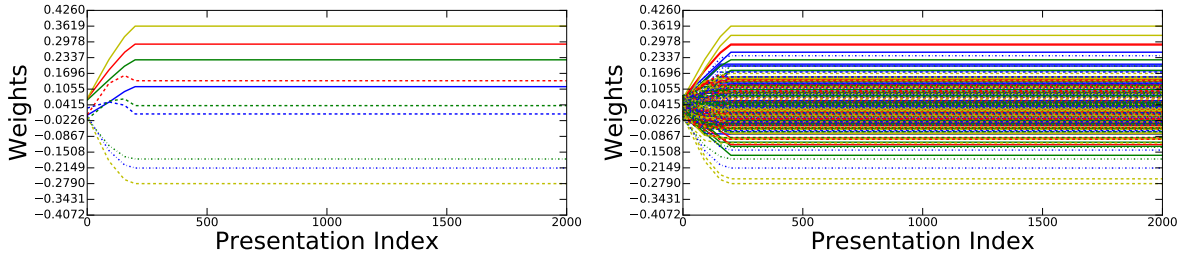


Figure 8.29: Evolution of synaptic weights for the mapping experiments (noiseless condition). In each plot individual weights are represented in a randomly selected colour. (Left) These show only neuron 0 to output neuron with 10 sub-connections during a whole training (2000 presentations). (Right) These demonstrate entire synaptic weight modifications during a whole training (2000 presentations).

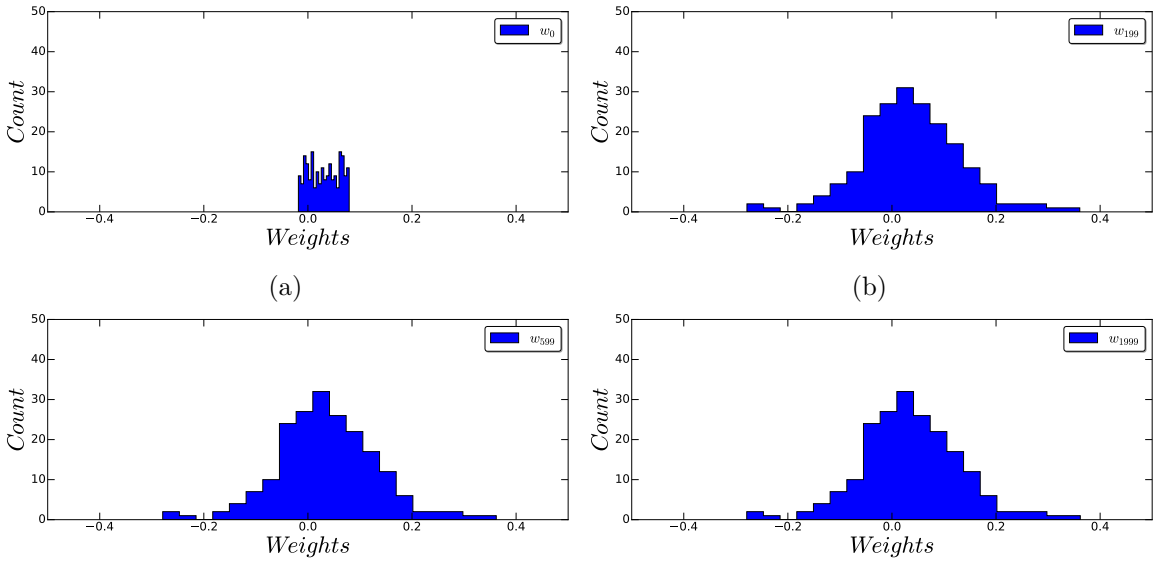


Figure 8.30: Histogram of weights is prepared from the experiment in Figure 8.29 (noiseless condition). Four different characteristic time points are shown in the figure with legend texts: w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{1999} for 2000^{th} presentation in (d).

While the learning is performed in DelReSuMe, the distribution covers the full range of available weights seen through selected time points similar to the ReSuMe experiments in subsection 8.7.1. It is also clear that the shape of weight distributions including their ranges almost do not change between 200^{th} and 2000^{th} presentation; because the stabilization of the

learning is already achieved by this presentation (see also Figure 8.27 and Figure 8.29).

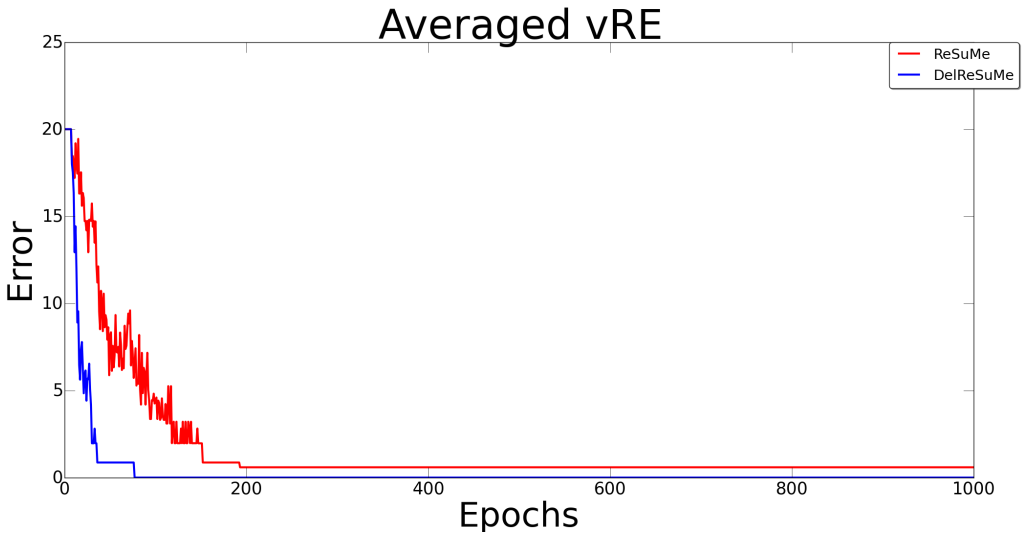


Figure 8.31: The comparison of Remote Supervised Method and Delayed Remote Supervised Method through mapping tasks. X-axis (Epochs) indicates the number of epoch N_E which consists of presentations N_P summarized in Table 9.1. See text for a detailed explanation of the figure.

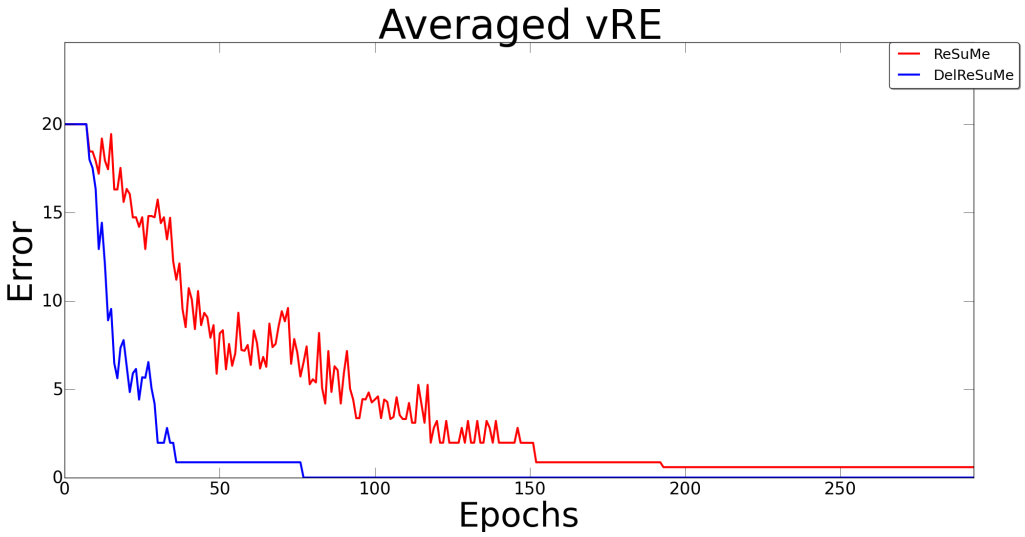


Figure 8.32: Zoomed version of the comparison of Remote Supervised Method and Delayed Remote Supervised Method through mapping tasks. X-axis (Epochs) indicates the number of epoch N_E which consists of presentations N_P summarized in Table 9.1. See text for a detailed explanation of the figure.

The performance of DelReSuMe is compared with ReSuMe with the spatio-temporal mapping benchmark which is detailed in section 7.9. As seen in Figure 8.31, DelReSuMe has faster convergence than ReSuMe. In less than 40 epochs DelReSuMe can learn to map input-output patterns by adjusting synaptic weights and delays at the same time. However, ReSuMe learns

the same task in around 180 epochs in subsection 8.7.1. Furthermore, some patterns which have early spiking activities cannot be learnt by ReSuMe. This effect can be seen in Figure 8.8. However, those tasks are learnt by DelReSuMe precisely. The zoomed version of the same figure focusing on early training cycles is illustrated in Figure 8.32.

In the following section, the simulations are also shown under noise to demonstrate any detrimental effect that the plasticity may have. This helps to verify the reliability and robustness of the plasticity mechanism compared to ReSuMe.

8.10.2 Results of Noisy Simulations

Similar to the ReSuMe experiments in section 8.7.2, we start with the deterministic model of neurons in noise-free learning conditions through experiments in previous section. In addition, the reliability of the network is also investigated in the face of noise interference during training sessions. Same noise levels in section 8.7.2 as relatively low noise and relatively high noise are considered. The details of the inserted noise and how they affect neuronal activity are detailed in section 7.3. The characteristics of low noise and high noise are depicted in Figure 7.1 and in Figure 7.2, respectively. For the low noise, the actual and desired spiking activities through learning are illustrated in Figure 8.33. The evolution of synaptic weights is depicted in Figure 8.34. Also the histogram of weights through four different learning times is demonstrated in Figure 8.35.

For the high noise, the actual and desired spiking activities through learning are illustrated in Figure 8.36. The evolution of synaptic weights is depicted in Figure 8.37. Also the histogram of weights through four different learning times is demonstrated in Figure 8.38.

Inserting noise into the experiments causes slight shifting of actual spiking times (compared to desired spiking times) especially for the relatively high noise experiments. DelReSuMe has faster convergence compared to ReSuMe under the low noise without any spike shifting and undesired activity. Although ReSuMe and DelReSuMe have similar undesired spike shifting in the face of high noise, DelReSuMe does not generate extra actual spikes compared to desired activity in the further learning points unlike ReSuMe. Although the precision of actual spiking activities gets worse while noise level is increased illustrated in Figure 8.36, the speed of

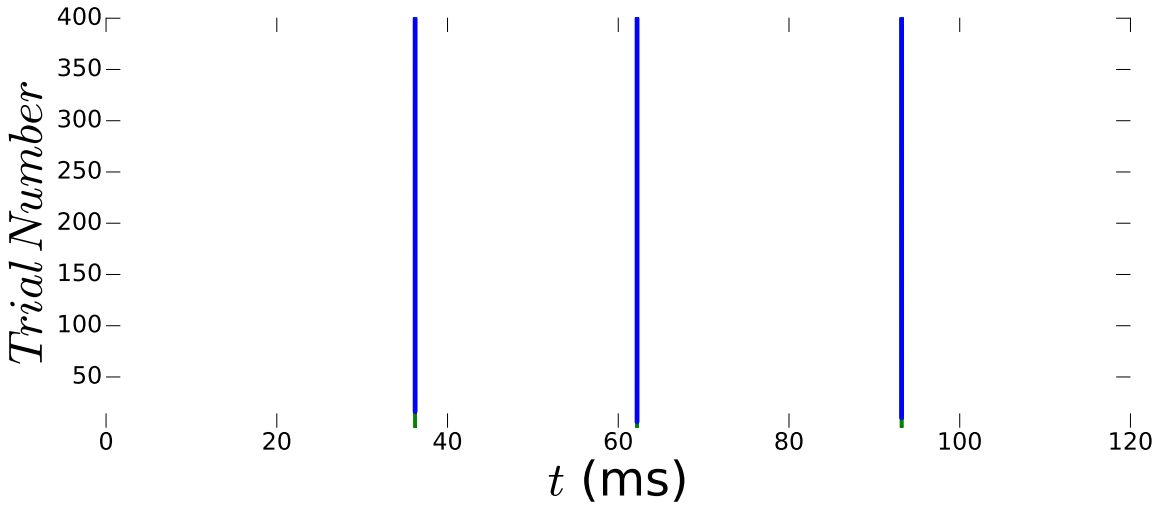


Figure 8.33: Reconstruction of the transformation from input patterns to output spike timings (low noise). Trial number (y-axis) indicates the number of epoch N_E which consists of presentations N_P summarized in Table 9.1. The current network is trained to map a spatio-temporally encoded input spike trains into another spatio-temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour.

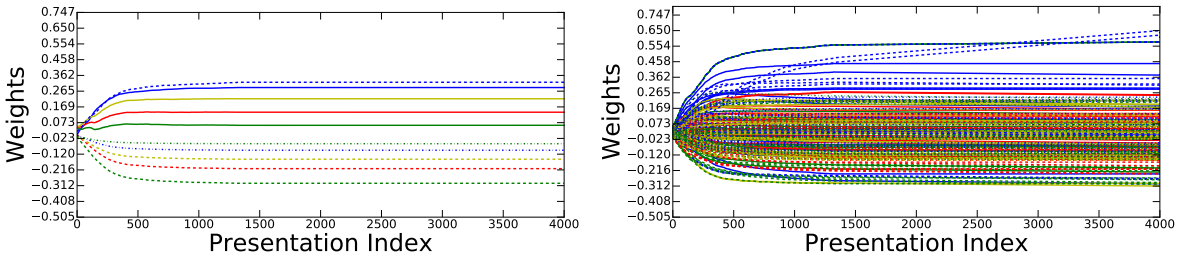


Figure 8.34: Evolution of synaptic weights for mapping experiment (low noise). In each plot individual weights are represented in a randomly selected colour. (Left) It shows only neuron 0 to output neuron with 10 sub-connections during a whole training (4000 presentations). (Right) It demonstrates entire synaptic weight modifications during a whole training (4000 presentations).

learning convergence in DelReSuMe gets worse. However, DelReSuMe is affected less from the noise in terms of the convergence speed. To sum up, DelReSuMe has faster convergence, also DelReSuMe is more robust to the same noise level compared to ReSuMe. The shape of synaptic weight distribution is the unimodal similar to noiseless conditions.

8.10.3 Discussion of Mapping

Although ReSuMe generates extra actual spikes compared to desired activity in the further learning cycles under the high noise, DelReSuMe does not generate extra spikes under the same noise level. Therefore, DelReSuMe is more robust to the limited level of high noise compared

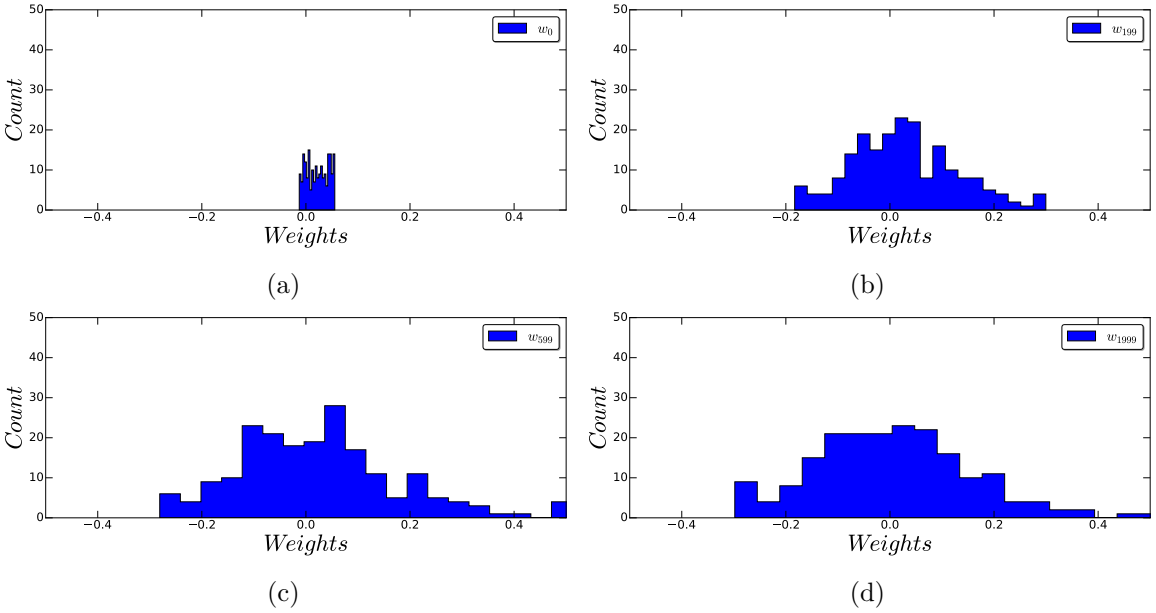


Figure 8.35: Histogram of weights is prepared from the experiment in Figure 8.34 (low noise). Four different characteristic time points are shown in the figure with legend texts: w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{1999} for 2000^{th} presentation in (d).

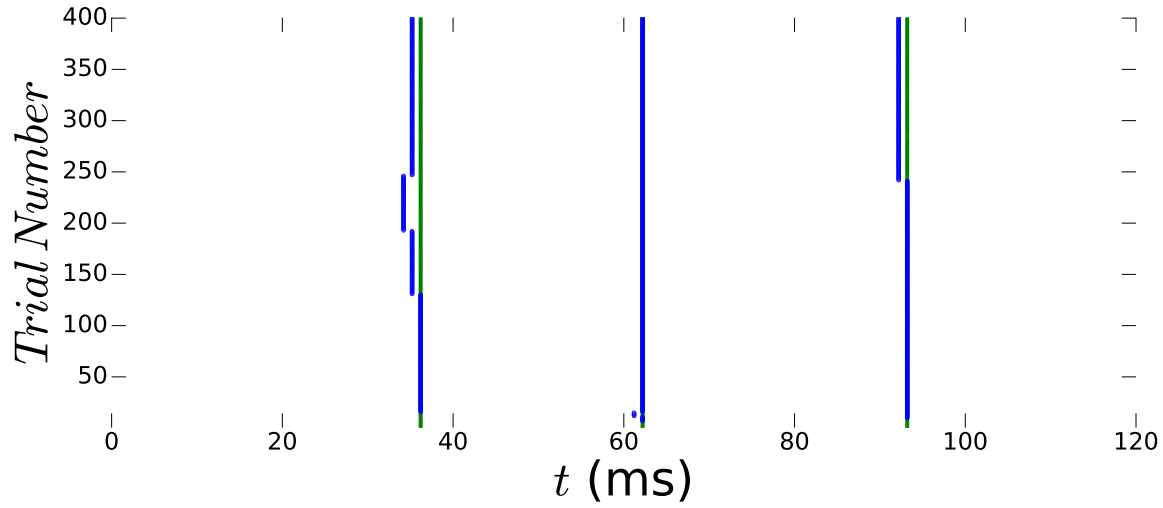


Figure 8.36: Reconstruction of the transformation from input patterns to output spike timings (high noise). Trial number (y-axis) indicates the number of epoch N_E which consists of presentations N_P summarized in Table 9.1. The current network is trained to map a spatio-temporally encoded input spike trains into another spatio-temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour. Upper figure is from one experiment and below figure is from another experiment.

with ReSuMe. Also DelReSuMe has faster learning speed during not only noiseless but also noisy conditions: low noise and high noise. Another observation is the stabilization level for the synaptic weight values is relatively around smaller values for DelReSuMe compared to

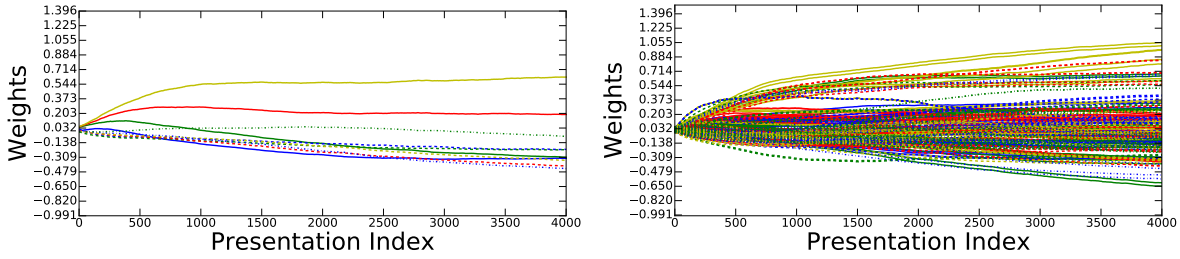


Figure 8.37: Evolution of synaptic weights for the mapping experiment (high noise). In each plot individual weights are represented in a randomly selected colour. (Left) It shows only neuron 0 to output neuron with 10 sub-connections during a whole training (4000 presentations). (Right) It demonstrates entire synaptic weight modifications during a whole training (4000 presentations).

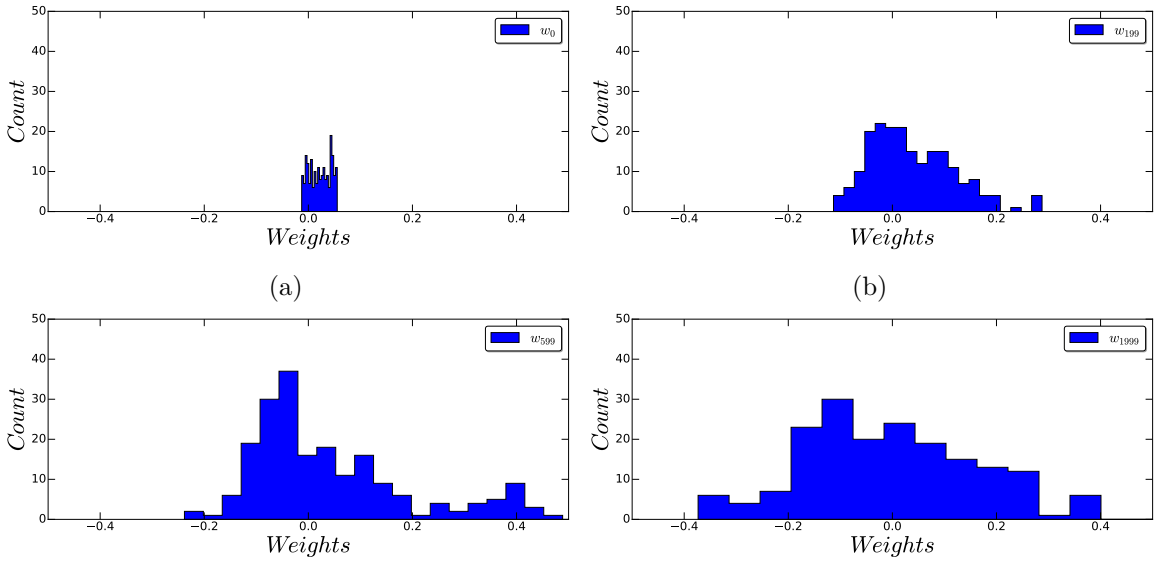


Figure 8.38: Histogram of weights is prepared from the experiment in Figure 8.37 (high noise). Four different characteristic time points are shown in the figure with legend texts: w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{1999} for 2000^{th} presentation in (d).

ReSuMe. This is caused from plasticity of synaptic delays in addition to the plasticity of synaptic weights in DelReSuMe.

8.11 Summary

A new model for existing remote supervision used in ReSuMe is introduced with experimental evidence. The proposed heterosynaptic architecture is an alternative interpretation of the remote supervision. Although the mechanism of remote supervision cannot be modelled on

existing SNN simulators (as in Brian simulator), the proposed alternative topology is currently modelled in Brian. It can also be modelled on other existing SNN simulators.

A computational model using temporal encoding and readout is presented to demonstrate the characteristics of ReSuMe learning. The scheme in the model is biologically plausible. The model is able to associate a spatio-temporal spike pattern with a target spike train. This is initially demonstrated with the mapping benchmark in noiseless and noisy conditions (relatively low and high noise). In addition to single input-output pattern mapping carried out with ReSuMe, more challenging tasks have been investigated under noiseless conditions: the logical operations TRUE, P1, AND, and XOR. Those tests are evidence for the proposed topology introduced in section 8.4 which works in the case of multiple input-output patterns as well. Also, this benchmark can be basis for future work in order to extend single input-output pattern mapping into multiple inputs-output patterns.

Furthermore, an extended version of the existing Remote Supervised Method has been developed, named Delayed Remote Supervised Method (DelReSuMe). The proposed method adjusts synaptic delays as well as synaptic weights. The practical application of the proposed algorithm is demonstrated through mapping experiments in noiseless and noisy conditions (low and high noise).

In order to compare Remote Supervised Method and Delayed Remote Supervised Method under noiseless and noisy conditions, same mapping benchmarks are executed. It is demonstrated that DelReSuMe has faster learning than ReSuMe for the mapping tasks in both noiseless and noisy conditions: low noise and high noise. In addition, DelReSuMe is more robust to the relatively high noise compared to ReSuMe. Although ReSuMe generates extra actual spikes compared to desired activity in the further learning cycles under the high noise, DelReSuMe does not generate extra spikes under the same noise level. Also the stabilization level for the synaptic weights is relatively smaller values for DelReSuMe compared to ReSuMe. This is caused from plasticity of synaptic delays in addition to the plasticity of synaptic weights in DelReSuMe. The shape of synaptic weight distribution is the unimodal for noiseless and noisy conditions for both ReSuMe and DelReSuMe unlike the U-shaped bimodal distribution in the Additive-STDP.

Chapter 9

Reward-modulated STDP

Contents

9.1 Introduction . . . 219
9.2 Reward Mechanism . . . 222
9.3 Eligibility Traces . . . 224
9.4 Comparison between R-STDP and R-max . . . 226
9.5 Overall Setup . . . 228
9.6 Experiments: Mapping . . . 230
9.6.1 Results of Noiseless Simulations (Single-Connection) . . . 231
9.6.2 Results of Noiseless Simulations (Multi-Constant-Delay) . . . 234
9.6.3 Results of Noisy Simulations (Single-Connection) . . . 234
9.6.4 Results of Noisy Simulations (Multi-Constant-Delay) . . . 241
9.6.5 Discussion . . . 242
9.7 Summary . . . 248

9.1 Introduction

In behavioural learning paradigms, strength of behaviour is modified by receiving reward or punishment as reinforcements. Reinforcement Learning (detailed in chapter 4) inspired by behaviorist psychology is an algorithmic approach to reward learning in the Machine Learning discipline (Sutton & Barto, 1998). Recently, several plasticity experiments, including Spike-timing Dependent Plasticity, demonstrate that neuromodulators, particularly Dopamine (DA) related to novelty and reward prediction, have global mechanisms for synaptic modification (Pawlak & Kerr, 2008; Wickens, 2009; Vasilaki et al., 2009). In order to consolidate the changes of synaptic strength in response to pre- and post-synaptic neuronal activities, an additional reward signal can be used to control learning (Bailey et al., 2000; Vasilaki et al.,

2009).

Spike-timing Dependent Plasticity (STDP) described in section 3.4 is a spike pairing protocol where weight changes are determined according to pre-synaptic and post-synaptic firing times (Bi & Poo, 1998; Sjostrom et al., 2001; Froemke et al., 2005). STDP and Hebbian mechanisms are types of unsupervised plasticity that result as a function of the neuron's own dynamics (Gerstner et al., 2014). On the other hand, according to experimental studies neuromodulators such as acetylcholine, noradrenaline, dopamine can also influence synaptic plasticity and memory formation (Rasmusson, 2000; Gu, 2002). The Reinforcement Learning discussed in chapter 4 shows that the neurotransmitter dopamine (DA) is linked to reward mechanisms (Schultz et al., 1997; Schultz, 1998). Also, the interaction of DA signals with STDP mechanism is studied in several regions of the brain as in the amygdala (Bissiere et al., 2003), in rat visual cortex layer (Seol et al., 2007), in corticostriatal synapses (Pawlak & Kerr, 2008), and in prefrontal cortex layer (Xu & Yao, 2010).

Updating synaptic strengths through global neuromodulators (Seol et al., 2007) can be viewed as a teacher signal, and called semi-supervised learning. This mechanism which is based on Dopamine neuromodulation and its biological justification is discussed in section 3.6. Temporal-Difference (TD) learning (see Figure 9.1) as a type of Reinforcement Learning technique is linked to biological considerations by using eligibility traces which track the synaptic firing times from recent spikes (Farries & Fairhall, 2007; Florian, 2007; Izhikevich, 2007b). Inspired by these observations, two levels of plasticities using STDP and reward are hypothesised as Reward-modulated Spike-timing Dependent Plasticity (R-STDP) in several theoretical and practical studies (Farries & Fairhall, 2007; Florian, 2007; Izhikevich, 2007b; Legenstein et al., 2008; Fremaux et al., 2010, 2013; Fremaux & Gerstner, 2015). Similar to them, experiments in this paper are implemented using R-STDP mechanism.

Our motivation here is that the precise times of individual spikes might be fundamental for efficient computation in capturing certain temporal dynamics. This paradigm is demonstrated in neurobiological research (Bi & Poo, 1998). The STDP process performs a small weight change for a longer time difference and a larger change for a shorter time difference between pairs of spikes. Here, instead of applying those changes directly, they are modulated using the reward signal similar to a supervisory signal, motivated by sensory feedback in the brain

(Carey et al., 2005).

During experiments, the reward is provided by criticizing the actual and desired output neuron activities at the end of each learning cycle. Then, the modulated weights are applied to synaptic connections. The goal of the Spiking Neural Network is to modify the vector of synaptic weights w to achieve desired spike patterns at the training neuron's output $S_d^{out}(t)$ in response to the given M input sequences $S_1^{in}(t)$, $S_2^{in}(t)$, ..., $S_M^{in}(t)$. Although, here input and output spike sequences use Poisson spike timings as random processes (detailed in subsection 6.3.2), the proposed plasticity mechanism is not dependent on this interpretation. Therefore, other spike-generation processes should work as well.

In this chapter, the R-STDP rule of maximizing the reward is implemented (Fremaux et al., 2010). Hebbian plasticity is modulated by the reward signal analogous to Dopamine modulation. A transient memory of synaptic events is stored at each synapse as synaptic eligibility traces. The reward is calculated at the end of each training cycle. Once the reward occurs, synaptic weights are updated.

Temporal sequence schemes can contain a single spike or multiple spikes in the simulation time window as described in section 6.4. The novel aspect of this study addresses multi-spike timings rather than a single-spike coding scheme to achieve a biologically more realistic scheme. A dopaminergic-inspired learning rule combined with STDP using multi-synaptic connections rather than a single connection from each input and output neurons is shown both analytically and through computer experiments to have rapid convergence under noiseless, low noise and high noise conditions. The development of R-STDP with increased learning speed helps generic learning tasks where a neuron is supposed to respond to input spike patterns with specific output spikes. Through the proposed architecture, the problem of getting multiple spikes into and out of the Spiking Neural Network is solved, and the task of mapping is fulfilled.

This chapter demonstrates that Spiking Neural Networks encoding information in the timing of multiple spikes are capable of learning to map spike patterns with multiple spikes. Firstly, the reward mechanism is introduced in section 9.2. Then, the details of eligibility trace is presented in section 9.3. In section 9.4, empirical Dopamine R-STDP is briefly compared with the theoretically derived R-max rule. In the following sections two benchmarks are

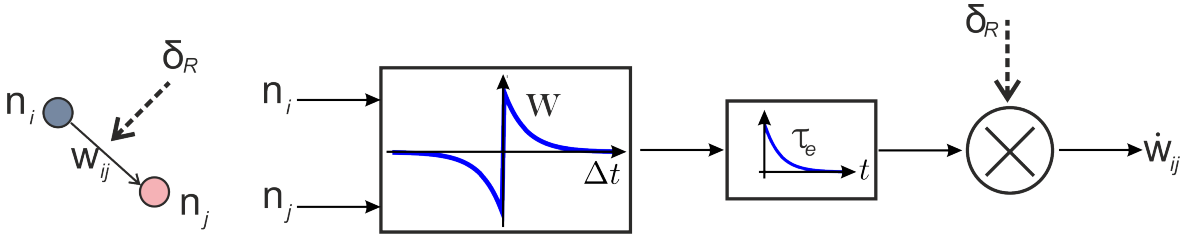


Figure 9.1: Schematic of Reward-modulated Spike-timing Dependent Plasticity learning rule. In STDP detailed in section 3.4, a coincidence window W consists of both post-before-pre and pre-before-post components. Spiking activities are counted if the post-synaptic firing time occurs within a few ms of a pre-synaptic spike based on decay time constants as τ_{pre} and τ_{post}. The result of the coincidence measure is filtered through a τ_e kernel, and then multiplied by the TD error δ_R in order to control synaptic weight w_{ij}.

demonstrated aiming to assess the computational performance of the proposed method as a biologically plausible implementation of Reinforcement Learning. Finally, the mapping benchmark in section 9.6 is investigated under noiseless, low noise and high noise conditions in order to compare robustness and performance of the proposed topology. The details of the inserted noise and how they affect neuronal activity are detailed in section 7.3. The characteristics of low noise and high noise are depicted in Figure 7.1 and in Figure 7.2, respectively.

9.2 Reward Mechanism

A reward (punishment) is generated as a global neuromodulatory signal based on how close the actual activity to the target spiking activity. The reward signal employs the synaptic weights regulated by STDP detailed in section 3.4. Hence, locally calculated and stored changes are applied according to the level of a global neuromodulator (Vasilaki et al., 2009). This broadcast of the global reward signal to all synapses leads to Reinforcement Learning. Neuromodulators such as Dopamine is linked to a reward signal in the nervous system (Schultz et al., 1997; Schultz, 1998; Vasilaki et al., 2009) (see section 3.6).

The normalized measure D_R^{norm} described in Equation 6.13 (detailed in subsection 6.5.4) between the actual and desired spike trains for the output neuron as a function of time t in the current presentation is mapped into a reward signal r (Farries & Fairhall, 2007) as:

$$r = exp(-\alpha D_R^{norm}) \tag{9.1}$$

where $D_R^{norm} \in [0, \infty]$ and $r \in (0, 1]$. The reward mapping factor α is set as 3 in order to ignore distances $D_R^{norm} > 1$ as in Table 9.2. The restrictions can be summarized as:

$$r = \begin{cases} 0, & \text{if} \quad \text{no output spike,} \\ 1, & \text{if} \quad D_R^{norm} = 0, \\ exp(-\alpha D_R^{norm}), & \text{otherwise} \quad . \end{cases} \tag{9.2}$$

$r = 1$ indicates a perfect match between actual and desired spike trains. In order to avoid network stagnation, we explicitly set $r = 0$, if there is not any output spikes generated.

A Temporal-Difference (TD) technique tracks discounted reward as it is defined in Reinforcement Learning (Sutton & Barto, 1998) and discussed in chapter 4. An adapted version of the TD rule is applied in order to track average reward as proposed by Farries & Fairhall (2007).

$$\delta_R^n = r_{curr}^n - r_{avg}^n \tag{9.3}$$

where n is the trial number, δ_R^n is the Temporal-Difference error between the current reward received r_{curr}^n and the expected reward r_{avg}^n smoothed exponentially over previous trials as defined in Equation 9.4, and is used to improve the current policy on synaptic strengths. Maximizing the average reward per presentation maximizes the total future reward in order to achieve an optimal policy (Sutton & Barto, 1998; Farries & Fairhall, 2007). In this model, the environmental state in RL concept is the spike trains provided by the input layer and the chosen action in RL is the spike trains generated by the output neuron.

The running average of reward as a function of time t through learning cycles is calculated as:

$$r_{avg}^n = \gamma r_{avg}^{n-1}(t) + (1 - \gamma) r_{curr}^n \tag{9.4}$$

where n indicates the presentation number (learning cycle). Here, all future rewards are discounted by discount factor, γ where $0 < \gamma < 1$. It is selected near to the maximum value so recent rewards have higher utility than older rewards as $\gamma = \frac{9}{10}$ and $(1 - \gamma) = \frac{1}{10}$ as in Farries & Fairhall (2007); Sutton & Barto (1998). Hence, it provides exponential smoothing.

9.3 Eligibility Traces

The eligibility trace is the basic building block of Reinforcement Learning. Synaptic eligibility traces based on theoretical considerations store a temporary memory of the past Hebbian coincidence events at the site of the synapses (local memory) until a reward signal is received. The eligible time of the synapse for reinforcement by a reward signal $r(t)$ is determined by the term of the eligibility trace $e_{ij}(t)$ between neuron i and j. It is a decaying memory over time between pre-synaptic and post-synaptic spiking activities. Multiple spike pairs are accommodated by assuming linear contributions.

Inspired from the essence of Dopamine modulation of STDP in Izhikevich (2007b), STDP-dependent eligibility trace model can be expressed as:

$$\frac{de_{ij}(t)}{dt} = -\frac{e_{ij}}{\tau_e} STDP(\Delta t) \tag{9.5}$$

where $STDP(\Delta t)$ is traditional Hebbian-STDP (see section 3.4), τ_e is a decay time constant of the eligibility trace e_{ij}. The candidate changes in synaptic weights e_{ij} are known as the eligibility trace in Reinforcement Learning (Sutton & Barto, 1998; Fremaux et al., 2010). Eligibility is generated by filtering with an exponential function the synaptic eligibility trace $e_{ij}(t)$ of the synapse from pre-synaptic neuron i to post-synaptic neuron j. Firing times of pre- and post-synaptic neurons, occurring at times t_i^{pre} and t_j^{post} respectively, change $e_{ij}(t)$ by the amount of $STDP(\Delta t)$ with $\Delta t = t_j^{post} - t_i^{pre}$. The eligibility trace captures an unsupervised Hebbian learning mechanism through correlations between pre- and post-synaptic firing activities. The process is illustrated for both positive and negative spike timings in Figure 9.2.

The time constant of the eligibility trace τ_e regulates the rate of decay. It determines the maximal interval between the pre-post coincidences and the reward signal. Decay constant τ_e is typically chosen around one second (Florian, 2007; Fremaux et al., 2010); however, it is manipulated in our experiments in order to capture all pre-post coincidences within a single trial. It is the same as the duration of the input pattern in our simulations (see Table 9.2).

The actual synaptic modification also requires the presence of a neuromodulatory signal which is derived previously as δ_R in Equation 9.3. δ_R is general symbol of TD error, δ_R^n is the TD error at learning cycle n. Therefore, the product of eligibility trace and the reward signal drive

the synaptic plasticity. Under constant reward (Dopamine), the overall change in synaptic strength based on the modulated plasticity rule is the same as "standard" STDP detailed in section 3.4. The reward signal is the same for all synapses at the end of each learning cycles throughout the network; therefore, it can be considered as a global factor transmitted by a neuromodulator. The synaptic modification formula of R-STDP weights can be defined as:

$$\Delta w_{ij}^{n} = \eta \ \delta_{R}^{n} \ e_{ij}(T_{pe}) \tag{9.6}$$

where Δw_{ij}^{n} is the weight change, η is the learning rate, δ_{R}^{n} shows the presence of a neuromodulatory success signal (not to be confused with the Dirac Delta function δ), T_{pe} is the period of a single learning cycle presentation time, $e_{ij}(T_{pe})$ is the value of the eligibility trace over the n^{th} presentation with time $t = T_{pe}$ (see Table 9.1). This learning paradigm is applied into all synapses between input neurons and the actual output neuron at the end of each presentation. The modification Δw_{ij} is applied to a weight w_{ij} according to an additive update rule (Morrison et al., 2008) detailed in subsection 3.4.2 as:

$$w_{ij}^{n+1} \leftarrow w_{ij}^{n} + \Delta w_{ij}^{n} \tag{9.7}$$

Instead of delivering the reward to the network at every time moment, we apply it only at the end of each learning cycle $t = t_r = T_{pe}$ as depicted in Figure 9.2. Each pre-synaptic and post-synaptic spike pairs create a step change contribution to the eligibility trace e_{ij}. Overall plastic change at a single synapse is then the sum of contributions from both positive and negative spike timings. The eligibility decays exponentially with time constant τ_e. The magnitude of the reward signal (reinforcement signal) across trials is changed according to the eligibility trace as shown in Figure 9.2. The parameter of learning rate η controls the speed of learning and it is positive-valued (see Table 9.2).

In Reward-modulated Spike-timing Dependent Plasticity, the synaptic weight changes from STDP are not immediately applied. The relationship of pre- and post-synaptic spikes is stored in each synaptic eligibility trace locally for further weight changes. The product of this synaptic trace with a reward signal is integrated through weight changes illustrated in Figure 9.1. The eligibility trace in the form of R-STDP permits learning with delayed reward as well.

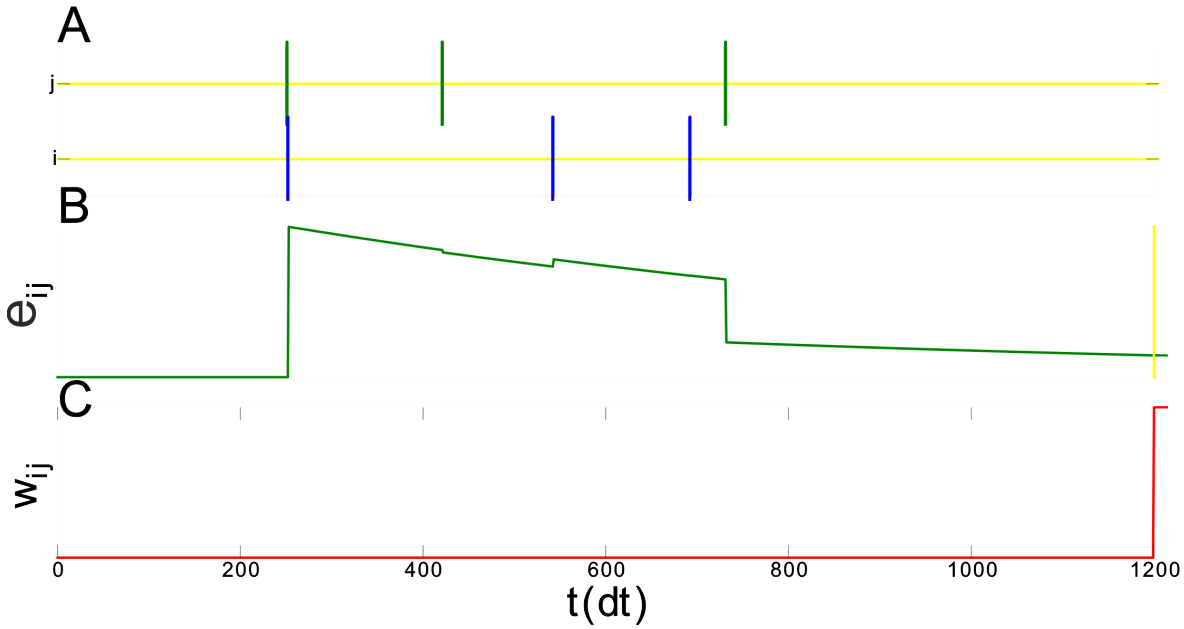


Figure 9.2: An example of eligibility trace and weight update during a single learning cycle. X-axis shows total simulation time steps during the period of single learning cycle T_{pe} as $T_{pe} * dt$ = 120 ms / 0.1 ms = 1200 time steps. A) The pre-synaptic firing times are shown in the top row with t_j^n coloured green and the post-synaptic spike train in the bottom row with t_i^f coloured blue. B) The eligibility trace e_{ij} coloured green keeps past Hebbian events from pre-synaptic and post-synaptic spikes in (A). At the end of the learning cycle T_{pe} = 120 ms, the current reward receives which is drawn in yellow. Weight change is determined using the value of eligibility trace at T_{pe} = 120 ms and the reward in Equation 9.6. C) Weight change is applied once the reward is received at the end of the presentation at time $t = t_R$ corresponding to the time of reward delivery, $t = t_R = T_{pe}$.

9.4 Comparison between R-STDP and R-max

Although Hebbian learning depends on two factors as pre- and post-synaptic firing activities, R-STDP depends on three factors (see section 4.1.1 for three-factor learning), two Hebbian factors and the neuromodulator, (Fremaux & Gerstner, 2015). R-STDP is based on the experimental evidence that neuromodulators regulates the synaptic modifications due to STDP (Florian, 2007; Farries & Fairhall, 2007; Legenstein et al., 2008). The effect of Dopamine in the brain is strongly related to the Reinforcement Learning according to various experiments (Schultz et al., 1997; Reynolds et al., 2001; Pawlak & Kerr, 2008).

Reward-maximisation (R-max) rule is derived theoretically in order to maximize the received reward from the application of policy gradient methods for a stochastically spiking neuron model (Xie & Seung, 2004; Pfister et al., 2006; Florian, 2007; Fremaux & Gerstner, 2015). R-STDP is quite sensitive to changes in the mean reward value. The average synaptic weight

change is split into the unsupervised and reward learning parts because of RL rules (Farries & Fairhall, 2007; Florian, 2007; Fremaux et al., 2010). The reward learning element depends on the covariance between pre-synaptic/post-synaptic activity and the reward, while the unsupervised learning element depends on only the mean reward level. The main difference between R-STDP and R-max is the unsupervised part: in R-max rule it is always zero in contrast to R-STDP. Therefore, R-max rule is not influenced by the mean reward level unlike R-STDP (Farries & Fairhall, 2007; Legenstein et al., 2008; Fremaux et al., 2010).

Both rules then have very similar requirements for LTP except the shape of coincidence kernels (Fremaux et al., 2010). However, the LTD requirements are quite different. In R-STDP, LTD depends on post-synaptic spikes, whereas only the instantaneous firing rate, which is equal to the density of spikes at time t (Gerstner et al., 2014), counts for R-max rule (Fremaux et al., 2010; Fremaux & Gerstner, 2015). Both plasticity mechanisms rely on three-term rule (detailed in section 4.1.1) with a structure of a Hebbian plasticity that is modulated by the global reward. Although derived R-max rule has performance advantages as faster learning because of the insensitivity to mean success signal, there is a lack of experimental evidence for the biological underpinnings about R-max rule (Florian, 2007; Fremaux et al., 2010). The success signal in our notation is interpreted as a Temporal-Difference error δ_R^n which calculates the difference between an internal estimate of the expected reward and the current reward. In addition, R-STDP without LTD performs better than balanced R-STDP for single-connection, its performance quite similar to R-max (Fremaux et al., 2010).

To sum up, there are two main forms of learning mechanisms for reward modulation based on Equation 9.5 and Equation 9.6 as an empirically formulated R-STDP mechanism, and a theoretically derived Reward-maximisation (R-max) rule. We choose to focus on R-STDP because of its biological basis/biological realism on spike timing. In addition, the proposed network architecture with multi-synaptic connection is not dependent only the learning rule of Reward-modulated Spike-timing Dependent Plasticity. Hence, R-max rule could also be mapped to the proposed architecture.

9.5 Overall Setup

The form of single connection without synaptic delays is compared with multiple constant delay mechanism detailed in section 7.4 under the learning mechanism of R-STDP. In the case of multi-constant-delay, the number of synaptic connections is $N_{sub} = K = 10$ (see Table 9.4) with non-programmable delays between $1 - 10$ ms, $d_{ij,k} = k(ms)$. However, the bias input neuron $n_{i=0}$ (see subsection 7.5.3) has a single connection without a delay, $d_{ij,k=0} = 0$ ms. Network architecture used in R-STDP tasks is already introduced in section 7.5. The types of neuron models used in proposed SNNs under the noiseless conditions are summarized in section 7.2. In addition, the training and testing stages of performed experiments are also described in subsection 7.8.1 and subsection 7.8.2, respectively.

Each presentation runs for the simulated $T_p = 100$ ms duration of the input spike trains also with two times the maximal synaptic delay $2 * \tau_m = 20$ ms added as $T_{pe} = 120$ ms (see Table 7.1). Similar to the experiments in section 8.7 and section 8.10, the simulation of network dynamics is performed on a time step of $dt = 0.1$ ms. The number of epochs N_E and the number of presentations N_P are shown in Table 9.1. Total simulation time can be calculated by $T_{pe} * N_P * N_E$ with the parameters in Table 7.1 (see subsection C.9.10 for details).

Parameter Type	Parameter Names	Values
Presentation Time	T_{pe}	120 ms
Run Time Resolution	dt	0.1 ms
Presentation Number	N_P	1
Epoch Number	N_E	3000

Table 9.1: Length parameters used for the computer simulations through R-STDP.

In order to avoid either silencing or extreme network activity by frequent firings of neurons, all synaptic weights are bounded with a lower boundary w_{min}^{TRN} and an upper boundary w_{max}^{TRN} similar to previous experiments in section 8.7 and section 8.10 .

In this model, the actual output neuron is trained to respond with spatio-temporally encoded input spike patterns by tuning synaptic weights. The strength of synaptic connection w^* between input layer and desired output neuron is fixed and set to maximum weight value as $w^* = w_{max}^{TRN}$ in order to force desired neuron to generate spikes at each target times. Other

Parameter Type	Parameter Names	Values
Plasticity Amplitudes	A_{pre}, A_{post}	0.010, 0.005
Decay constants	τ_{pre}, τ_{post}	10 ms, 20 ms
Weight Limits	$[w_{min}^{TRN}, w_{max}^{TRN}]$	[-1, +1]
Initial Weight Limits	$[w_{min}^{init}, w_{max}^{init}]$	[-0.02, 0.08]
Excitatory/Inhibitory Rate	$[r_{exc}^{init}, r_{inh}^{init}]$	(0.2, 0.8)
Eligibility Decay	τ_e	100 ms
vRD Decay	τ_R	10 ms
vRD Period	T_R	120 ms
Learning Rate	η	200.0
Reward Mapping Factor	α	3.0

Table 9.2: Model parameters used for the computer simulations during R-STDP.

weights are initialised uniformly with the range $[w_{min}^{init}, w_{max}^{init}]$ as in Table 9.2. They are chosen quite small so there is no spike generation until the learning performs adjustment of firing rate. Synaptic connections are allowed to contain a mix of both positive and negative weight values. Synaptic connections with negative weights behave like inhibitory synapses. Therefore, the initial weight range is set as 20% (inhibitory) and 80% (excitatory) analogous to observations in mammalian neocortex (DeFelipe & Farinas, 1992) as described in Table 9.2 with r_{exc}^{init}, r_{inh}^{init}, respectively.

Activity-dependent homeostatic weight renormalization is performed in order to stabilize neuronal firing rates. Synaptic scaling described in subsection 3.5.1 is implemented in the proposed plasticity scheme with the parameters of scaling factor β, scaling range rate p in Table 9.3. The desired spike number from the output neuron N_{des} is $N_{des} = 3$ because $r_{out} = 0.03$ Hz in the window $T_p = 100$ ms as $N_{des} = r_{out} * T_p$.

Parameter Type	Parameter Names	Values
Desired Spike Number	N_{des}	3
Scaling Range Rate	p	0.02
Scaling Factor	β	0.001

Table 9.3: Scaling parameters for R-STDP.

9.6 Experiments: Mapping

Same mapping benchmark used in previous experiments in section 8.7 and section 8.10 is also implemented throughout this chapter. The mapping benchmark detailed in section 7.9.1 is to map the timings of input spike patterns into the target output patterns precisely. The main learning phenomena relied on here is the framework of Reward-modulated Spike-timing Dependent Plasticity, which modulates the outcome of Spike-timing Dependent Plasticity by a neuromodulator signal. One of the fundamental aims through experiments in this chapter is to compare the learning performance and the reliability of single-connection without delay versus the proposed multi-constant-delay connection under noiseless, low noise and also high noise conditions detailed in section 7.3.

The block diagram of mapping benchmark (see section 7.9.1) is demonstrated in Figure 7.11. There is a single input bank named $P1$. A spatiotemporally encoded spike pattern is applied to the input bank ($P1$). The spiking network is trained to produce a desired spike train which corresponds to the input pattern at the output (Q).

To have a fair comparison between without delay and with multi-constant-delay structures, we use the same number of synapses in both cases: a single connection without delay and multi-synaptic connections with delays from each input neuron to output neuron summarized in Table 9.4. The network form with single synaptic connection between each input-output neuron has 200 neurons in the input layer. The form of multiple synapses with multi-constant-delay architecture has only 20 input neurons with 10 synaptic connections each with varied delays from each input neuron to output neuron. In both cases, the total number of plastic synapses is the same, 200x1=200 total synaptic connections in the mechanism of single-connection and 20x10=200 total synaptic connections in the mechanism of multi-constant-delay.

Parameter Type	Values for single-connection	Values for multi-connection
$N_{in} = M$	200	20
$N_{sub} = K$	1	10
$N_{out} = O$	2	2
$N_{syn} = S$	200	200

Table 9.4: Parameters of the network architecture for mapping benchmark with a single-connection and multi-constant-delay mechanism.

Each neuron in the input layer is fed with a different spike train pattern, $S_1^{in}(t)$, $S_2^{in}(t)$, ..., $S_M^{in}(t)$, driven by external Poisson spike trains which are outlined in the section 7.6. There is also a bias input neuron layer with a single neuron (see subsection 7.5.3) as a reference in order to force the comparator neuron's desired spike pattern generation. In the output layer, there are two neurons, $N_{out} = 2$, one is generating the actual output patterns $S_a^{out}(t)$. The other neuron produces desired spiking activities $S_d^{out}(t)$. Furthermore, a critic unit is included which uses actual cell activity and desired target cell output in order to generate appropriate reward modulation of synaptic weights between input neurons and the output neuron. To sum up, all mapping experiments are carried out using a two-layer network with two types of input neuron numbers with single or multiple synapses to output layer, 1 bias input neuron, 1 desired output neuron and 1 training output neuron summarized in Table 9.4.

During entire simulations, we use the same learning rate η as in Table 9.2. Same initialization method is used for all simulations. All desired output patterns have exactly the same number of spikes to have fair comparison. It is 3 spikes based on the selected firing rate (see Table 7.2). Experiments are executed 10 times with randomly selected configurations, and averaged results are demonstrated.

Similar to previous experiments in section 8.7 and section 8.10, we execute the mapping benchmarks under noiseless, low noise and high noise conditions (see section 7.3) in order to compare robustness and performance of the network in the following sections.

9.6.1 Results of Noiseless Simulations (Single-Connection)

Under the noiseless conditions, a desired output signal $S_d^{out}(t)$ and the convergence of the actual output $S_a^{out}(t)$ over training cycles are depicted in Figure 9.3 for single-connection without delay. On the early runs, there are less spiking activities compared to desired spiking activities because synaptic weights are initialized with small values at the beginning. Once the training continues, the number of generated spikes at the output of network becomes closer to desired activity. After less than 600 episodes the network produces 3 desired spikes within 3 ms of target. With this configuration, it can be clearly seen that the network reliably learned the given pattern within 1500-2500 cycles. Furthermore, the stability preservation of the network by keeping frequencies of neuronal activity in a range can be seen over the entire simulation.

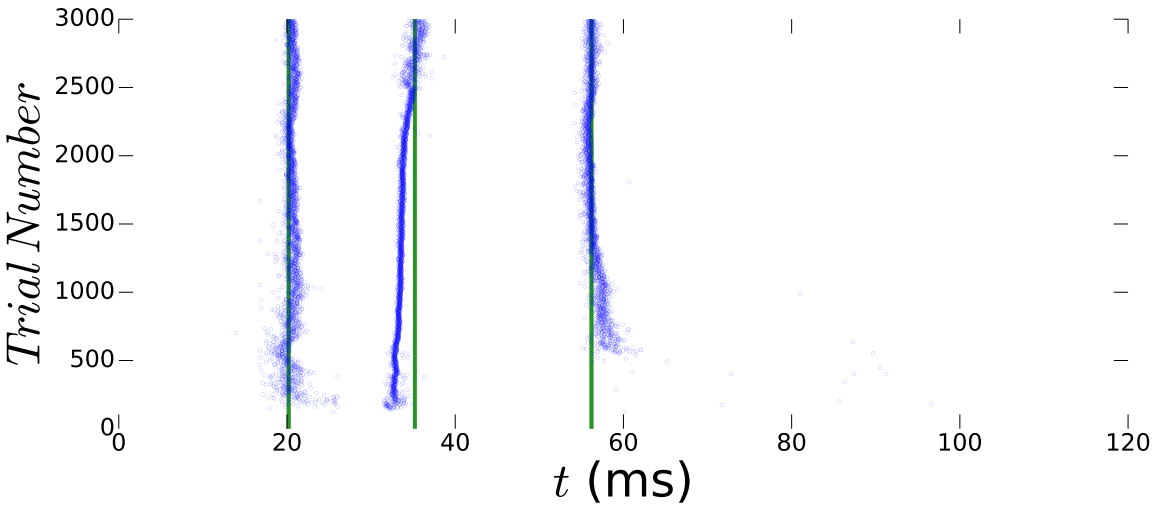


Figure 9.3: Reconstruction of the transformation from input patterns to output spike timings (noiseless, without delay). The current network is trained to map a spatio temporally encoded input spike trains into another spatio temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour.

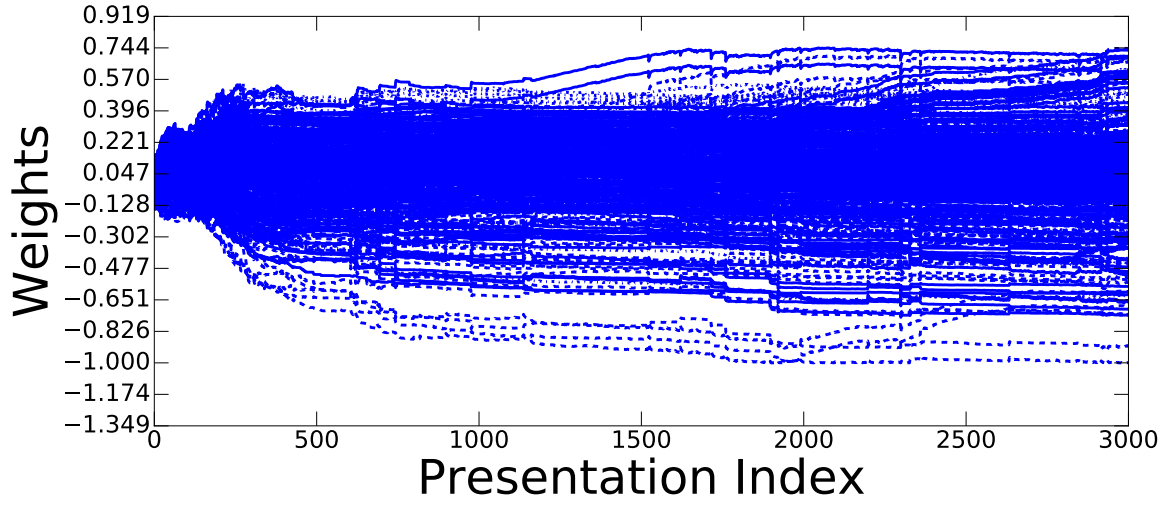


Figure 9.4: Evolution of synaptic weights for the mapping experiment (noiseless, without delay). The figure demonstrates entire synaptic weight modifications during a whole training..

Figure 9.4 demonstrates the weight trajectories during an example training session. It is clear that all synaptic strengths get closer a stable value after 1200-1300 learning cycles. Also, the values of synaptic weights at the end of the successful training can be intermediate values between weight boundaries w_{min}, w_{max} rather than all converging to w_{min} and w_{max} (see subsection 3.4.2).

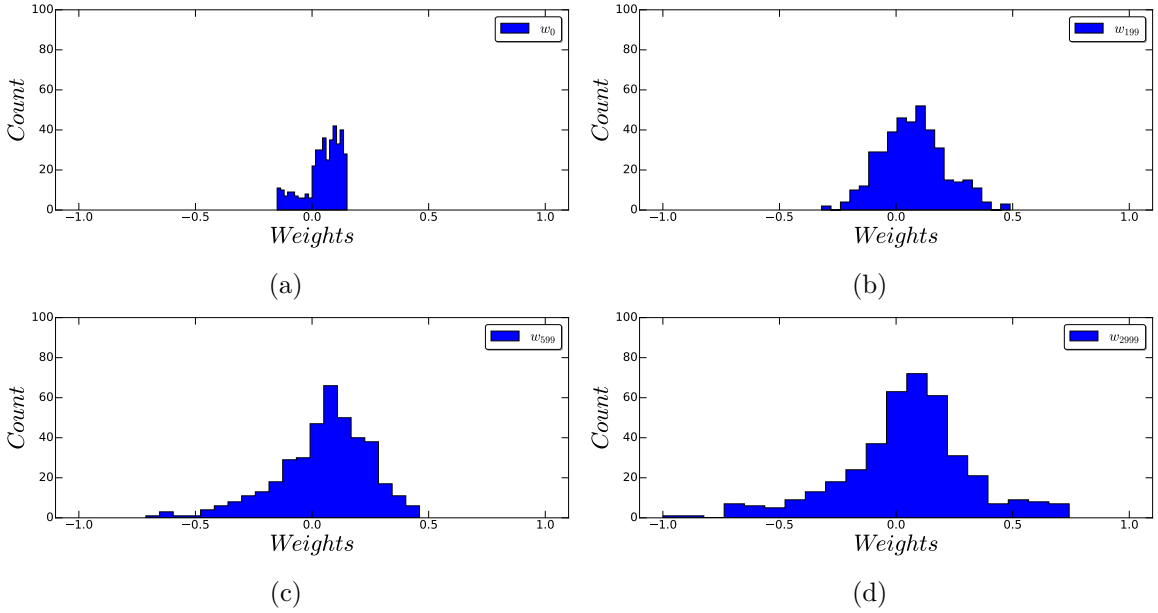


Figure 9.5: Histogram of weights is prepared from the experiment in Figure 9.4 (noiseless, without delay). Four different characteristic time points are shown in the figure with legend texts: w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{2999} for 3000^{th} presentation in (d).

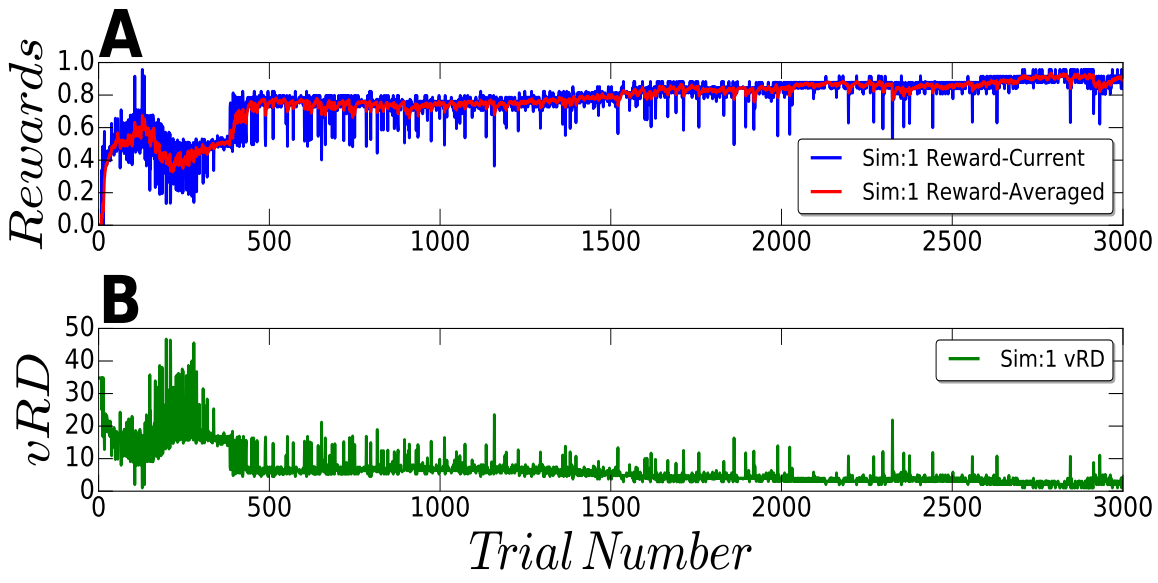


Figure 9.6: The trajectory of current and averaged reward versus van Rossum Distance (vRD) during randomly selected simulations (noiseless, without delay). A) A snapshot of averaged rewards (coloured red) with running average of current rewards (coloured blue). The red line shows the average reward time course $r^{avg}(t)$ over trial numbers 1 to 3000. The blue line shows the current reward time course $r^{curr}(t)$ over epochs. B) The evolution of mismatch between the desired and the actual output signal, $D_R(S_a, S_d)$ based on the van Rossum Distance.

Histogram of weights through noiseless conditions, prepared from the experiment in Figure 9.4, is illustrated in Figure 9.5. Four different characteristic time points throughout learning cycles

are selected in order to show the evolution of synaptic weights. w_0 for 1^{st} presentation is the case of initial weights illustrated in Figure 9.5a. w_{199} for 200^{th} presentation is the case of after limited amount of learning epochs illustrated in Figure 9.5b. w_{599} for 600^{th} presentation is the case of after more amount of learning epochs illustrated in Figure 9.5c. w_{1999} for 3000^{th} presentation is the case of end of simulation illustrated in Figure 9.5d. Those four time points are selected similar to the experiments in section 8.7 and section 8.10. While the learning is performed, the distribution covers the full range of available weights seen through selected time points: 1^{st}, 200^{th}, 600^{th} and 3000^{th} presentations. The unimodal steady-state distribution of synaptic weights is produced similar to the experiments in section 8.7 and section 8.10.

Convergence and error trajectory through the learning for single-connection without delay can be seen in Figure 9.6. The figure is clipped after 3000 learning cycles because the convergence behaviour and error measure do not change after that. The depicted number of trials as 3000 is sufficient to learn the input spike patterns in a stable manner. The trend of averaged reward, applied during the learning, has smoother convergence compared to current reward which is not directly applied. The level of reward signal converges to its maximum value of 0.91 during learning cycles. The distance between desired and actual activity move closer to lower values. Reward has inverse relation with van Rossum Distance as depicted in Figure 9.6.

9.6.2 Results of Noiseless Simulations (Multi-Constant-Delay)

Under the noiseless conditions, a desired output signal $S_d^{out}(t)$ and the convergence of the actual output $S_a^{out}(t)$ over training cycles are depicted in Figure 9.7 for multi-constant-delay connections. Figure 9.8 demonstrates the weight trajectories during an example training session. In addition, histogram of weights through noiseless conditions, prepared from the experiment in Figure 9.8, is illustrated in Figure 9.9. Finally, convergence and error trajectory through the learning for multi-constant-delay architecture (with multiple synapses) can be seen in Figure 9.10.

9.6.3 Results of Noisy Simulations (Single-Connection)

Similar to the experiments in section 8.7 and section 8.10, we start with the deterministic model of neurons in noise-free learning conditions through experiments in previous section. In addition, the reliability of the network is also investigated in the face of noise interference

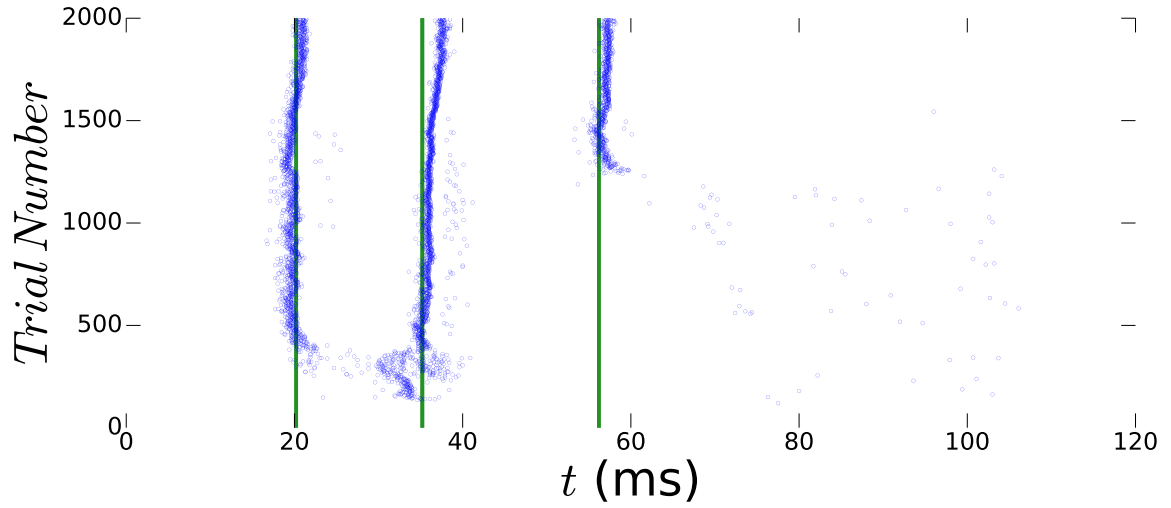


Figure 9.7: Reconstruction of the transformation from input patterns to output spike timings (noiseless, multi-delay). The current network is trained to map a spatio temporally encoded input spike trains into another spatio temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour.

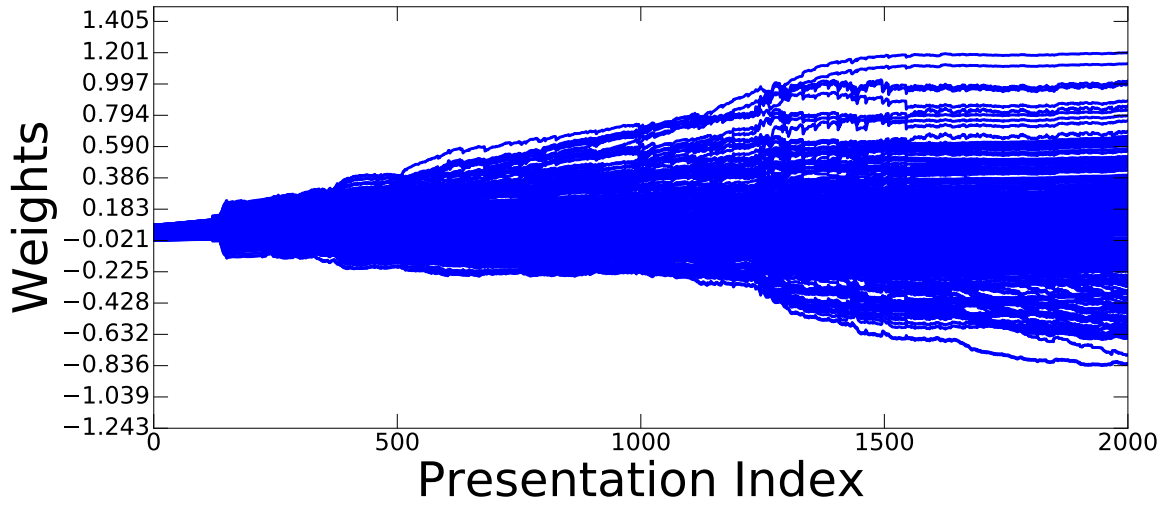


Figure 9.8: Evolution of synaptic weights for the mapping experiment (noiseless, multi-delay). The figure demonstrates entire synaptic weight modifications during a whole training.

during training sessions. Same noise levels in section 8.7 and section 8.10 as low noise and high noise are considered. The details of the inserted noise and how they affect neuronal activity are detailed in section 7.3. The characteristics of low noise and high noise are depicted in Figure 7.1 and in Figure 7.2, respectively.

For the low noise, the desired output signal $S_d^{out}(t)$ and the convergence of the actual output $S_a^{out}(t)$ over training cycles are depicted in Figure 9.11 for single-connection without delays.

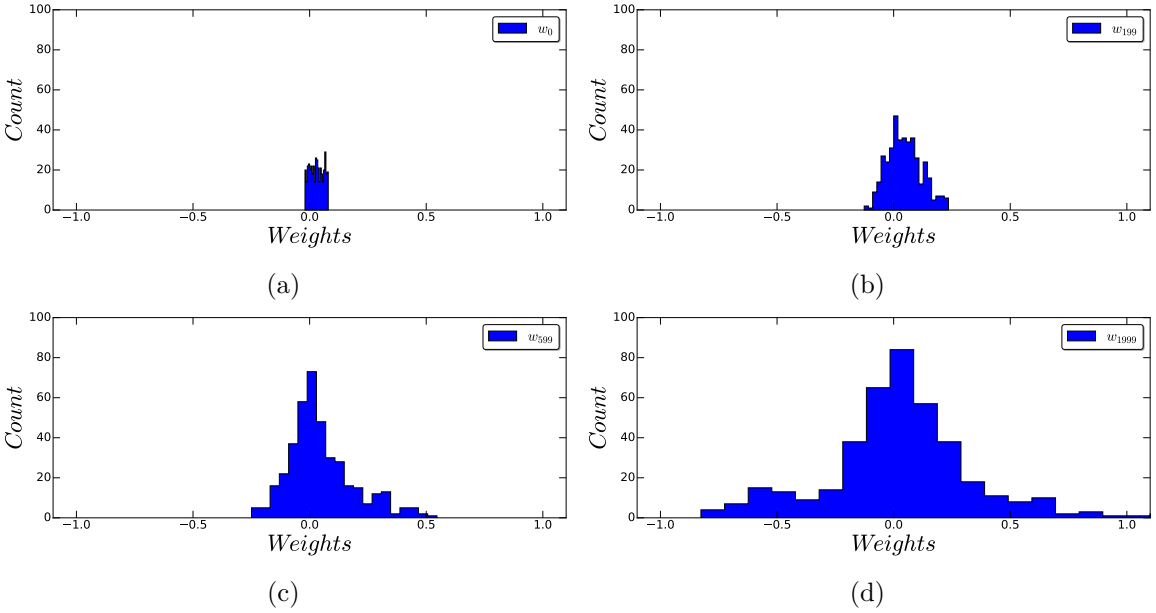


Figure 9.9: Histogram of weights is prepared from the experiment in Figure 9.8 (noiseless, multi-delay). Four different characteristic time points are shown in the figure with legend texts: w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{1999} for 2000^{th} presentation in (d).

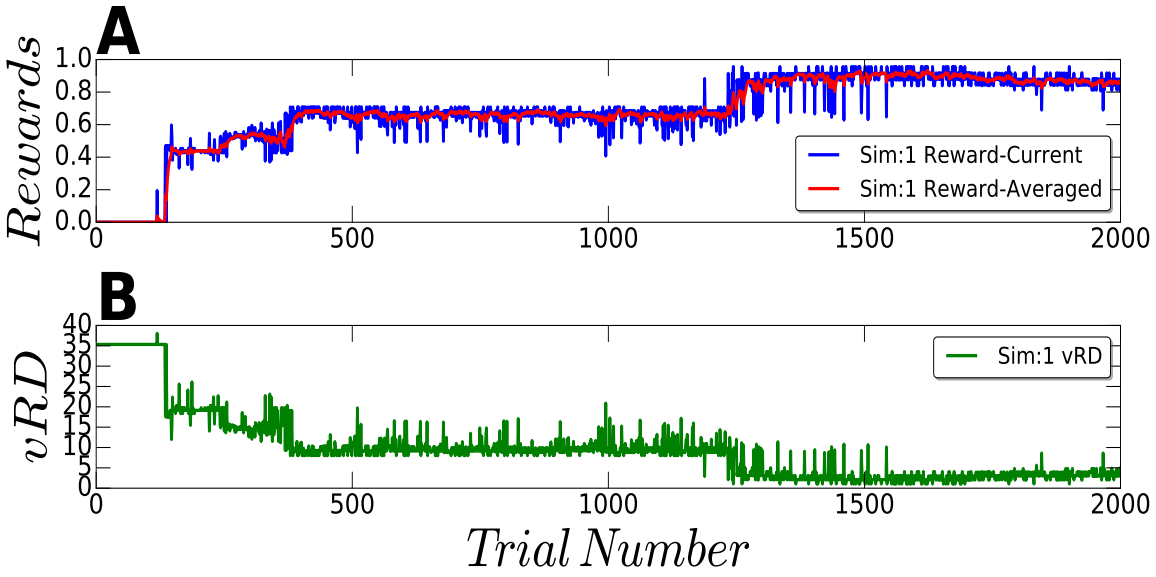


Figure 9.10: The trajectory of current and averaged reward versus van Rossum Distance (vRD) during randomly selected simulations (noiseless, multi-delay). A) A snapshot of averaged rewards (coloured red) with running average of current rewards (coloured blue). The red line shows the average reward time course $r^{avg}(t)$ trial numbers 1 to 3000. The blue line shows the current reward time course $r^{curr}(t)$ over epochs. B) The evolution of mismatch between the desired and the actual output signal, $D_R(S_a, S_d)$ based on the van Rossum Distance.

Figure 9.12 demonstrates the weight trajectories during an example training session. In addition, histogram of weights through low noise conditions, prepared from the experiment in

Figure 9.12, is illustrated in Figure 9.13. Finally, convergence and error trajectory through the learning for single-connection (without delays) can be seen in Figure 9.14.

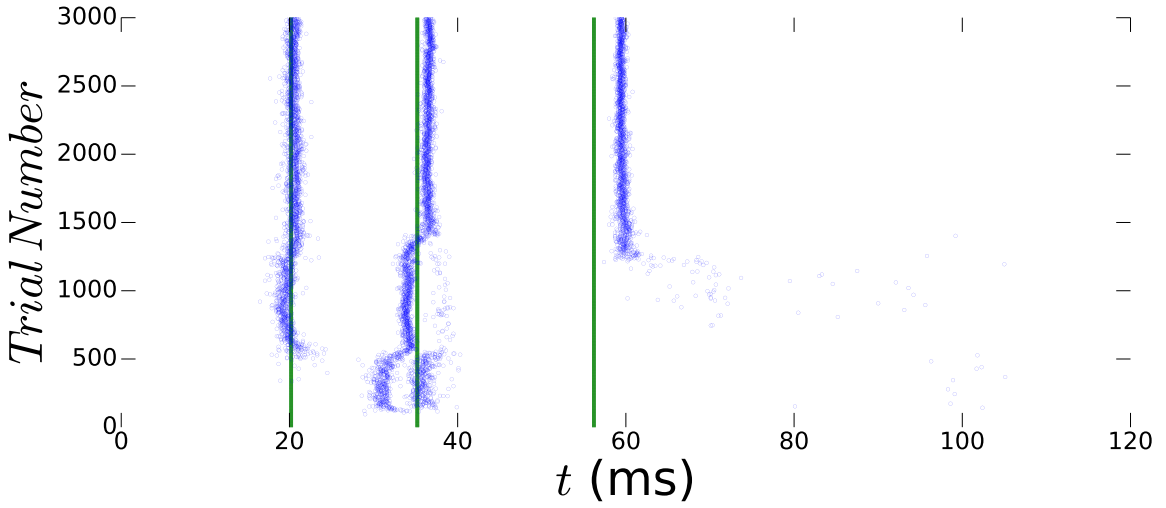


Figure 9.11: Reconstruction of the transformation from input patterns to output spike timings (low noise, without delay). The current network is trained to map a spatio temporally encoded input spike trains into another spatio temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour.

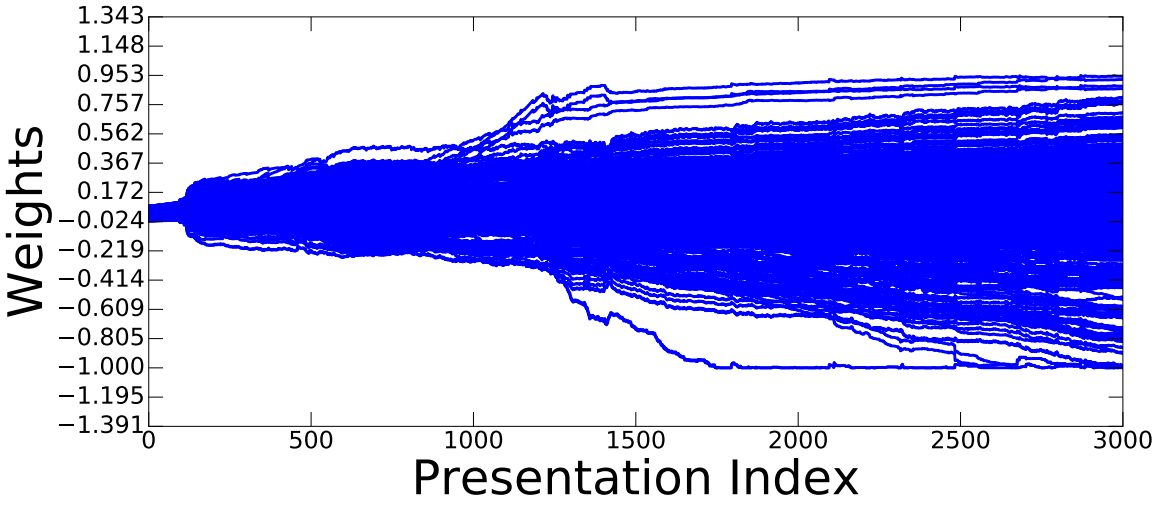


Figure 9.12: Evolution of synaptic weights for the mapping experiment (low noise, without delay). The figure demonstrates entire synaptic weight modifications during a whole training.

For the high noise, the desired output signal $S_d^{out}(t)$ and the convergence of the actual output $S_a^{out}(t)$ over training cycles are depicted in Figure 9.15 for single-connection without delays. Figure 9.16 demonstrates the weight trajectories during an example training session. In addition, histogram of weights through high noise conditions, prepared from the experiment in Figure 9.16, is illustrated in Figure 9.17. Finally, convergence and error trajectory through

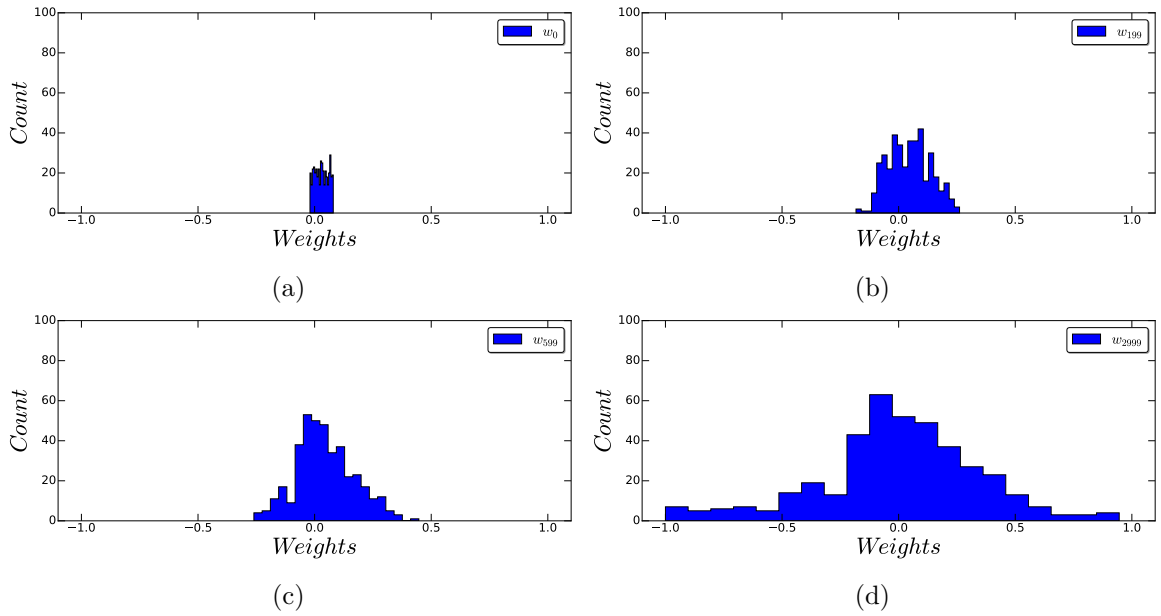


Figure 9.13: Histogram of weights is prepared from the experiment in Figure 9.12 (low noise, without delay). Four different characteristic time points are shown in the figure with legend texts: w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{2999} for 3000^{th} presentation in (d).

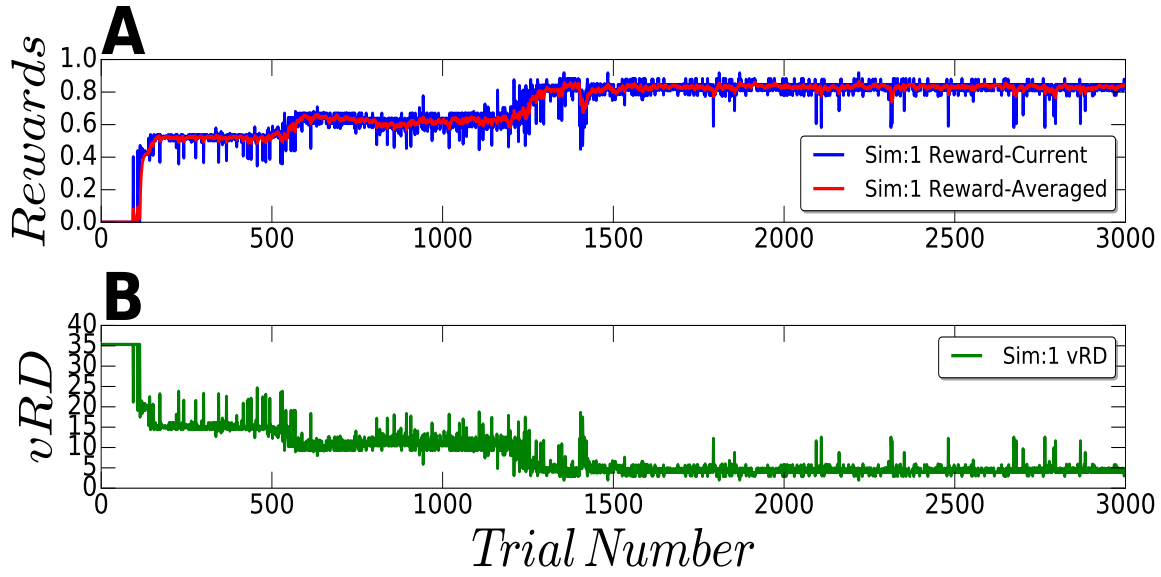


Figure 9.14: The trajectory of current and averaged reward versus van Rossum Distance (vRD during randomly selected simulations) (low noise, without delay). A) A snapshot of averaged rewards (coloured red) with running average of current rewards (coloured blue). The red line shows the average reward time course $r^{avg}(t)$ trial numbers 1 to 3000. The blue line shows the current reward time course $r^{curr}(t)$ over epochs. B) The evolution of mismatch between the desired and the actual output signal, $D_R(S_a, S_d)$ based on the van Rossum Distance.

the learning for single-connection (without delays) can be seen in Figure 9.18.

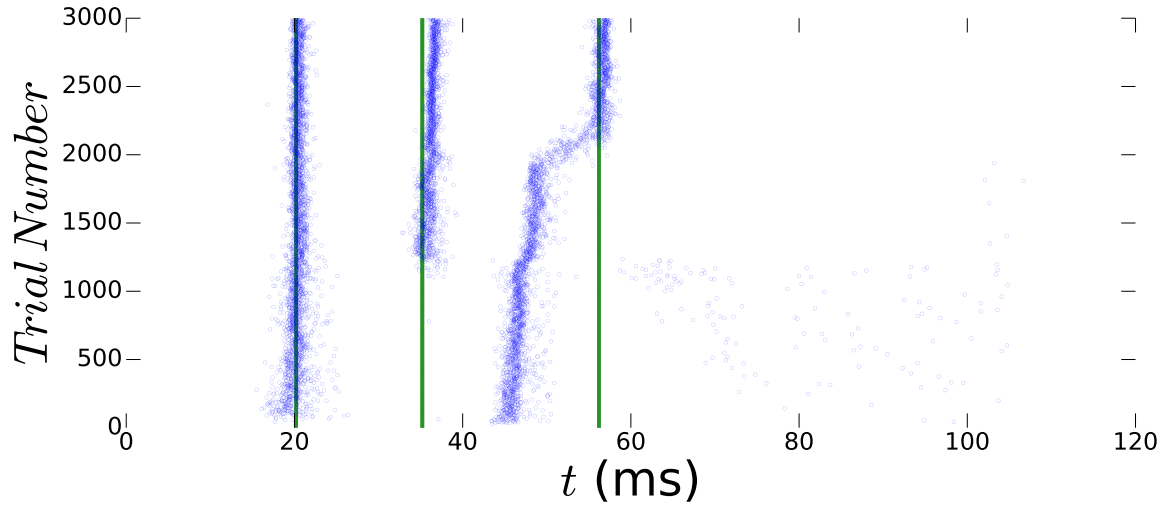


Figure 9.15: Reconstruction of the transformation from input patterns to output spike timings (high noise, without delay). The current network is trained to map a spatio temporally encoded input spike trains into another spatio temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour.

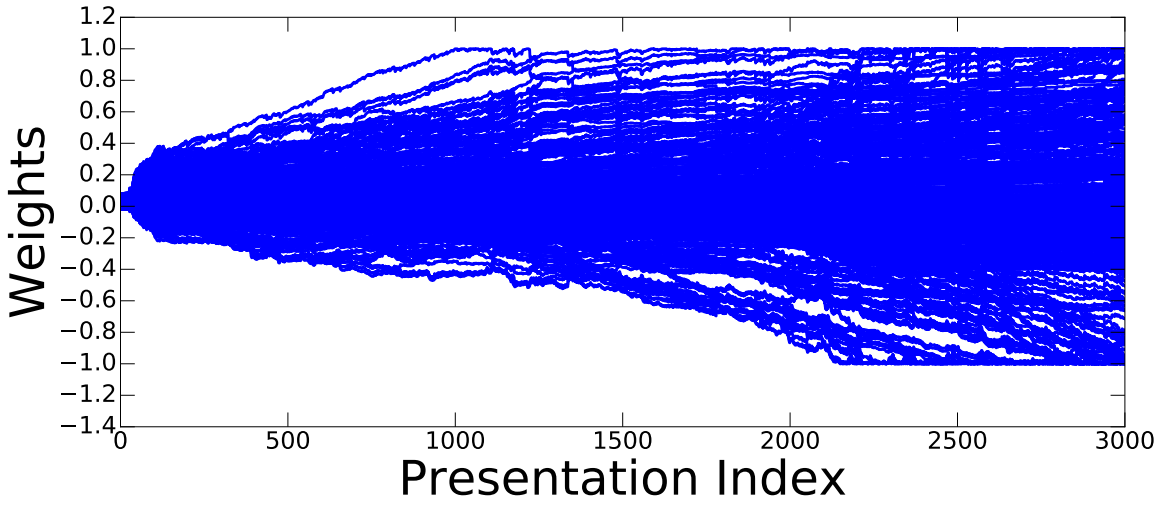


Figure 9.16: Evolution of synaptic weights for the mapping experiment (high noise, without delay). The figure demonstrates entire synaptic weight modifications during a whole training.

In the following sections, the simulations are also shown under noise to demonstrate any detrimental effect that multi-constant-delay architecture may have. This helps to verify the reliability and robustness of the proposed architecture compared to the architecture of single-connection without delay through R-STDP.

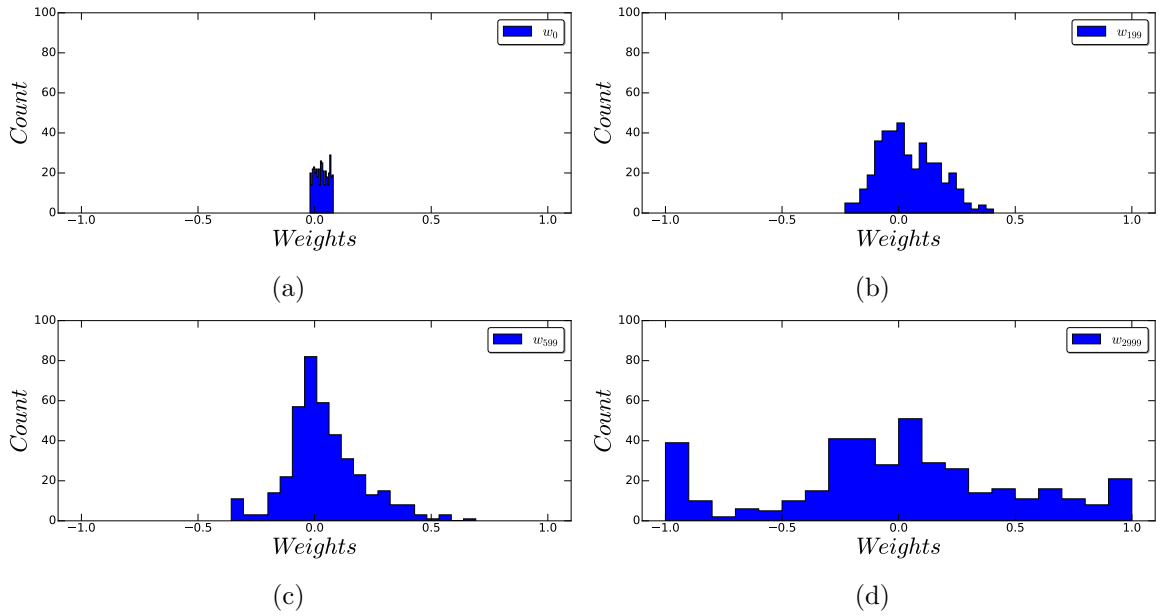


Figure 9.17: Histogram of weights is prepared from the experiment in Figure 9.16 (high noise, without delay). Four different characteristic time points are shown in the figure with legend texts: w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{2999} for 3000^{th} presentation in (d).

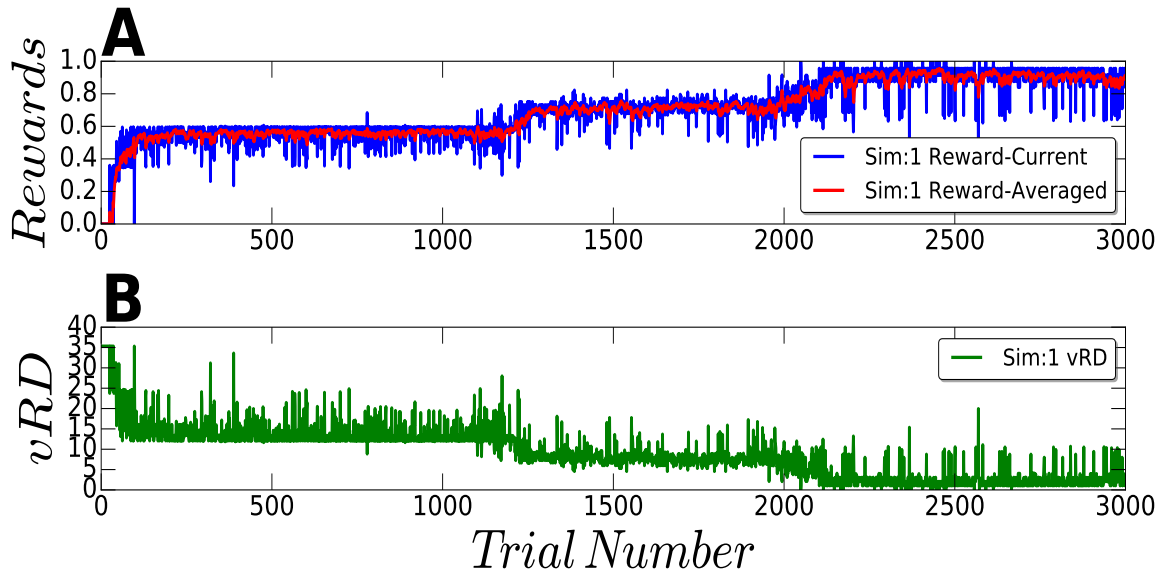


Figure 9.18: The trajectory of current and averaged reward versus van Rossum Distance (vRD (high noise, without delay). A) A snapshot of averaged rewards (coloured red) with running average of current rewards (coloured blue). The red line shows the average reward time course $r^{avg}(t)$ trial numbers 1to 3000. The blue line shows the current reward time course $r^{curr}(t)$ over epochs. B) The evolution of mismatch between the desired and the actual output signal, $D_R(S_a, S_d)$ based on the van Rossum Distance.

9.6.4 Results of Noisy Simulations (Multi-Constant-Delay)

The same cases are also executed for multiple synapses with multiple delays under the low noise and high noise. For the low noise, the desired output signal $S_d^{out}(t)$ and the convergence of the actual output $S_a^{out}(t)$ over training cycles are depicted in Figure 9.19 for multi-constant-delay. Figure 9.20 demonstrates the weight trajectories during an example training session. In addition, histogram of weights through low noise conditions, prepared from the experiment in Figure 9.20, is illustrated in Figure 9.21. Finally, convergence and error trajectory through the learning for multi-constant-delay (with multiple synapses) can be seen in Figure 9.22.

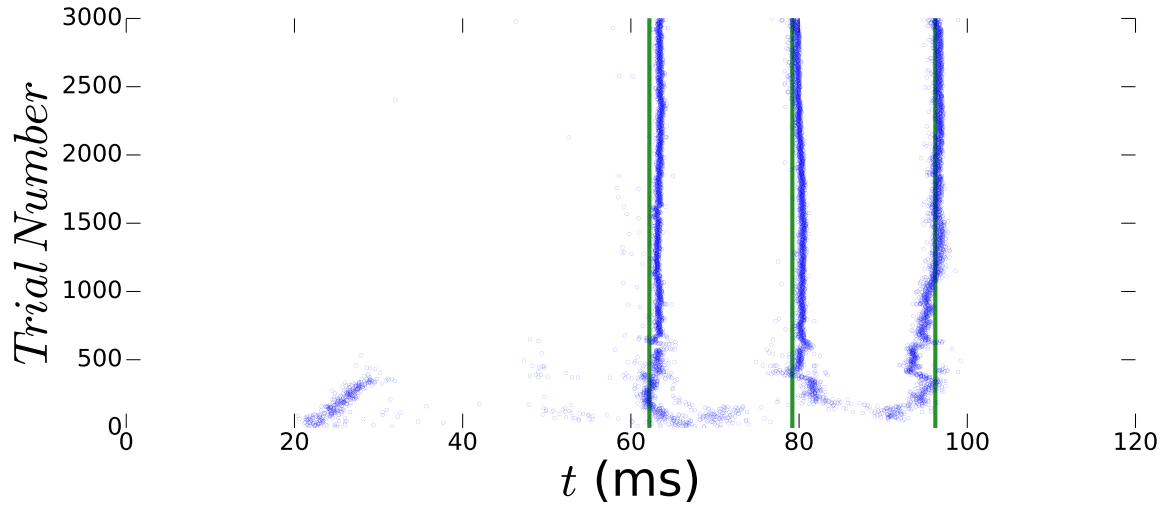


Figure 9.19: Reconstruction of the transformation from input patterns to output spike timings (low noise, multi-delay). The current network is trained to map a spatio temporally encoded input spike trains into another spatio temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour.

For the high noise, the desired output signal $S_d^{out}(t)$ and the convergence of the actual output $S_a^{out}(t)$ over training cycles are depicted in Figure 9.23 for multi-constant-delay. Figure 9.24 demonstrates the weight trajectories during an example training session. In addition, histogram of weights through high noise conditions, prepared from the experiment in Figure 9.24, is illustrated in Figure 9.25. Finally, convergence and error trajectory through the learning for multi-constant-delay (with multiple synapses) can be seen in Figure 9.26.

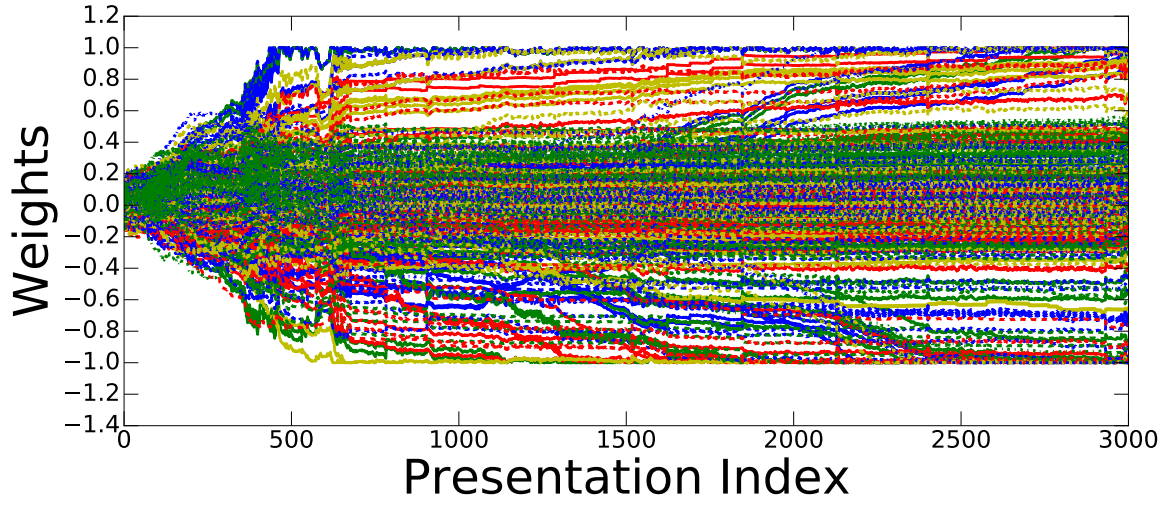


Figure 9.20: Evolution of synaptic weights for the mapping experiment (low noise, multi-delay). In each plot individual weights are represented in a randomly selected colour. The figure demonstrates entire synaptic weight modifications during a whole training.

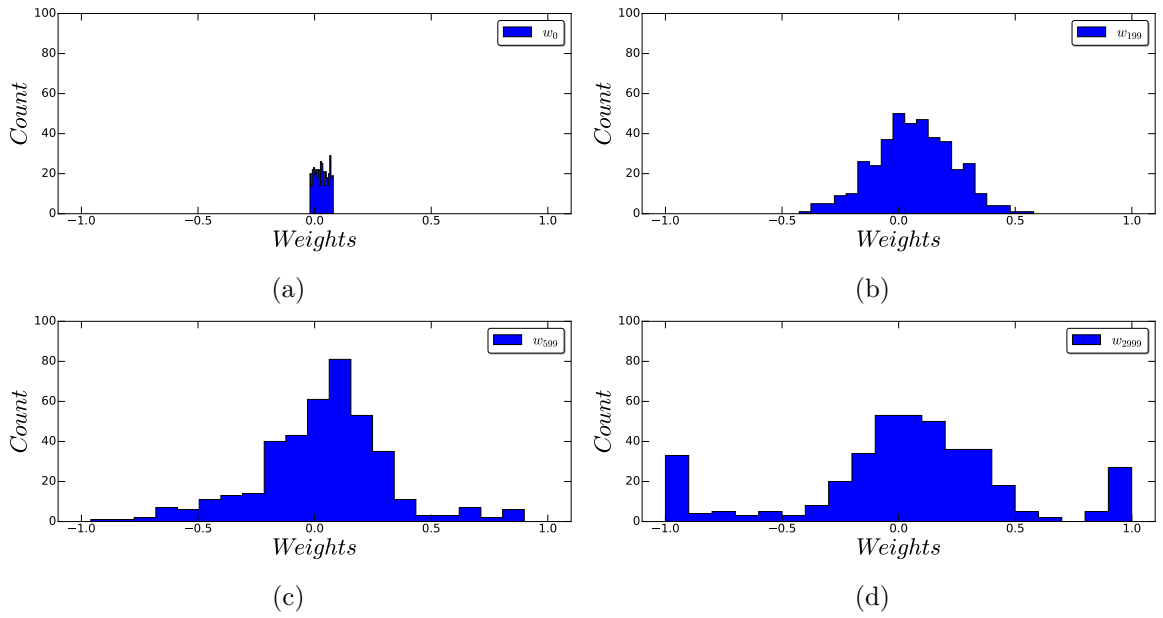


Figure 9.21: Histogram of weights is prepared from the experiment in Figure 9.20 (low noise, multi-delay). Four different characteristic time points are shown in the figure with legend texts: w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{2999} for 3000^{th} presentation in (d).

9.6.5 Discussion

The magnitude and direction of synaptic update is determined by the relative timing of pre- and post-synaptic firings using Reward-modulated Spike-timing Dependent Plasticity. However, one of the common problems for Supervised Hebbian paradigms: Synaptic weights continue to adjust their parameters although the desired spike timings are achieved by applying

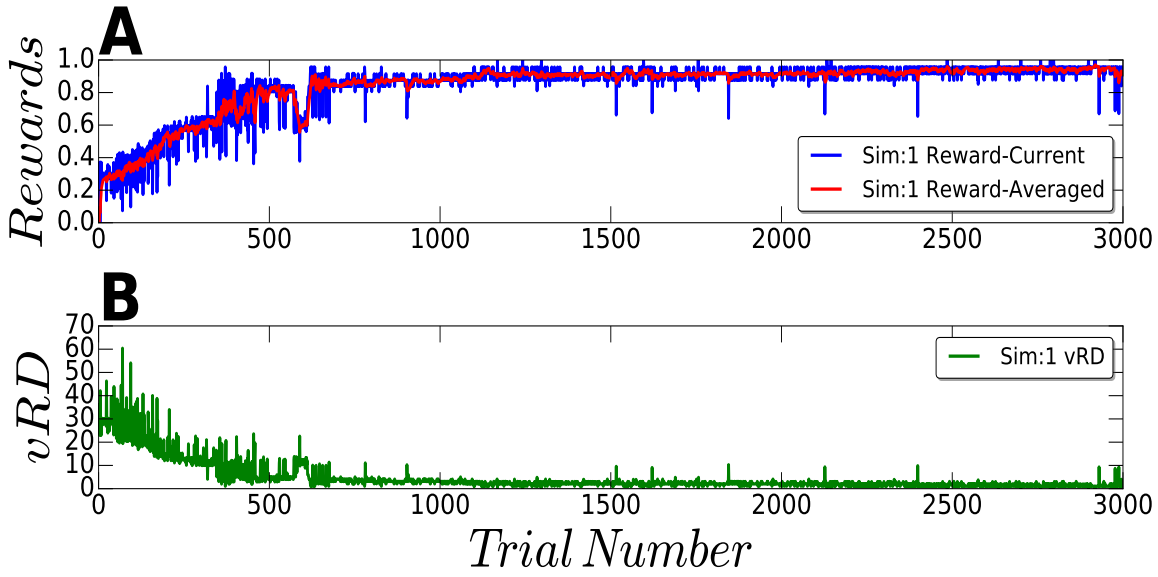


Figure 9.22: The trajectory of current and averaged reward versus van Rossum Distance (vRD during randomly selected simulations) (low noise, multi-delay). A) A snapshot of averaged rewards (coloured red) with running average of current rewards (coloured blue). The red line shows the average reward time course $r^{avg}(t)$ trial numbers 1 to 3000. The blue line shows the current reward time course $r^{curr}(t)$ over epochs. B) The evolution of mismatch between the desired and the actual output signal, $D_R(S_a, S_d)$ based on the van Rossum Distance.

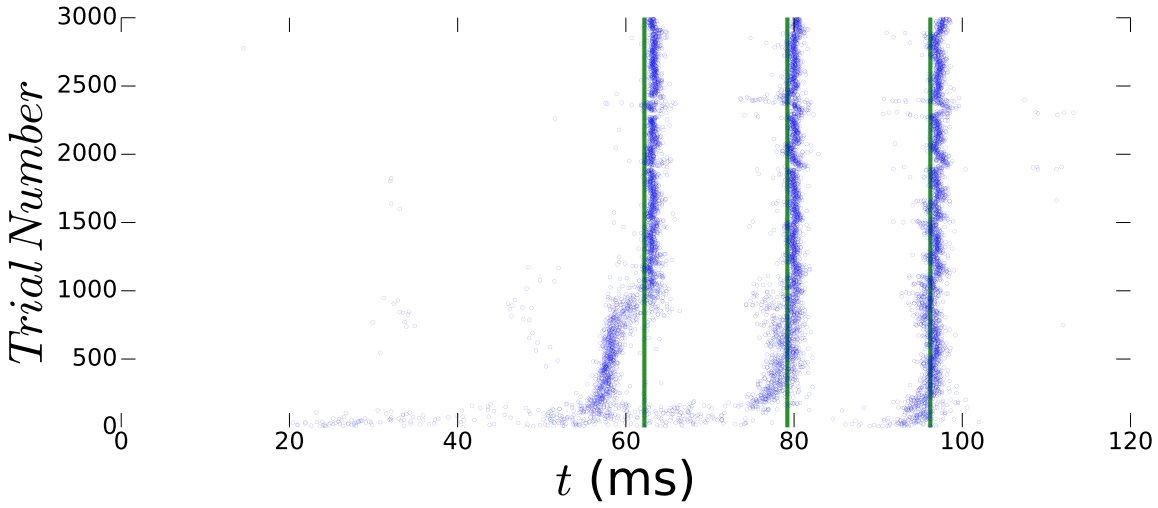


Figure 9.23: Reconstruction of the transformation from input patterns to output spike timings (high noise, multi-delay). The current network is trained to map a spatio temporally encoded input spike trains into another spatio temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour.

a global modulator reward signal. This issue has been eliminated through modulator δ_R. Once actual and desired spike timings are quite close or the same, the reward is maximized and the error is minimized seen in Figure 9.27, Figure 9.28. Hence, current weights are no longer be modified because of the minimized or zero factor of error. This factor stabilizes the

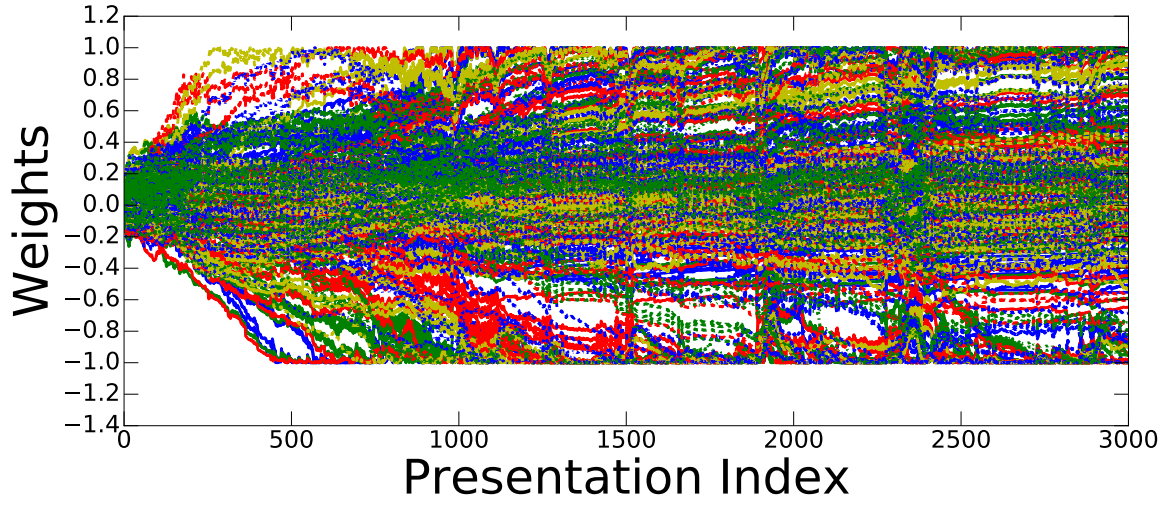


Figure 9.24: Evolution of synaptic weights for the mapping experiment (high noise, multi-delay). In each plot individual weights are represented in a randomly selected colour. The figure demonstrates entire synaptic weight modifications during a whole training.

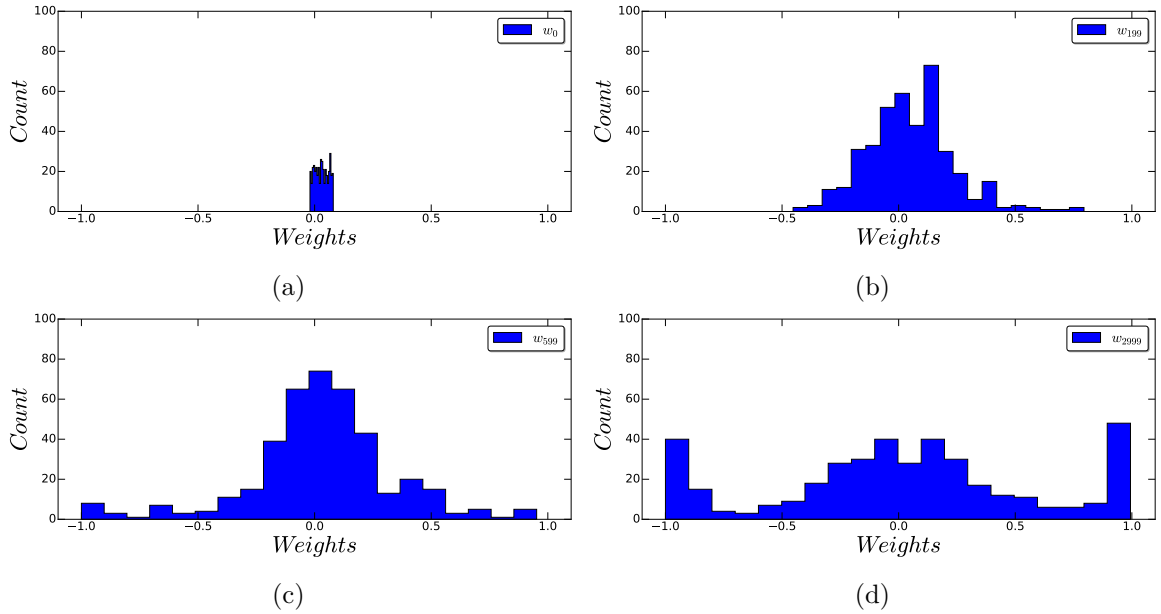


Figure 9.25: Histogram of weights is prepared from the experiment in Figure 9.24 (high noise, multi-delay). Four different characteristic time points are shown in the figure with legend texts: w_0 for 1^{st} presentation in (a), w_{199} for 200^{th} presentation in (b), w_{599} for 600^{th} presentation in (c), w_{2999} for 3000^{th} presentation in (d).

network activity as demonstrated experimentally.

In this experiment, we have explored the approach of Spiking Neural Networks to perform a spatio-temporal mapping task. In the learning for spiking neurons, the reward mechanism has been used as a modulator signal through a STDP window relevant to biological observations.

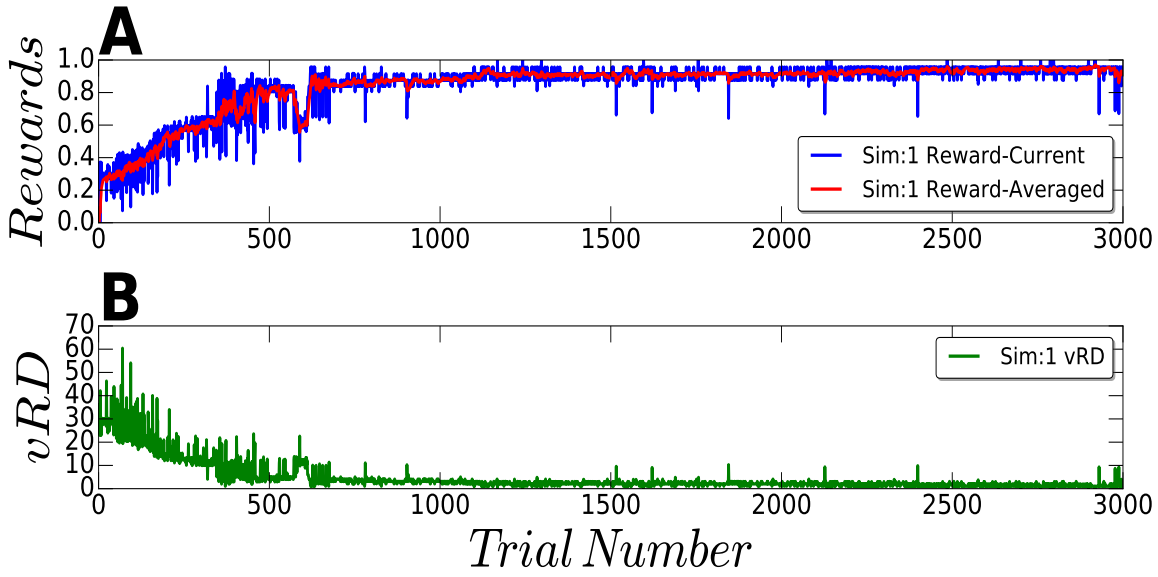


Figure 9.26: The trajectory of current and averaged reward versus van Rossum Distance (vRD during randomly selected simulations) (high noise, multi-delay). A) A snapshot of averaged rewards (coloured red) with running average of current rewards (coloured blue). The red line shows the average reward time course $r^{avg}(t)$ trial numbers 1 to 3000. The blue line shows the current reward time course $r^{curr}(t)$ over epochs. B) The evolution of mismatch between the desired and the actual output signal, $D_R(S_a, S_d)$ based on the van Rossum Distance.

The learning neuron can reach a high accuracy within 1200-1500 training iterations. The learning neuron responded to the given stimulus by generating an output spike train instead of the timing of single firing. Multi-spike timings also increase the number of possible maximum output patterns. Furthermore, each neuron receives partial information through external selection in the encoding scheme. The encoding period of the spike trains is chosen to be on a scale of hundreds of milliseconds which matches the biological observations (Panzeri et al., 2010; Butts et al., 2007).

In R-STDP experiments, we start with the choice of balanced windows between the pre-before-post and post-before-pre sides similar to chapter 8. The balance detailed in subsection 3.4.2 is handled by setting the LTD/LTP ratio to 1.0 with the choice of parameters: A_{pre}, τ_{pre}, A_{post}, and τ_{post} (see Table 9.2). However, we have observed that ignoring LTD side rather than a balanced windows with LTD and LTP performs better for multi-constant-delay architecture as well similar to the observations for single-connection architecture (Fremaux et al., 2010). This also strengthens the link between our R-STDP implementation and R-max rule. Hence, entire simulations in this chapter use only LTP part in STDP windows.

Two delay mechanisms are compared in the learning framework for feed-forward Spiking Neural Network by Reward-modulated Spike-timing Dependent Plasticity. Reward-modulated Spike-timing Dependent Plasticity is used which relies on both Hebbian plasticity modulated by reward and synaptic eligibility traces as transient memory of past Hebbian events in each individual synapses. The reward signal for each episode is derived from comparison of the outputs of the actual neuron and desired spiking times. Thus, R-STDP synapses can drive the required activity changes despite delayed reinforcement signal. Only delayed rewards are taken into account because of biological plausibility (Schultz, 1998; Pawlak & Kerr, 2008).

There are two separate setups in order to compare single-connection without the delay mechanism and with the delay mechanism through R-STDP. In both cases, the total number of synapses N_{syn} for the network is the same. We have also compared robustness and performance of the network based on mapping benchmarks under noiseless, low noise and high noise conditions.

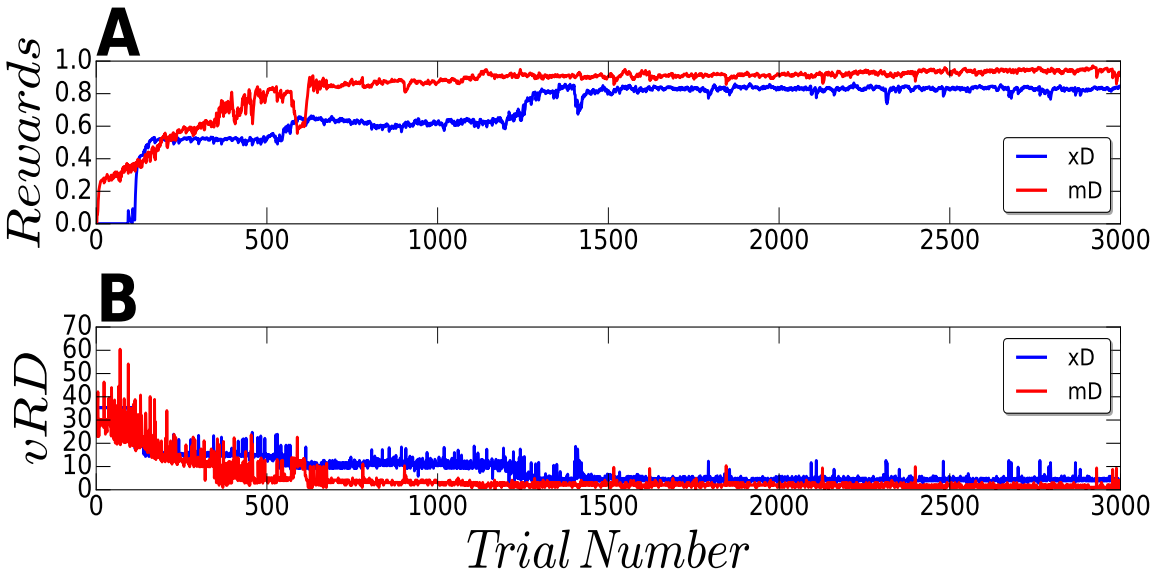


Figure 9.27: The trajectory of current and averaged reward versus van Rossum Distance (low noise, single connection and the multi-delay comparison). Red lines with the legend of mD and blue lines with the legend of xD indicate multi delays and single connection without delays, respectively. A) A snapshot of averaged rewards of current rewards for multi delays (mD) and for single connection without delays over epochs. B) The evolution of mismatch between the desired and the actual output signal, $D_R(S_a, S_d)$ based on the van Rossum Distance for the multi-delay and single connection without delays.

While the learning is performed in R-STDP, the distribution covers the full range of avail-

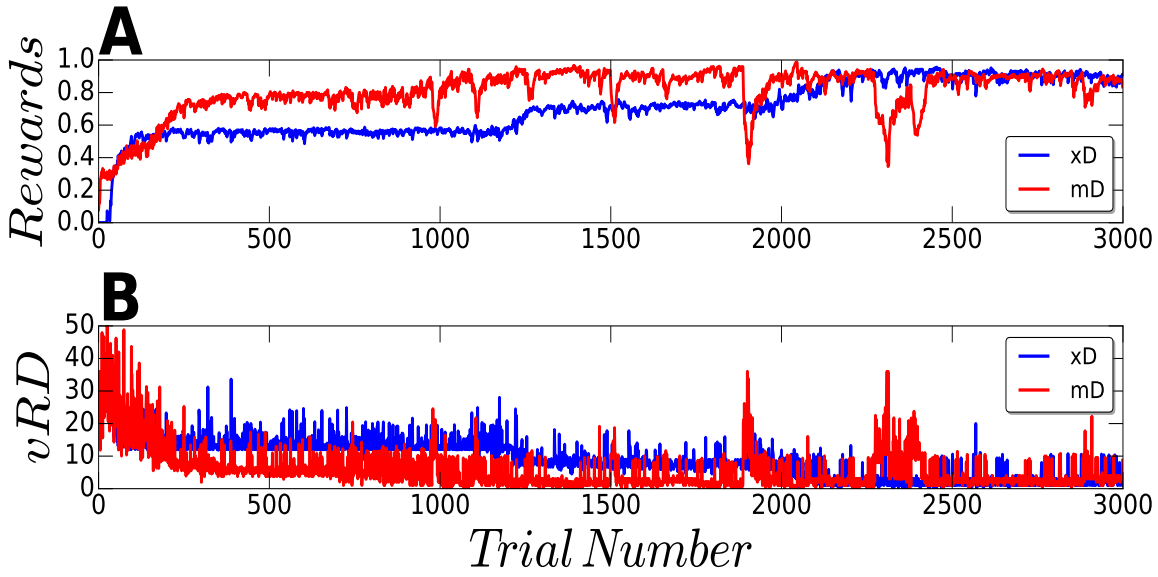


Figure 9.28: The trajectory of current and averaged reward versus van Rossum Distance (high noise, single connection and the multi-delay comparison). Red lines with the legend of mD and blue lines with the legend of xD indicate multi delays and single connection without delays, respectively. A) A snapshot of averaged rewards of current rewards for multi delays (mD) and for single connection without delays over epochs. B) The evolution of mismatch between the desired and the actual output signal, $D_R(S_a, S_d)$ based on the van Rossum Distance for the multi-delay and single connection without delays.

able weights seen through selected four time points similar to the ReSuMe and DelReSuMe experiments. It is clear that all synaptic strengths get closer to stable values during R-STDP simulations in the both structures. The values of synaptic weights at the end of the successful trainings can be intermediate values between weight boundaries w_{min}, w_{max} rather than all converging to w_{min} and w_{max} (see subsection 3.4.2). The shapes of synaptic weight distributions in noiseless and low noise cases are the unimodal similar to the ReSuMe and DelReSuMe experiments. However, once the level of noise is increased for R-STDP, synaptic weights are more likely to achieve its weight boundaries in both mechanisms. This can be seen clearly in Figure 9.17 and Figure 9.25. This is not the case for the ReSuMe and the DelReSuMe; because the shape of weight distributions is still unimodal under the high noise through the ReSuMe and the DelReSuMe.

Faster convergence and better learning in multi-constant-delay structure compared to single-connection can be seen under the low noise and high noise in Figure 9.27 and Figure 9.28, respectively. The comparison of noiseless conditions is quite similar to the low noise performance; hence, it is not drawn repeatedly. Inserting noise into the experiments causes

shifting of actual spiking times compared to desired spiking times similar to the experiments in section 8.7.2 and section 8.10.2. However, multi-constant-delay structure fluctuates more for the averaged reward and van Rossum Distance especially on the impact of high noise. This reflects more reduced reliability caused by high noise for multi-constant-delay structure compared to single-connection without delay.

9.7 Summary

This chapter presents an architecture of spiking neurons with multi-constant-delay mechanism through Reward-modulated Spike-timing Dependent Plasticity with several experimental results based on spatio-temporal mappings. In R-STDP, synaptic modifications only occur if there is a release of reward within a determined time window. The trained memory is formed through synaptic weight modification during learning cycles to respond in a temporally precise manner.

Throughout the chapter, we propose a two-layer SNN with multiple delays between the input-output neuron pairs. The simple Leaky-Integrate-and-Fire spiking neuron model with two different modelling parameters is used in this chapter. At the level of Neural Networks, actor-critic topology is proposed, including a bias layer.

We test the proposed architecture using the learning rule of R-STDP on a set of mapping tasks. The proposed network architecture with multi-synaptic connection is not dependent only the learning rule of Reward-modulated Spike-timing Dependent Plasticity, it can be also transferred to the other reward modulated rules such as Reward-maximisation rule. The proposed mechanism of delays helps to perform the task well with less spiking input neurons and fewer input patterns than direct connections without delays in R-STDP. Experimental results show the learning capability and performance of the proposed mechanism. It has higher accuracy for temporal sequences of three spikes in 100 ms. Also, the introduced learning mechanism is able to learn to map input patterns into output pattern with multiple timings around 1200-1500 learning cycles (presentations) under noiseless conditions. For the similar tasks, ReSuMe requires around 1400-1500 training steps (presentations) in subsection 8.7.1, DelReSuMe requires around 300-350 training steps (presentations) in subsection 8.10.1.

There are two separate setups in order to compare single-connection without any synaptic delay and with multi-constant-delay mechanism. In both cases, the total number of synapses N_{syn} for the network is equal. Also we demonstrate that the form of multi-constant-delay structure has better convergence speed under noiseless and realistically noisy conditions. However, its robustness to the effect of relatively high noise is less compared to single-connection structure.

The model developed in this chapter primarily focuses on the multiple firing times during the encoding time frame both in the input layer and the output layer. Biological plausibility of the proposed approach is one of the primary aspects in our study using temporal encoding, plasticity mechanisms and multi-spike coding in input-output layers.

Chapter 10

Conclusion - Contributions and Future Research

Contents

10.1 Conclusion and Contributions 251
10.2 Future Research 257

The final chapter of this thesis describes the original contributions including theoretical and practical issues in section 10.1 and future works in section 10.2 are specified.

10.1 Conclusion and Contributions

Various achievements have been made during the course of this research. Let us begin with an overview of the physiology of the neuron and its mathematical model in chapter 2. Then, the relation between biological and artificial neurons is presented with the structural and functional inspirations by neurbiological findings. A brief history from a standard Artificial Neural Network (ANN) to biological Spiking Neural Network (SNN) is provided. Then, three main types of learning procedures as supervised learning, unsupervised learning and Reinforcement Learning are discussed. Example implementations of various approaches are reviewed in order to model the neuron transfer function such as Hodgkin-Huxley (HH) model, Integrate-and-Fire (IF) neuron model, Leaky-Integrate-and-Fire (LIF) neuron model, Spike Response Model (SRM) and Izhikevich model (IM). An overview of some common simulators are also given, focusing on the Brian simulator used throughout the thesis, for Spiking Neural Networks.

In spike train based plasticity rules in chapter 3, the temporal order of pre-synaptic and post-synaptic firing times can determine synaptic potentiation (Long Term Potentiation - LTP) vs. depression (Long Term Depression - LTD). A pre-synaptic spike arriving before a post-synaptic spike leads to synaptic potentiation, arrival after the post-synaptic spike activity causes depression of the synaptic connection (Bohte, 2004). This temporal relationship between pre- and post-synaptic activity is a paradigm derived from neurobiological experiments and it is known as Spike-timing Dependent Plasticity (STDP) which is experimentally observed in hippocampal neurons (Bi & Poo, 1998) discussed in chapter 3. In the thesis, several forms of STDP with time-dependent weight changes are used as a biologically plausible mechanism in order to map input spike patterns onto output spike patterns through three algorithms: Remote Supervised Method (ReSuMe) in section 8.3, Delayed Remote Supervised Method (DelReSuMe) in section 8.9, and Reward-modulated Spike-timing Dependent Plasticity (R-STDP) in chapter 9. Beyond Spike-timing Dependent Plasticity, homeostatic plasticity with the mechanism of synaptic scaling as a further process in regulating neuronal functions in the network is discussed. Also, Dopamine-modulated plasticity as an important modulations of synaptic plasticity is described with its biological background.

An important goal of chapter 4 is to describe the computational foundations for dealing with problems like states and actions in the context of Reinforcement Learning mechanism which is a particular branch of Machine Learning. Furthermore, a number of underlying concepts for the following chapters such as Temporal-Difference mechanism as an attractive formulation of RL is examined in this chapter. For instance, Reward-modulated Spike-timing Dependent Plasticity in chapter 9 as a biologically plausible learning for a network of spiking neurons is derived from the continuous Temporal-Difference formulation detailed in this chapter. We have also introduced the general overview of the three-factor learning rule in Spiking Neural Networks. The regulated form of original associative plasticity by Dopamine as the three-factor rule is summarized for the basis of synaptic plasticity and memory formation in chapter 9. In addition, the mechanism of eligibility trace, a transient memory of past events, through Temporal-Difference is examined in the context of RL. The block of eligibility trace, is the fundamental component in the proposed spiking learning in chapter 9.

In the beginning of chapter 5, Temporal-Difference learning is implemented on a maze task. Then, a set of novel rules in order to improve exploration in Reinforcement Learning, fo-

cusing on Temporal-Difference learning, are proposed using knowledge-based approaches. Experiments described in this chapter demonstrate the improvements on the exploration efficiency of the Reinforcement Learning algorithm for a maze navigation task. Although it is tested with one important RL algorithm, Temporal-Difference learning, it can be applied to any exploration algorithm for faster learning. We have also prepared a simulation platform which enables us to test various algorithms on the maze environment in chapter 4. Selectable techniques from a Graphical User Interface developed are two wall follower algorithms: LSR rule and RSL rule, Temporal-Difference learning algorithm, and knowledge-based learning (Replacement Rules) with the combination of LSR or RSL or Temporal-Difference learning. This provides a conceptual simulation framework for further path navigation models as well. All investigations for the maze task are prepared through this framework. In this chapter, the performance of TD learning for the maze task is successfully improved by producing new set of extraction rules. We introduce a set of novel Replacement Rules as Extended Replacement Rules for LSR (ExtRepLSR) and Extended Replacement Rules for RSL (ExtRepRSL) in section 5.3 for maze environments with a remarkable performance compared to LSR rule or RSL rule itself, also to previously offered set of rules from Venkata et al. (2011) (BscRepLSR).

In order to give overview about designing SNN experiments for further part of the thesis, network topologies, focusing on SNNs, are summarized in chapter 6. Spike train notation for generation of artificial spike trains throughout the performed experiments is described here. The encoding mechanisms for Spiking Neural Networks as rate coding, temporal coding and population coding are discussed here. Furthermore, several spike distance metrics in order to measure difference between spike trains are presented: Victor & Purpura Distance, Coincidence Factor, Schreiber Distance, and van Rossum Distance.

The common mechanisms and techniques during experiment setups used in chapter 8 and chapter 9 are described in chapter 7. The types of neuron models used in proposed SNNs are summarized considering the scenario of the presence of noise. Two different noise levels with the standard deviation of the membrane potential are considered as relatively high and low noise which are included as additive term in synaptic input current. In addition, three different delay mechanisms between each pre- and post-synaptic neuron pairs throughout experiments are introduced. They are multiple synaptic connections with constant axonal propagation speed, multiple synaptic connections with plastic axonal propagations, and a

single synaptic connection without axonal delay. The network architectures of spiking neurons for obtaining spatio-temporal experiments during training and testing are demonstrated with the details of bias neuron. The proposed encoding mechanism for spatio-temporal patterns during experiments is presented as well. The adoption of van Rossum Distance (vRD) in order to measure spike-train similarity and a (mis)classification error metric in order to evaluate task performance of the network are detailed. Also, the pseudocodes of training and testing mechanisms throughout the experiments in chapter 8 and chapter 9 are described in section 7.8. Finally, benchmarks used during experiments are introduced in section 7.9.

It is clear that multi-spike timings are more biologically realistic and they are amenable to empirical studies. For biological plausibility, we focus on plasticity rules which handle multi-spike coding. However, only a few learning rules to teach a SNN are proposed which can tackle precise input-output mappings of spike patterns with multiple pulses on each train instead of single spikes in spatio-temporal patterns. Remote Supervised Method (Kasinski & Ponulak, 2006), is the supervised learning of desired spike trains in response to input patterns.

In chapter 8, we start with a review of SpikeProp which is one of the baseline learning algorithms for SNNs. Then Remote Supervised Method (ReSuMe) (Kasinski & Ponulak, 2006), uses a combination of Hebbian-STDP and anti-Hebbian-STDP in a single synapse, is detailed with experimental work. Two benchmarks of mapping and logical operations detailed in section 7.9 are addressed. Mapping tasks use a designed architecture trained to respond to spatio-temporal patterns of input spikes from multiple neurons with a pattern of output spikes in the face of noiseless, relatively low noise and relatively high noise. Logical operation tasks consist of a range of operations: TRUE, AND, OR, and XOR detailed in section 8.8. In addition, a novel connection scheme using the proposed bias neuron is introduced for ReSuMe with its implementation and its mathematical/topological descriptions. It is experimentally proved that the proposed heterosynaptic topology can exactly mimic the ReSuMe weight change through two synapses from input to actual output neuron and desired output neuron.

Although, Remote Supervised Method can be used for classifying input patterns by spatio temporally encoded patterns, it is not reliable and stable enough for information processing in the nervous system (Gruning & Sporea, 2012). For this purpose, we have investigated how to find a better technique for multi-spike coding. Therefore, a novel plasticity mechanism as

an extended version of ReSuMe, named DelReSuMe, is proposed by adding delay learning into weight learning. A novel framework for ReSuMe and DelReSuMe is developed with heterogeneous synapses in chapter 8. The same topology as an alternative interpretation of the remote supervision developed for ReSuMe is also used in DelReSuMe experiments. The learning efficiency of the modified algorithm DelReSuMe is compared to ReSuMe on a series of mapping tasks in the face of noiseless, relatively low noise and relatively high noise. Although ReSuMe generates extra actual spikes compared to desired activity in the further learning cycles under the high noise, DelReSuMe does not generate extra spikes under the same noise level. Therefore, DelReSuMe is more robust to the limited level of high noise compared with ReSuMe. Also DelReSuMe has faster learning speed during not only noiseless but also noisy conditions: low noise and high noise. Another observation is the stabilization level for the synaptic weight values is relatively around smaller values for DelReSuMe compared to ReSuMe. This is caused from plasticity of synaptic delays in addition to the plasticity of synaptic weights in DelReSuMe. Briefly, faster learning and convergence with slightly better accuracy through mapping benchmarks has been achieved with DelReSuMe in section 8.9 compared to ReSuMe in section 8.9.

Various learning types are proposed to comprehend how animals/humans adjust their behaviour, how they perform iterative optimization of skills like riding a bike or playing video games. However, it is widely believed to rely on synaptic plasticity of the brain through the paradigm of "practice makes perfect" (Seung, 2003). Different forms of synaptic plasticity rules might help to optimize the neural structure, if biological plausibility is taken into account. Therefore, in chapter 9, we have proposed a framework of Reward-modulated Spike-timing Dependent Plasticity (R-STDP) for multi-spike-coding in chapter 9. In addition, empirical Dopamine R-STDP is briefly compared with the theoretically derived R-max rule. Through implemented experiments using R-STDP, we have explored the approach of Spiking Neural Networks to perform a spatio-temporal mapping task in the face of noiseless, relatively low noise and relatively high noise conditions. For spiking neurons, the reward mechanism is used as a modulator signal through the STDP window. There are two separate setups in order to compare single connection without synaptic delay and with the multi-delay mechanism. In both cases, the total number of synapses for the network is the same. While the learning is performed in R-STDP, the distribution covers the full range of available weights similar to the ReSuMe and DelReSuMe. All synaptic strengths get closer stable values at mostly intermediate values between weight boundaries in the both structures. The shapes of synaptic

weight distributions in noiseless and low noise cases are the unimodal similar to the ReSuMe and DelReSuMe experiments. However, once the level of noise is increased for R-STDP, synaptic weights are more likely to achieve its weight boundaries in both mechanisms. This is not the case for the ReSuMe and DelReSuMe; because the shapes of weight distributions are still unimodal under the high noise through the ReSuMe and DelReSuMe. Inserting noise into the experiments causes shifting of actual spiking times compared to desired spiking times similar to the ReSuMe and DelReSuMe. Multi-constant-delay structure compared to a single-connection fluctuates more for the averaged reward and the spike distance especially on the impact of high noise. This reflects more reduced reliability caused by high noise for multi-constant-delay structure compared to single-connection without delay. As a future work, the performance of R-STDP can be investigated on other tasks such as logical operations as XOR task and maze task.

We have also developed a framework for the Brian simulator to handle training and testing mechanisms. The proposed simulation framework is detailed in Appendix C. The framework enables us to perform all experiments on SNNs in chapters 8 and 9. It can be reused and extended for future research of various plasticity based on Feed-forward Networks. Most of the parts of our implementation are not dependent on Brian package such as spike generation, data storage, data reading, file/folder organization, results analysis. Although the framework is tested with the previously mentioned algorithm as Spike-timing Dependent Plasticity (STDP) (detailed in section 3.4), Remote Supervised Method (ReSuMe) (detailed in section 8.3), Delayed Remote Supervised Method (DelReSuMe) (detailed in section 8.9), and Reward-modulated Spike-timing Dependent Plasticity (R-STDP) (detailed in chapter 9), other methods/experiments with the help of existing SNN simulators can be set up as well.

One of the novel aspects of this study is the usage of multi-spike timings not only in each input pair, but also in the output spike train to achieve a more biologically realistic scheme. A dopaminergic inspired learning rule combined with STDP using multi-constant-delay mechanism is shown both analytically and through computer experiments to have faster convergence under noiseless and realistically noisy conditions. The development of Reward-modulated Spike-timing Dependent Plasticity with the delay mechanism in chapter 9 with its increased learning speed helps generic learning tasks where a neuron is supposed to respond to input spike patterns with specific output spikes.

To sum up, we propose a set of novel rules for the maze task in order to shorten exploration period. Also, we develop ReSuMe and a novel extension of ReSuMe, DelReSuMe with proposed simulation framework. We compare ReSuMe and DelReSuMe under noiseless and noisy conditions using same mapping benchmarks. It is demonstrated that DelReSuMe has faster learning than ReSuMe for the mapping tasks in both noiseless and noisy conditions. An alternative connection scheme using the proposed bias neuron is introduced for ReSuMe and DelReSuMe with its experimental evidences. In addition, we develop R-STDP for mapping task by combining reward and SNN through actor-critic topology using the multi-delay architecture rather than a single connection without delays. We demonstrate that the proposed multi-delay topology has faster convergence and better learning speed compared to single connection without delay under noiseless, relatively low noise and relatively high noise conditions. All of plasticity/experiments on SNNs are performed considering with multi-spike coding because of their potential and biological plausibility.

10.2 Future Research

The brain is still a great mystery. Lots of questions related with brain function still remain unanswered. However, we hope that this research, which focussed on various biologically plausible plasticity, helps unravel further questions about the learning in the central nervous system. Here, we now detail how our researches that can be extended with further considerations and experiments.

The simulation framework can be further developed to combine not only the Reinforcement Learning technique but also further approaches such as Temporal-Difference learning with Spiking Neural Networks for path navigation tasks. We have only used a reward mechanism rather than a combination of reward and punishment. This can give a better understanding about the domain during exploration. For instance, each detected U-turn point can be labelled as a punishment area by giving maximum negative value as -100 to this point. We have a single target in a single exploration period. This can be increased and the performance of multiple targets can be interesting to analyse. The proposed techniques can be tested on real time robotic path navigation problems.

Various experimental approaches demonstrate that the computational power in SNNs is inherited from biology. Further improvements on those approaches can be explored to perform better than conventional ANNs and Machine Learning techniques considering increased biologically plausible methods. Using the SNN architecture, more complex real-world tasks can be investigated. Therefore, proposed networks and plasticity mechanisms can be tested with real-world tasks such as the computational problem of Simultaneous Localization and Mapping (SLAM) (Thrun, 2002).

Existing Reinforcement Learning algorithms for Spiking Neural Networks have a number of advantages and disadvantages. For instance, the proposed model in Xie & Seung (2004) is restricted by the assumed Poisson firing characteristics of the neurons and the assumption of an instantaneous response of neurons without considering the temporal characteristics of the membrane potential. Another neuron implementation concept for SNN introduces synapses that are "hedonistic" (Seung, 2003). Hedonistic can be described here as "reward seeking" through responding to global reward signal by increasing probabilities of release and failure. If each synapse is hedonistic, the whole network also can be described as a hedonistic network. Seung (2003) demonstrates how those synapses serve the function of optimization using the statement of hedonism. The Replacement Rules combined with RL algorithm is applied into the maze task without SNNs in chapter 4. This combined algorithm can also be applied the task with RL and SNNs in order to increase the hedonistic performance of SNNs.

Reinforcement Learning techniques for Spiking Neural Networks are important and are studied on different tasks (Yamauchi & Beer, 1994; Blynel & Floreano, 2002; Florian, 2003; Seung, 2003; Xie & Seung, 2004; Vasilaki et al., 2009). In those spiking networks, the agent constructs its own internal representation of the environment through learning trials. Each of these studies derives a learning rule for the networks of neurons firing different patterns of spike trains. In this thesis, we also aim to apply Reinforcement Learning techniques into SNNs. We use different approaches such as R-STDP for multi-spike coding, ReSuMe learning and, DelReSuMe. Therefore, further work can consider application of RL and Spiking Neural Networks for maze navigation.

Further tests can be performed with some known benchmarks such as the Iris dataset (Fisher,

1936) against several existing methods. Especially the one newly proposed, DelReSuMe, can be compared with other existing plasticity rules on SNNs.

During the artificial spike generation mechanism, we have only used Poisson distribution with dead time as spike generators. Although we believe that there is no reasons the approach does not work with other spike train distributions, this can be proven with practical work.

The underlying plasticity for ReSuMe and DelReSuMe is Spike-timing Dependent Plasticity. We have used a set of reasonable parameters for STDP. However, various sets of parameters can be compared using network performance metrics. For instance, the effect of the A_{pre}/A_{post} ratio can be analysed for ReSuMe and DelReSuMe. There are already available works for R-STDP for this comparison (Legenstein et al., 2008). Also, other parameters such as τ_m, τ_e, reward mapping factor, discount factor for the averaged reward, the effect of initialization range, learning rate can be analysed further in order to compare performance and maximize the performance. The number of neurons in the input layer can affect the learning performance of the network. This can be investigated in future work. In addition, proposed DelReSuMe with multi-plastic-delay using the delay selection approach can be compared with the performance of the method in Taherkhani et al. (2015) uses a single-connection architecture using the delay shift approach.

ReSuMe, DelReSuMe and R-STDP can be tested with different neuron models, different neuron parameters, different synapse parameters, and different neuron numbers in input and output layers. We have not used hidden layers in our network topologies. This can be interesting to test and compare. Especially, the XOR task which cannot be handled by the current ReSuMe and DelReSuMe with two layers, it may be worthwhile to try XOR with a hidden layer topology. The effect of introducing hidden layers into operations performed successfully with two layers can be analysed to investigate whether the performance can further improve. The proposed framework is flexible to handle those tests. In addition, more practical tasks such as controlling a robot or path planning in a maze task can also be considered.

We have proposed an extended version of ReSuMe, DelReSuMe, which modifies both synaptic weights and delays, there are various potential works to do. As it is mentioned in section 8.9, we modify each synaptic delay once during a training epoch in order to avoid instability

of the network. Synaptic weights are continuously modified during entire training. However, synaptic delays can be continuously modified parallel to synaptic weights for faster convergence compared with DelReSuMe. The instability of the network in the continuous update can also be resolved by adding another term here, such as the reward factor in chapter 9.

The framework has been tested with the fixed network topology as Feed-forward Network. Although the mechanism is not dependent on the topology, it is worthwhile to test the structure with alternative topologies. The framework is tested with Brian simulator locally. However, an extended package for Brian simulator is prepared for public release. On the other hand, the details of future work about the proposed software tool and implementation side is described in appendix C.14.

To sum up, we have considered a number of options for future work combining Reinforcement Learning, Spike-timing Dependent Plasticity and Spiking Neural Networks. There are many important ways this work can be extended in the future.

Appendix A

Functions Used

Contents

A.1 Dirac Delta Function **261**
A.2 Alpha Function **261**
A.3 Signum Function **262**
A.4 Heaviside Step Function **262**
A.5 Truth Tables for Used Operations **262**
A.6 LCE Calculation Formula **263**

A.1 Dirac Delta Function

$$\delta(t) = \begin{cases} \infty, & \text{if} \quad t = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{A.1}$$

the integral of the impulse function $\delta(t)$ is one:

$$\int_{-\infty}^{+\infty} \delta(t) = 1 \tag{A.2}$$

A.2 Alpha Function

Effective potential after a single pre-synaptic spike with α function can be described as:

$$\alpha(t) = \begin{cases} 0 & \text{for} \quad t \leq t^f \\ \frac{1}{\tau_s} e^{\frac{-(t-t^f)}{\tau_s}} & \text{for} \quad t > t^f \end{cases} \tag{A.3}$$

where t^f is the firing time of the pre-synaptic neuron, τ_m is the time constant of membrane and τ_s is the synaptic time constant.

Spike response function can be described as:

$$\epsilon(t) = \begin{cases} 0 & \text{for} \quad t \leq t^f \\ \frac{1}{1-(\tau_s/\tau_m)}(e^{\frac{-(t-t^f)}{\tau_s}} - e^{\frac{-(t-t^f)}{\tau_m}}) & \text{for} \quad t > t^f \end{cases} \tag{A.4}$$

A.3 Signum Function

The signum or sign function of a real number x is defined as follows:

$$sgn(x) := \begin{cases} -1, & \text{if} \quad x < 0, \\ 0, & \text{if} \quad x = 0, \\ +1, & \text{if} \quad x > 0, \end{cases} \tag{A.5}$$

A.4 Heaviside Step Function

The Heaviside step function, or the unit step function, denoted by $H(t)$ can be described as:

$$H(t) = \begin{cases} 1, & \text{if} \quad t \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{A.6}$$

A.5 Truth Tables for Used Operations

This section covers truth tables of used logical operations in order to demonstrate the output for all possible input conditions. Truth tables have two inputs ($P1$ and $P2$) with 1-bit input. All combinations of the inputs for each operation are shown using logic 1 for TRUE and logic 0 for FALSE. Desired and the inverse of the desired output are denoted Q, $Q^{'}$, respectively.

P1	P2	Q	$Q^{'}$
0	0	0	1
0	1	0	1
1	0	1	0
1	1	1	0

Table A.1: Operation P1.

P1	P2	Q	$Q^{'}$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	0

Table A.2: Operation TRUE.

P1	P2	Q	$Q^{'}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Table A.3: Operation AND.

	P1	P2	Q	$Q^{'}$
	0	0	0	1
s	0	1	1	0
	1	0	1	0
	1	1	1	0

Table A.4: Operation OR.

P1	P2	Q	$Q^{'}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Table A.5: Operation Exclusive Or (XOR).

A.6 LCE Calculation Formula

$$\hat{LCE_E} = \sum_{N_E}(\frac{1}{N_{Sp}^{TST}} \sum_{N_{EXP}} (\frac{1}{N_P^{TST}} \sum_{N_P}(N_{miss}^P))) \tag{A.7}$$

where N_{miss}^P is the number of misclassification per presentation P, N_P^{TST} is the number of presentations in a single testing epoch. N_P^{TST} corresponds to total number of classification during an epoch. N_{Sp}^{EXP} is the number of spike sets performed during each experiment.

Appendix B

Simulators

Contents

B.1 NEURON **265**
B.2 NEST **266**
B.3 GENESIS **266**
B.4 Nengo **267**

B.1 NEURON

The NEURON simulator [1] is developed by John W. Moore, Michael Hines, and Ted Carnevale at Yale and Duke University (Hines & Carnevale, 1997, 2001). It can support creation and evaluation of various morphologically correct neurons and biological mechanisms. Both clock-driven and event-driven mechanisms are implementable in NEURON. Support is handled through the website and a mailing list [2]. The simulator runs under Unix, Linux and MS Windows platforms.

The construction of neuron and network models can be handled by a Graphical User Interface (GUI), or by using its interpreter language called *hoc*. Customized processes such as specifying ion-channels can be implemented with a special language called *NMODL*. It is originally designed for detailed neuronal modelling at the ionic channel level although it is capable of running network models. However, the computation cost for solving all the equations and propagation of signals makes the simulator not optimized for SNNs.

[1] http://neuron.yale.edu/neuron/
[2] https://www.neuron.yale.edu/phpBB2/

B.2 NEST

The NEST simulator (Gewaltig & Diesmann, 2007), which stands for Neuron Simulation Tool, is designed to simulate large Neural Networks with biologically accurate connectivity in the context of their anatomical, morphological and electrophysiological properties. It is designed for heterogeneous networks of primarily point neurons or neurons with a small number of compartments.

Primary goal of the simulator is its scalability and parallelism. It is written in C++ and optimized for large spiking networks of neurons so it is an important option for distributed computations. The software is provided to researchers under an open source license through the NEST website [3]. Support is handled through the website and a mailing list. Unlike the NEURON, NEST is built specifically to run Spiking Neural Networks and is optimized with SNNs.

B.3 GENESIS

GENESIS (General Neural Simulation System) [4] is designed to be an extensible simulation system for the plausible modelling of neural and biological systems (Bower & Beeman, 1998). It is well-documented for users in order to extend its capabilities and flexibilities by adding new user-defined GENESIS classes. Although it suits for realistic models of neurons and biological systems based on known anatomy, it does not support simplified models like Izhikevich model and other Integrate-and-Fire neurons.

GENESIS is primarily designed for multi-compartmental neurons and networks of these neurons. It has a Graphical User Interface which provides object types and script-level commands. The simulator is similar to NEURON and is originally designed for detailed neuronal modelling at the ionic channel level although it is capable to run network models. However, the computation cost for solving all the equations and propagation of signals make these models difficult to use in large network applications.

[3] http://www.nest-initiative.org/
[4] http://www.genesis-sim.org/GENESIS/

B.4 Nengo

Nengo [5] is an open-source software which offers a general method for implementing and simulating high-level cognitive theories using biologically plausible spiking neurons. It also provides a Graphical User Interface to construct networks. The simulator has a discrete clock-based time paradigm with multiple neuron models. It is designed for the large-scale networks of simple neural models (such as Leaky-Integrate-and-Fire) based on control theory oriented approach called the Neural Engineering Framework (NEF).

[5] http://nengo.ca

Appendix C

Simulation Framework Documentation

Contents

C.1 Introduction . . . 270
C.2 Motivation : The Memory Problem . . . 270
C.3 Software Specification . . . 271
C.4 Description of Simulation Sections . . . 271
C.5 Simulation Folder Mechanism . . . 272
C.5.1 Session Mechanism . . . 272
C.5.2 Initialization Mechanism . . . 272
C.5.3 Spike Set Mechanism . . . 273
C.6 Modes . . . 273
C.6.1 Run Mode . . . 273
C.6.2 Analyse Mode . . . 274
C.7 Stored Dynamics . . . 274
C.8 Learning Algorithms . . . 275
C.9 Parameter Settings . . . 275
C.9.1 Neuron Types and Dynamics . . . 276
C.9.2 Number of Neurons . . . 276
C.9.3 The Distribution of Input Types . . . 276
C.9.4 Learning Rate . . . 277
C.9.5 Min/Max Learning Weights . . . 278
C.9.6 Min/Max Initial Weights . . . 278
C.9.7 Scaling Status . . . 278
C.9.8 Scaling Factor . . . 278
C.9.9 Initialization Methods . . . 278
C.9.10 Timings . . . 279
C.10 Generation of Spike Sets . . . 280
C.10.1 File System of Spike Sets . . . 282
C.11 File Content of Session for Training and Testing . . . 283
C.12 The Generations of Dynamic Initializations (File System) . . . 283
C.13 Used File Formats . . . 284
C.14 Further Work . . . 284
C.15 Discussions . . . 285

C.1 Introduction

The management of training and/or testing in the Spiking Neural Network is slightly more complicated in a traditional ANN. As the size of network or the duration of simulation grows, the performance of experiments for training/testing and the management of the implementation become much more complex in SNN simulations. However, as far as we know, there is no public tool available for the Brian package or other SNN simulators to ease this issue. Thus, we have produced our own additional development for the SNN simulation environment. This appendix describes the structure and operation of the simulation environment.

This tool is an additional package for SNN simulation tools. The purpose of the design is to ease the management of simulation load/reload mechanisms, store/restore training trajectories or results, separation of training and testing phases and further analysis mode with additional flexibilities. After each training procedure for the network, the structures save themselves to permanent store in a computer independent format to re-activate for further learning cycles, to test performance with various testing scenarios or analyse the trajectories and results during training and testing throughout further simulations.

In this part, a lot of implementation details related with SNN simulation are not mentioned such as synaptic connection managements, event managements in the network, the types of synapses, building the spiking network, etc. All of those details are implemented based on Brian documentation [1]. Also, Brian syntax is quite similar to other simulators (Goodman & Brette, 2008; Stimberg et al., 2014). However, in this section we detail the organization of various simulations, experiments, sessions, epochs and presentations.

C.2 Motivation : The Memory Problem

In order to monitor the behaviour of the network during the training, learned objects such as synaptic strengths and/or synaptic delays need to be accessed not only during the simulation and after the simulation. However, once the number of epochs and presentations are increased, the performance of simulation is affected severely because of the lack of sufficient program memory (temporary) in the machine. This is also problematic in order to monitor trajectories from larger simulations. In order to handle this issue, connected layers of nodes and all

[1] http://www.briansimulator.org

relevant parameters including the connection weights and/or delays matrices are dumped into a file between epochs, experiments and simulations. So, by this structure, the knowledge from each training steps are preserved in the permanent memory (disk) rather than the temporary memory.

C.3 Software Specification

The software is implemented in Python 2.7 as the main programming language thus making it flexible to use. It is integrated to Brian2 simulation package as an SNN package. However, it can be used with other SNN or NN simulators based on Python. This software has been tested on Windows 7. However, it can run on other operating systems such as Linux smoothly because we do not use any platform-specific system libraries or packages during our implementation. Eventually, it will be available for public use.

C.4 Description of Simulation Sections

Overall structure is demonstrated in Equation C.1. Experiments consist of sessions, and sessions consist of epochs, and epochs consist of presentation periods as shown.

- **Presentation** : This is the single simulation phase of a single stimulus (p_1). The duration of each presentation is fixed represented as T_{pe}. The number of presentation in an epoch is N_P.
- **Epoch** : An epoch e_1 is the group of presentation cycles. The number of epoch in a session is N_E.
- **Session** : A session s_1 is the group of epoch periods. The number of session in an experiment is N_S.
- **Experiment** : An experiment exp_1 is the group of sessions for specified aims. For instance, the aim for the experiment can be executing training and testing parts throughout all ranges in the determined spike train set. Users can execute as many experiments as needed.

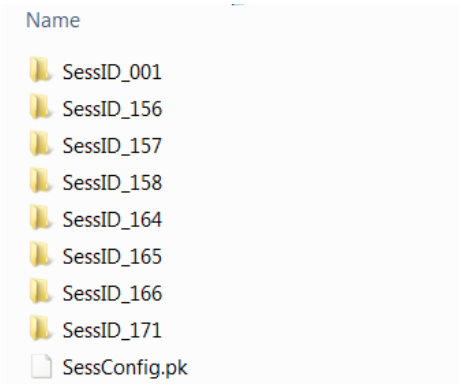
$$\underbrace{\overbrace{\overbrace{p_1,p_2,...,p_{N_P}}^{e_1},\overbrace{p_1,p_2,...,p_{N_P}}^{e_2},\overbrace{p_1,p_2,...,p_{N_P}}^{e_{N_E}},}^{s_1},...,\underbrace{\overbrace{p_1,p_2,...,p_P}^{e_1},\overbrace{p_1,p_2,...,p_P}^{e_2},\overbrace{p_1,p_2,...,p_{N_P}}^{e_{N_E}},}_{s_{N_S}}}_{exp_1} \tag{C.1}$$

C.5 Simulation Folder Mechanism

As mentioned, all parameters, trajectories and values are stored in files to use them further during testing or analysing the results. In order to handle all of our issues during our experiments, we have decided to create three main directories: the folder of sessions (*FolderSess*), initializations (*FolderInit*) and spike sets (*FolderSets*). The details of those folders/structures are described below. However, another folder mechanism can be added or any folder or part of the current structure can be removed or edited without knowing the details of implementation.

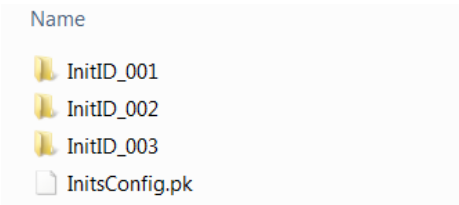
C.5.1 Session Mechanism

Example screenshot of folder structure for sessions can be seen on the right. Folder name of sessions is constructed from the prefix *"SessID_"* and its unique session id. All of details, settings and parameters for the indicated session are stored in *"SessConfig.pk"* file. Whenever a new session is created or any existed session folder is removed, the configuration file is updated. All files and trajectories generated during the session are stored under the indicated session directory.



C.5.2 Initialization Mechanism

Example screenshot of folder structure for initializations can be seen on the right. Folder name of initializations is constructed from the prefix *"InitID_"* and its unique initialization id. All of details, settings and parameters for the indicated initialization are stored in *"InitsConfig.pk"* file. Whenever a new initialization is created or any existed initialization folder is removed, the configuration file is updated. The content of file for weights and delays can be seen in Table C.1. All initialization sets generated before experiments are stored under the indicated initialization directory.

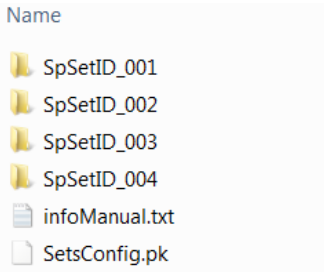


RangeInd	DataWeights
RangeInd	DataDelays

Table C.1: Format for initializations : Weights (in the top row) and delays (in the bottom row).

C.5.3 Spike Set Mechanism

Example screenshot of folder structure for spike sets can be seen on the right. Folder name of spike sets is constructed from the prefix *"SpSetID_"* and its unique spike set id. All of details, settings and parameters for the indicated spike sets are stored in *"SetsConfig.pk"* file. Whenever a new spike set is created or any existed spike set folder is removed, the configuration file is updated. All spike sets generated before experiments are stored under the indicated spike set directory.



C.6 Modes

The proposed structure has two modes: *Run Mode* in order to execute different training and testing sessions, and *Analyse Mode* in order to analyse and display learning trajectories/results from the run mode. The pseudo-codes of training and testing procedures are described in Algorithm 4 and in Algorithm 6, respectively. This separation gives opportunity for design modularity. During run mode, only selected parameters/trajectories are stored onto the disk. Once all needed sessions are executed, related graphs/figures can be drawn in the analyse mode. Analyse mode functions can only read previously stored files and parameters; however, none of them cannot be modified during analysis unlike run mode.

C.6.1 Run Mode

Simulating the network to obtain the results in *Run Mode* has two phases: training and testing. For each group of spike patterns, trainings and testings are performed separately.

- **Training Phase**: This procedure is designed to perform the iterative learning process based on the selected learning strategy. This learning process adjusts the synaptic values (weights and/or delays) of the spiking model in such way that the output generates a different spike pattern for each input pattern.

- **Testing Phase**: This procedure is to test the performance of each recorded training steps with all possible input combinations. For instance, during the experiment of AND logical operations with two bits (input), corresponding spike patterns for all 4 possibilities in the format of {(Input1, Input2) → Output} are {{(0,0) → 0}, {(0,1) → 0}, {(1,0) → 0}, {(1,1) → 1}} are processed and recorded for the single testing epoch result by comparing the resulting outputs against the desired outputs. There are no adjustments related with any configurable parameters (weights/delays). At the end of each testing epoch, the state of all neurons are reset. The stop criterion adopted in this algorithm is the maximum epoch number of testing E_{TST} same as the dependent training epoch number E_{TRN} (see Table C.2).

C.6.2 Analyse Mode

Although the main purpose in this mode is to analyse the results of testing sessions, files of training sessions also can be analysed under this mode. Results, training trajectories, testing results can be visualized using various types of figures. In this thesis, all demonstrated results are prepared with the help of this mode.

- **Training**: The generated trajectories during training sessions can be analysed in this mode.

- **Testing**: Primary aim is to use this mode with testing sessions in order to demonstrate overall performance of the learning. Not only training details also entire testing results are already stored under the files of testing sessions.

C.7 Stored Dynamics

The following parameters/trajectories during training and/or testing can be stored/restored during the simulation by the proposed mechanism.

- **Firing times**: All firing activities of all neurons can be recorded during the training and testing sessions. However, if it is not necessary to monitor them, recording and storing can be turned off over the configuration file as well. For instance, in order to increase the speed of simulation, we disable the monitoring of spiking times during training; however,

enable it during testing because we need them in order to calculate error/performance from testing outputs.

- **Weights**: In addition to initial weights, all synaptic weights are stored during every steps of training sessions. Testing does not store them because they are already reloaded from training based on the session id.

- **Delays**: If the learning algorithm adjusts synaptic delays, they are also stored during every steps of training sessions. Testing does not store them, similar to weights.

- **Network parameters**: Number of neurons for all layers, number of synaptic connections, the types of synaptic connections are also stored.

- **Simulation parameters**: Some other stored parameters can be listed as simulation clock resolution, type of learning algorithm, parameters for learning algorithms, weight scaling status, weight scaling parameters.

C.8 Learning Algorithms

The package now has four training algorithms that apply weight and/or delays by capturing the detailed firing effects of individual neurons. One algorithm modifies weights and delays. The other algorithms apply learning rules in only weights. We tested the framework with: Spike-timing Dependent Plasticity (STDP) (detailed in section 3.4), Remote Supervised Method (ReSuMe) (detailed in section 8.3), Delayed Remote Supervised Method (DelReSuMe) (detailed in section 8.3), and Reward-modulated Spike-timing Dependent Plasticity (R-STDP) (detailed in chapter 9).

C.9 Parameter Settings

Each parameter in the simulation is adjustable. Those parameters can be listed as neuron types and dynamics, number of neurons, the distribution of input types, learning rate, min/max learning weights, min/max initial weights, scaling status, scaling factor, initialization methods, and timings.

C.9.1 Neuron Types and Dynamics

Neuron types for input and outputs can be selectable through implemented models as dummy Leaky-Integrate-and-Fire, Leaky-Integrate-and-Fire, Integrate-and-Fire, Izhikevich model and Spike Response Model. Details of those neurons can be found in Chapter 1. For each neuron or for each group of neurons, all neuron dynamics such as threshold voltage, resting potential, reset voltage, decaying time and refractory period can be adjustable from the configuration file. Also, a new type of neuron model can be inserted using Brian syntax.

C.9.2 Number of Neurons

The number of processing nodes per layer are important decisions. Although there are some general rules, there is not a single best answer to the layout of network for any particular problem. The number of neurons is generally chosen empirically to achieve a balance between accurate results and rapid learning. For that reason, the following parameters are established in an easily configurable way to be able to choose different number of spiking neurons.

- Number of banks in input layer
- Number of neurons in each bank
- Number of output neurons

Input layer consists of various banks. Each bank is the group of neurons in order to represent different inputs. The number of neurons in the input layer is the summation of numbers of all bank neurons. Although majority of our experiments are based on single bank ($P1$) or two banks ($P1$ and $P2$), this number can be modified. Single bank is used for the mapping experiments. Two banks are used for logical operations with 2 logic inputs.

C.9.3 The Distribution of Input Types

This parameter determines how the input logic set is prepared. There are 4 different types defined below. Always Logic 0 and 1 cases are useful to test the learning algorithm for single input-output pattern mappings, especially for the initial benchmarks. Sequential scenarios are used during testing in order to cover entire input possibilities.

- Always Logic 0: Input 1 and Input 2 are always Logic 0

- Always Logic 1: Input 1 and Input 2 are always Logic 1

- Sequential: All combination of inputs have equal probabilities. For two inputs, possible input preparation with 4 presentations (typical testing) can be shown as {(0,0), (0,1), (1,0), (1,1)}.

- Random-Independent : The distribution of both logic values for each input can be adjusted with the logic separation parameter during each epoch. In the random case, this parameters is setted into 0.5. For the case of 10 presentations with the 0.5 of separation value, [0 0 1 1 1 0 0 1 0 1] and [1 1 0 0 0 1 1 0 1 0] can be example input logic streams for Input 1 and for Input 2, respectively.

- Random-Dependent: The distribution of input pairs are controlled here rather than controlling logic values inside each input. For 2 inputs, there are already 4 possible pairs. Those 4 possibilities are extended based on the given presentation number. For the case of 10 presentations (typical training), [0 0 1 0 1 1 1 1 1 0] and [0 1 0 0 0 1 0 1 1 1] can be example input logic streams for Input 1 and for Input 2, respectively.

If we compare Random-Independent and Random-Dependent through given examples, the difference can be easily seen. In the Random-Independent case, although each input pattern has equal True-False distributions, once they have been combined some of pairs have much more ((0,1) and (1,0)) than others ((0,0) and (1,1)) in the example scenario. However, in the Random-Dependent all scenarios are chosen to equal distributions as in the above example. Random-Dependent case is ideal for training. Although this type has random distribution of input possibilities, the strength of each input pairs is similar inside each epoch unlike the Random-Independent.

C.9.4 Learning Rate

Some learning algorithms can be heavily dependent on the learning factor η. It is chosen empirically to ensure convergence to optimum learning. Hence, the learning factor of each experiment can be adjustable through the configuration file. Moreover, adaptive learning rates are tested like higher learning rate for early epochs, smaller rates for later.

C.9.5 Min/Max Learning Weights

Synaptic weights are restricted between w_{min} and w_{max} during training. These parameters can be adjusted before the simulation. If any real-valued synaptic weight is out of the given boundaries during the weight update, they will be limited by the lower and upper bounds by clipping. Otherwise, some synaptic weights would increase or decrease excessively without any impact on the learning.

C.9.6 Min/Max Initial Weights

Those parameters ($w_{min}^{init}, w_{max}^{init}$) are used once weights in a new initialization set generated. The selected range for the initial weights can be adjustable through those two parameters. However, the initialization weight range must be in the range of learning weight ranges with $w_{min} <= w_{min}^{init} <= w_{max}^{init} <= w_{max}$. Also, inhibitory connections can be arranged by assigning negative values to weights.

C.9.7 Scaling Status

Whether neglecting the scaling effect for synaptic weights is applied at the end of the epoch or not can be adjustable with this parameter.

C.9.8 Scaling Factor

Weight scaling factor of each experiment can be adjustable through configuration file. Based on the scaling type, weights are readjusted inside the range (w_{min}, w_{max}).

C.9.9 Initialization Methods

- **The type of weight initialization**: This parameter defines how the weights for synaptic connections to output neurons from input neurons should be initialized based on the constrained range ($w_{min}^{init}, w_{max}^{init}$). This procedure is separated from training/testing sessions. By this separation, the effect of the initialization set can be investigated under different circumstances (such as different learning algorithms, different network parameters). At the beginning of each training experiment, the selected set is reloaded into the network. For this purpose, we prepare various sets for each of following techniques:

- Specific initialization: Initialize all weights into any given value from the interval (w_{min}^{init}, w_{max}^{init}).

- Uniform Initialization: The element of training synapses are assigned values from uniform random distribution subject to bounds of w_{min}^{init} and w_{max}^{init}.

- **Delays**:

- Specific initialization: Initialize all delays into pre-specified values.

- Sequential initialization: Based on the synapse number between the same input and output neuron, each synapse gets a different sequential delay like [1ms, 2ms, ...].

- **Spike Generation**:

- Poisson (see subsection 6.3.2)

- Uniform

C.9.10 Timings

The following durations and timings are adjustable:

- Duration of single presentation (T_p and T_{pe})
- Duration of silent period after each presentation (T_e)
- Number of presentations for training (N_P^{TRN}) and testing (N_P^{TST})
- Number of epochs for training (N_E^{TRN}) and testing (N_E^{TST})

Although number of presentations and epoch both for training and testing can be adjustable separately, epoch number for both cases should be the same like $N_E^{TRN} = N_E^{TST}$ because of the dependency of testing to training, or at least $N_E^{TRN} \geq N_E^{TST}$. Based on above parameters, the total training or testing time in a session can be calculated as in Equation C.2.

$$T_{sess} = \overbrace{(T_p + T_{silent})}^{T_{pe}} * N_P * N_E * N_S \tag{C.2}$$

where N_P is the number of presentation in an epoch, N_E is the number of epoch in a session, S is the number of session in an experiment. Time for session, presentation, and extended presentation are T_{sess}, T_p, and T_{pe},respectively. Although typical timings for training and testing are summarized in Table C.2, each of those durations as above can be adjusted in the

Name	Value (Training)	Value (Testing)	Unit
T_p	100	100	ms
T_{silent}	20	20	ms
T_{pe}	120	120	ms
N_P	10	4	-
N_E	1000	1000	-
N_S	5	5	-
T_{total}	100	40	min

Table C.2: Typical durations of simulation parameters for training and testing.

configuration file. Based on chosen parameters, the total duration of an experiment can be calculated with:

$$\begin{aligned} T_{exp} &= T_{pe} * (N_P^{TRN} + N_P^{TST}) * (N_E^{TRN} + N_E^{TST}) * N_{Sp} \\ &= T_{pe} * (N_P^{TRN} + N_P^{TST}) * 2 * N_E^{TRN} * N_{Sp} \end{aligned} \tag{C.3}$$

where N_P^{TRN}, N_P^{TST}, and N_{Sp} are number of training, testing and spike patterns, respectively. Because of $N_P^{TRN} = N_P^{TST}$, it is reformulated in the next line. A typical experiment including training and its dependent testing phase is 140 minutes $\approx$ 2.3 hours as summarized in Table C.2.

C.10 Generation of Spike Sets

The training of SNNs is done using spike sets stored in the folder with the indicated spike set id. Therefore, before the training procedure, spike set generation is already completed. During the simulation, spike sets can be chosen from previously generated sets.

The mechanism of spike set generation is separated from the main loop of the simulation procedure. However, based on the desired parameters in the simulation, a suitable spike set can be loaded into the current simulation in order to feed the network. Although various types of spike sets are provided with the framework, new sets with desired parameters can also be generated. Following parameters for spike generation should be adjusted before preparing a new set in the file:

- **Firing rates for input patterns (r_{in})**: The average number of spikes during a presentation time (T_p) is divided by the duration of the presentation (T_p) to get the firing rate. If we assume that $r_{in} = 0.1sp/ms$, it means each input presentation pattern roughly have 10 spikes as:

$$N_{in}^{sp} = r_{in} * T_p = 0.1sp/ms * 100ms = 10sp \tag{C.4}$$

- **Firing rates for output patterns (r_{out})**: Chosen firing rates for outputs should be enough to discriminate among the input firing rates. If we assume that $r_{out} = 0.06spike/ms$, it means each input presentation pattern has roughly 6 spikes as:

$$N_{out}^{sp} = r_{out} * T_p = 0.06sp/ms * 100ms = 6sp \tag{C.5}$$

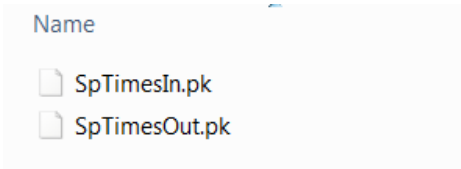
- **Spike Set Range**: This determines the number of time series set that are generated in total for each neuron in the network. For instance if this is 3, it generates 3 different pattern classes belong to the same spike set. For the experiment to test the same algorithm with various spike sets, this feature helps to organize sets in this manner.

- **Minimum Inter-spike Interval (ISI) type**: This parameter guarantees that the shortest time distance between two spikes cannot be less than min-ISI (similar to the absolute refractory period of a neuron). During, the poisson spike generation, if any two sequential spikes have time difference less than the min-ISI, it is shifted based on selected types :
  - Fixed Offset
  - Gaussian Offset

C.10.1 File System of Spike Sets

The folder system for the Spike Sets can be found in subsection C.5.3. Here we focus on the files and the structure of their contents. An example screenshot of file structure for a spike set can be seen on the right. Firing times for input neurons and output neurons are stored in separate files. Each spike set has at least two files as *"SpTimesIn.pk"* for input neurons and *"SpTimesOut.pk"* for output neurons.

A single-spike set contains a group of spike patterns and each of them represented with spike range index (RangeInd). The content of each file is illustrated in Table C.3. The first format with {(RangeInd, NeuInd, LogicInd) : $[t^{f1}, t^{f1}, ..., t^{fN}]$ } is used inside the file system. Based on the spike range index, suitable spike patterns can be parsed and loaded into program memory for each experiment simulation.



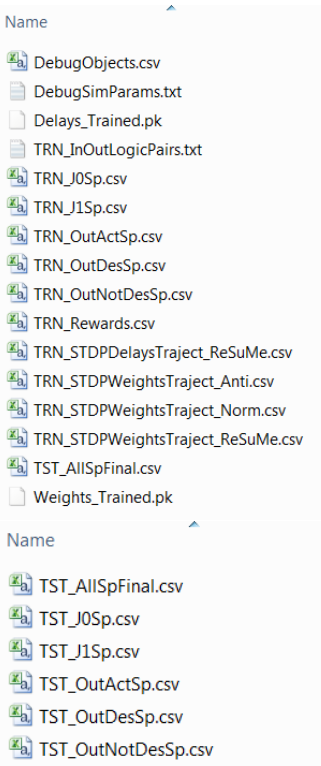
The other three formats are used in program memory when needed. For instance the second format {(NeuInd, LogicInd, PresentationInd) : $[t^{f1}, t^{f1}, ..., t^{fN}]$ } is used during each experiment because spike set cannot be changed during an experiment so spike range ind (RangeInd) is already fixed. However, in each presentation although the spike pattern is not changed, they need shifted and adjusted based on the simulation time and silent period between presentations.

RangeInd	NeuInd	LogicInd	DataSpikeTimes
NeuInd	LogicInd	PresentationInd	DataSpikeTimes
NeuInd	LogicInd	DataSpikeTimes	
NeuInd	PresentationInd	DataSpikeTimes	

Table C.3: Format for firing times to write into file and program memory.

C.11 File Content of Session for Training and Testing

An example screenshot of file structure for each session mode can be found on the right. Based on the running mode (training or testing), suitable files are prepared during the simulation. Training files (above) are generated with the prefix *"TRN_"* and testing files (below) are generated with the prefix *"TST_"*. Firing times of all neurons in each bank in input (Bank names : J0, J1) and in the output are recorded into the indicated files. Also, the trajectories of all types of weights are recorded in suitable files. Synaptic delay updates are also recorded if the selected learning mechanism is configuring those delays. Rewards are also stored for the reward based learnings. Although those trajectories are recorded (weights, delays and rewards), new mechanisms can easily be integrated into the system in order to store/restore it.



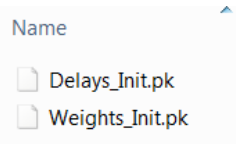
In the testing mechanism, only firing activities of every neuron are recorded. Weights, delays or other learning parameters are not stored because those parameters are already restored from dependent training mechanism. Therefore, if any of those trajectories or values are needed, they can be access with the dependent training session id. The content of session configuration file can be seen in Table C.4.

SessIdMain	SessIDFrom	ExpInd	InitId
SpSetId	InitRangeIndAct	SpSetRangeIndAct	SessTypeInd
OpInd	PrNum	EpNum	CstName

Table C.4: The contents of session configuration file.

C.12 The Generations of Dynamic Initializations (File System)

Folder system for the Inits can be found above. Here we focus on the files and the structure of their contents. Example screenshot of file structure for an initialization can be seen on the right. Initial weights and delays are stored in separate files. Each initialization has at least two files *"Delays_Init.pk"* for delays and *"Weights_Init.pk"* for weights.



C.13 Used File Formats

In order to serialize all prepared data for the simulation and generated data during the simulation to the disk, we use two file formats: Comma Separated Values (CSV) and the optimized version of Python's standard serialization tool (cPickle). The reason to choose pickle format, it is quite straightforward to store/restore objects, classes, data if you are not parsing or modifying it during the store/restore. However, the content of pickle files is not human readable like CSV file. Therefore, during the simulation if we need to see the data, it is preferred to use CSV format for this particular data (for instance firing activities of neurons). If they are only needed once for the simulation such as initialization files, restoring parameters, they are stored in the pickle format. However, all of format preferences easily changeable to other format, or entirely new format such as Binary Structured Object Notation (BSON), JavaScript Object Notation (JSON) can be used in the proposed tool.

C.14 Further Work

We have not prepared any software comparison tests because all of our implementation is in the Python language; and not tested in any other language. However, entire simulation including neural models can be converted into C++ language. This conversion may help to increase considerably the speed of long experiments. On the other hand, it can be reimplemented in a different language such as MATLAB, Java or in an embedded platform. This means that experiments do not has to be simulated in Brian simulator. Although this gives a chance for performance comparison, reimplementing the models and scenarios in other languages can be a time consuming task. Here, the SWIG [2] wrapper tool can be an important option to convert it into a target language, and then making comparisons. The benefit of SWIG is the auto-conversion to target language instead of typing entire details in the target language. This quick conversion can give a rough idea about the performance.

Furthermore, the organization of simulation is handled throughout the configuration file. This mechanism can be transferred into user-friendly visualization by Graphical User Interface. This feature can also support interactive engagements with experiments/results. In addition, this can be a dedicated Brian extension as a package of Python.

[2] http://www.swig.org/

C.15 Discussions

One of main advantage of this framework is modularity of prepared parts for the network and results generated from the network. Needed parts for the network before the simulation entire initialization sets (weights, delays) and spike pattern sets can be prepared externally (outside the main simulation of training and testing). Also, generated results such as the training trajectories of weights and delays are stored with their simulation parameters. This structure helps to compare different scenarios by selecting all/some of sets/specific sets during an experiment. Also, it prepares a structure for the analysis to access results/trajectories generated during the main training/testing loop.

The framework also separates training and testing inside the run mode and the analyse mode. This can enable more flexible testing and trainings scenarios. Various options can be bridged by playing with parameters in this modular design.

Another advantage of the framework is to ease the memory problem for larger simulations. Between sessions and experiments all parameters/trajectories are already saved in the disk. This allows to increase the performance of program memory.

Appendix D

Further Results

Contents

D.1 Further Results for Mapping Experiments 287
D.1.1 ReSuMe Experiments 287
D.1.2 DelReSuMe Experiments 287
D.2 Maze Platform on Python 287
D.3 Maze Generation 290
D.4 Maze Solving Algorithms 291
D.4.1 Left Wall Follower (LSR Rule) 292
D.4.2 Right Wall Follower (RSL Rule) 293
D.5 Example Screenshot from Workspace 294
D.6 Prerequisites 294
D.7 Software Contributions 295

D.1 Further Results for Mapping Experiments

D.1.1 ReSuMe Experiments

D.1.2 DelReSuMe Experiments

D.2 Maze Platform on Python

Python is a well known dynamic object-oriented scripting language. Reinforcement Learning is applied to find the shortest path using Python. The goal is to create a flexible Reinforcement Learning platform for further algorithm development and research based on Python which is free, open source, and highly portable.

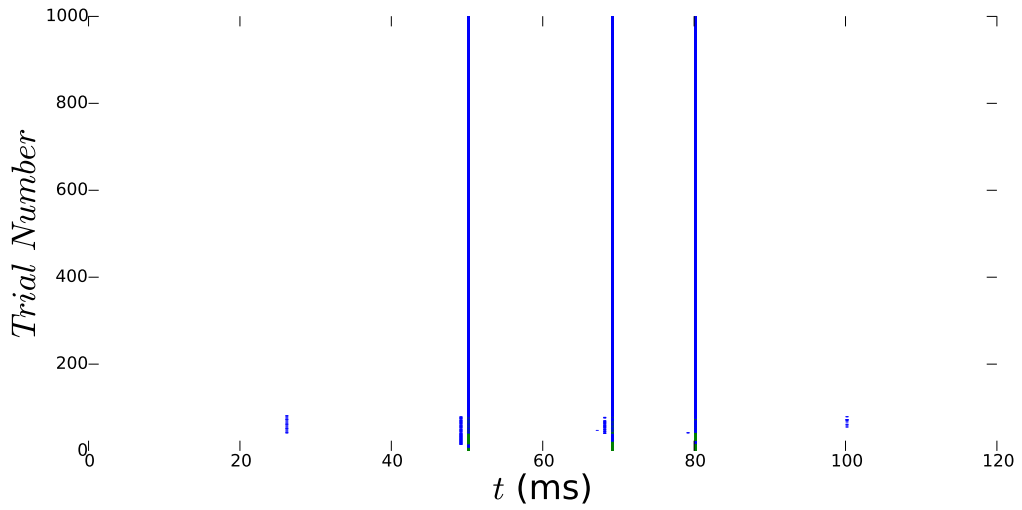


Figure D.1: Reconstruction of the transformation from input patterns to output spike timings through randomly selected experiment (Experiment 1 - noiseless).

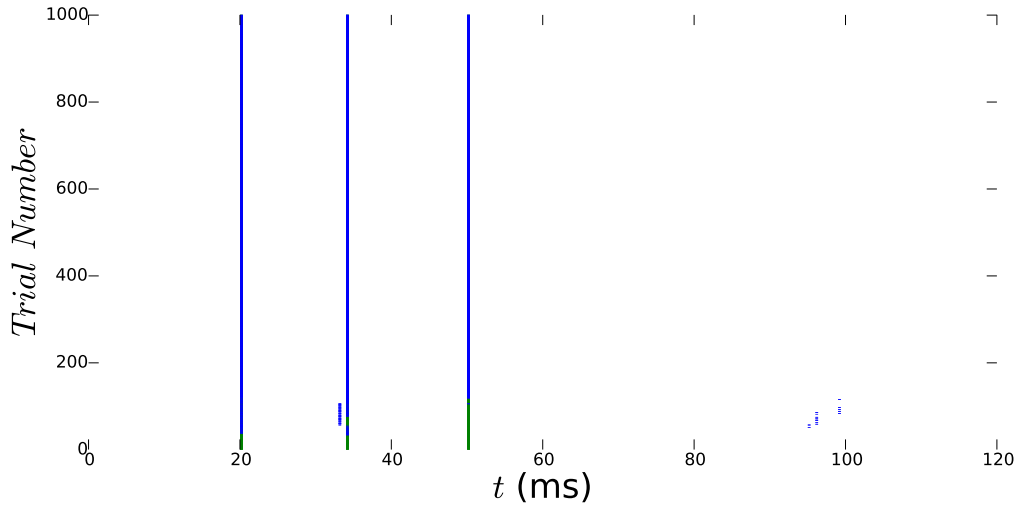


Figure D.2: Reconstruction of the transformation from input patterns to output spike timings through randomly selected experiment (Experiment 2 - noiseless).

A simple menu interface is created to add user-friendly features to the simulation. Figure D.8 shows how the menu options can be selected using keyboard control feature. Based on chosen user option from menu screen, the suitable method is called over maze environment as illustrated in Figure D.9.

Although the size of the maze environment can be adjusted from the code, a 10x10 maze environment is demonstrated. It is tested using random maze generation.

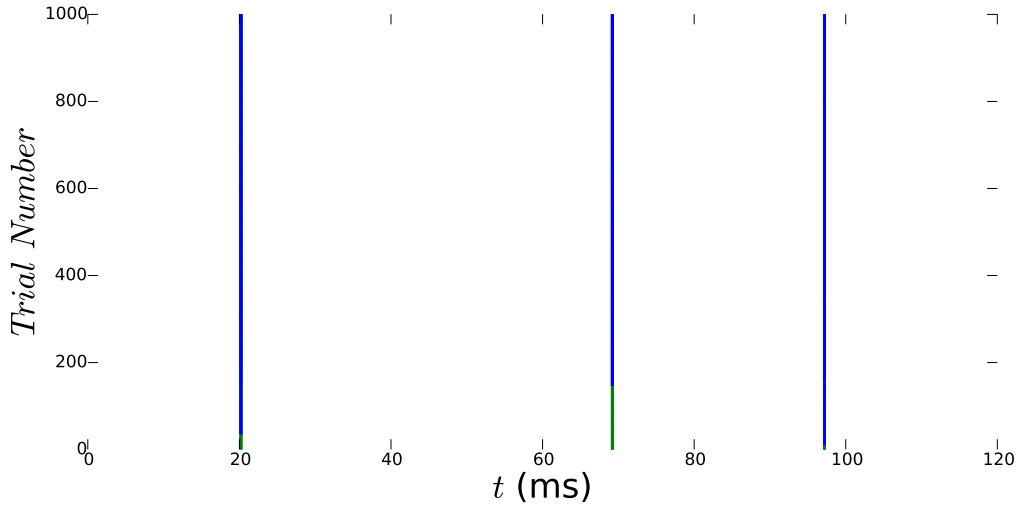


Figure D.3: Reconstruction of the transformation from input patterns to output spike timings through randomly selected experiment (Experiment 3 - noiseless).

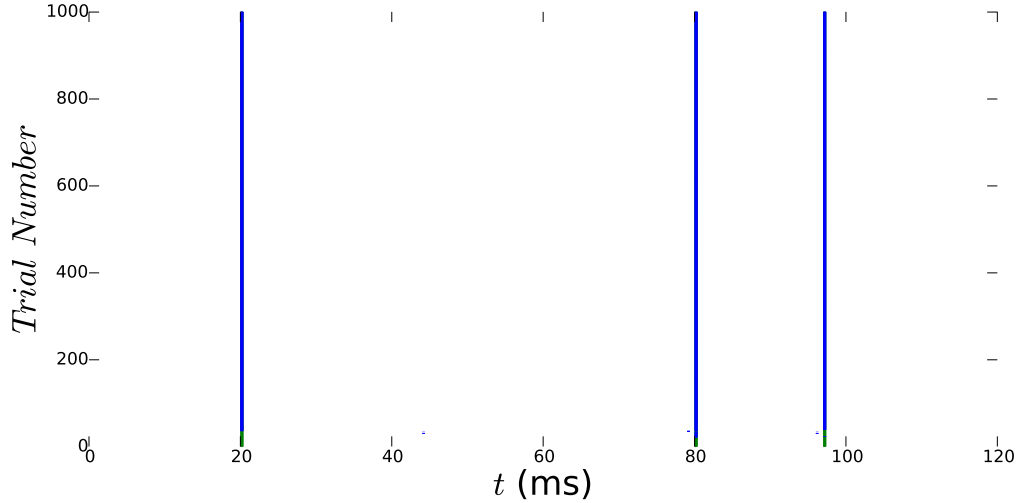


Figure D.4: Reconstruction of the transformation from input patterns to output spike timings through randomly selected experiment (Experiment 4 - noiseless).

In the simulation, the circle (agent) have three different colours which mean:

- Red Circle: More than one times visited rooms
- Green Circle: Only one time visited rooms
- Blue Circle: Current visited room

At each time step, an agent can move North, South, East, or West (not diagonally) into an adjacent square location. Squares may be blocked by walls as denoted by the black lines on D.9. The maze is unknown at the beginning.

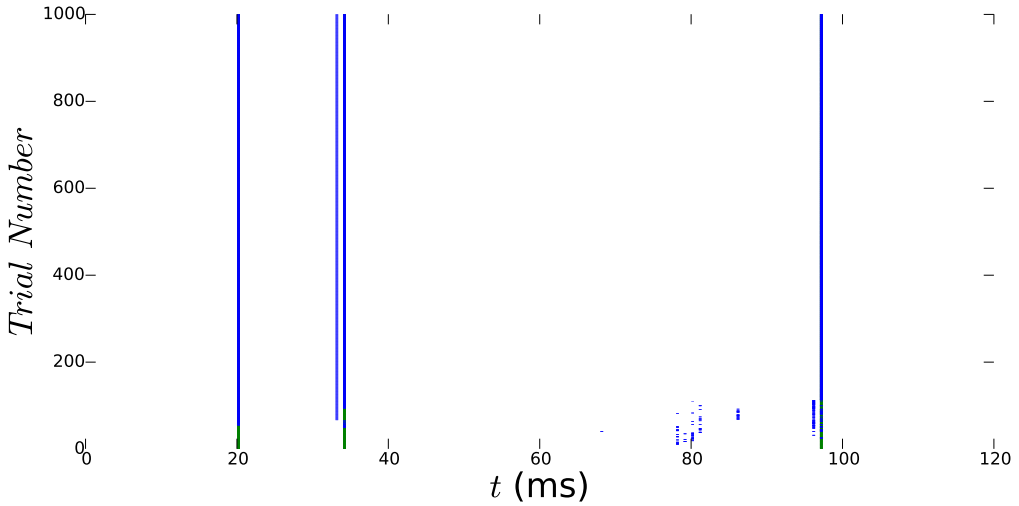


Figure D.5: Reconstruction of the transformation from input patterns to output spike timings through randomly selected experiment (Experiment 5 - noiseless).

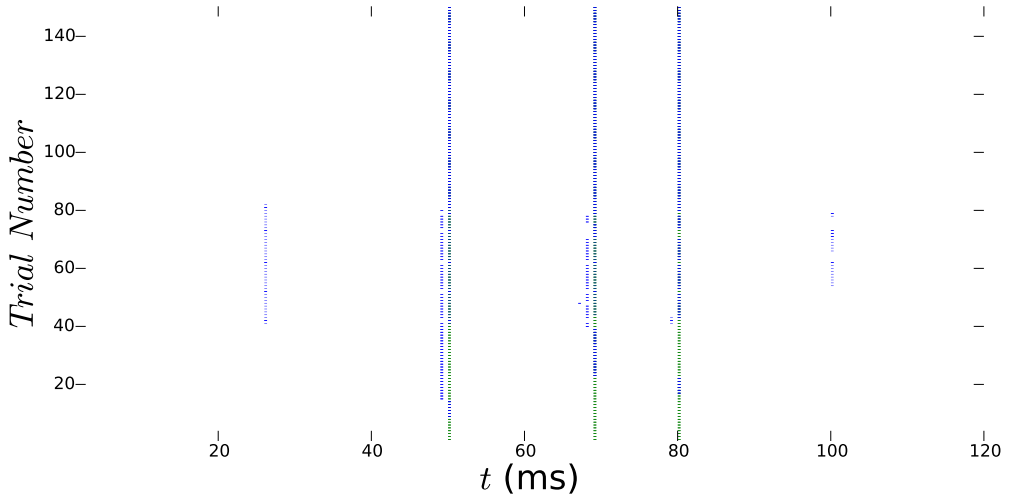


Figure D.6: Reconstruction of the transformation from input patterns to output spike timings through randomly selected experiment 1.

D.3 Maze Generation

Here, we describe details of recursive backtracker with an example demonstration. It is fast and straightforward to implement. As a starting point, it is selected the left-up corner of the graph also as a ending point, the right-down corner of the graph is selected. The width and height of the grid can be adjustable over the program. It can be generated as a larger maze environment as we want in this implementation. For instance, Figure D.10 demonstrates a 40 x 40 maze environment.

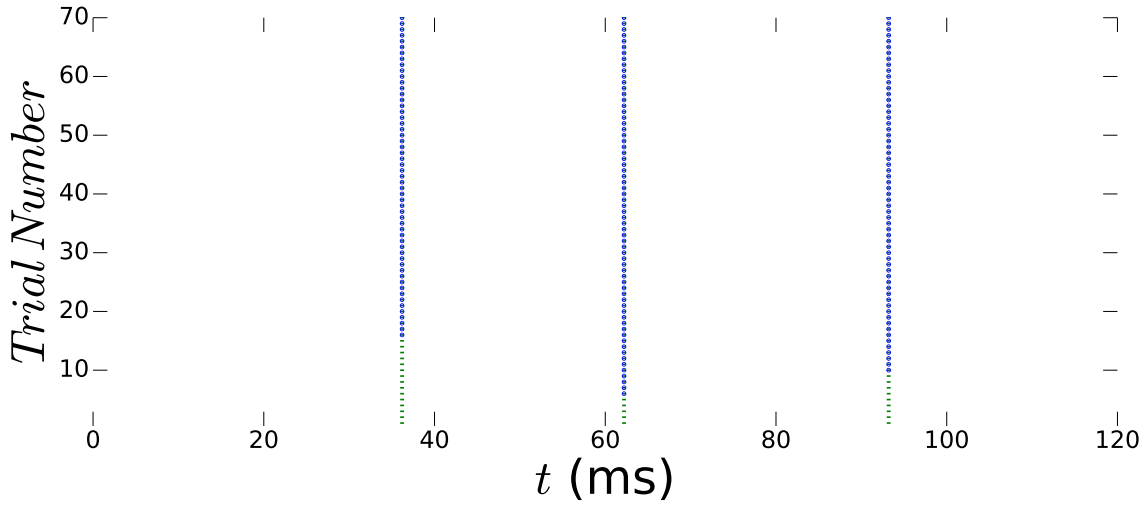


Figure D.7: Reconstruction of the transformation from input patterns to output spike timings (low noise). Zoomed version of Figure 8.9. The current network is trained to map a spatio-temporally encoded input spike trains into another spatio-temporally encoded output pattern. t_{out}^{act} with blue colour converges to t_{out}^{des} with green colour.

The Backtrack Recursion algorithm is implemented using the steps:

- Initialize all cells as not visited

  Initialize a grid environment full of walls

  Initialize specified starting point as left-up corner in the field

- Randomly choose a wall at current cell and remove this wall but only if the adjacent cell is not visited before. The adjacent cell becomes a new current cell.

- If it reaches a dead end, backtrack until get next to an unvisited cell and carrying on that direction.

- The algorithm ends when there is not any cell which hasn't been visited yet.

D.4 Maze Solving Algorithms

There are a number of maze solving algorithms available to be implemented which are automated methods with different characteristics such as Dead Reckoning, Dead End Learning, Tremaux's method, Flood Fill algorithm and Wall follower algorithms (Dudeney, 1970; Bellman, 1958; Law, 2013; Mishra & Bande, 2008). Dead Reckoning algorithm is simply that the agent goes straight until there is a fork or dead end. If there is a fork, the agent select one the available directions randomly. On the other hand, it turns around if there is a dead end.

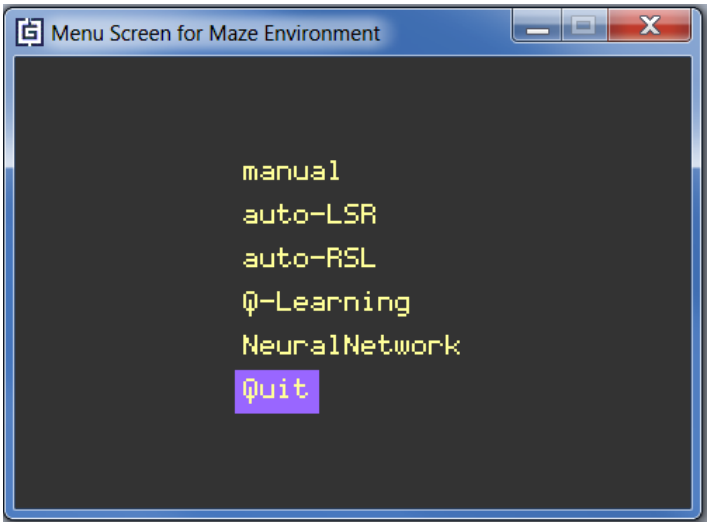


Figure D.8: Python maze simulation menu

Dead End Learning algorithm is similar to Dead Reckoning method but the agent inserts a virtual wall at the opening if it encounters any dead end.

Tremaux's algorithm is invented by Charles Pierre Tremaux and it is based on marking the entrances and exits where the galleries fork. A "new" path or node is one that is not entered before on the route; an "old" path or node is one that is already entered (Dudeney, 1970). The Flood Fill algorithm is derived from the "Bellman Ford Algorithm" (Bellman, 1958). The algorithm works by assigning value for all cells in the maze, where these values indicate the steps from any cell to the destination cell. Implementing this algorithm requires to have two arrays to hold the walls map values and to store the distance values (Law, 2013). However, one of the simplest maze solving algorithm, "wall follower", is used and presented here.

Two types of wall follower algorithm as left-hand rule and right-hand rule work the same way except the turning priority which can be either to the left or to the right. Although those are quite similar, we investigate both of them and we demonstrate Replacement Rule for both.

D.4.1 Left Wall Follower (LSR Rule)

The left-hand rule, LSR rule, states Left direction has highest priority compared to Straight and Right directions while there are options for turns. Likewise, Straight has higher priority than Right turn. The order of precedence can be summarized as:

$$\text{Left} \Rightarrow \text{Straight} \Rightarrow \text{Right}$$

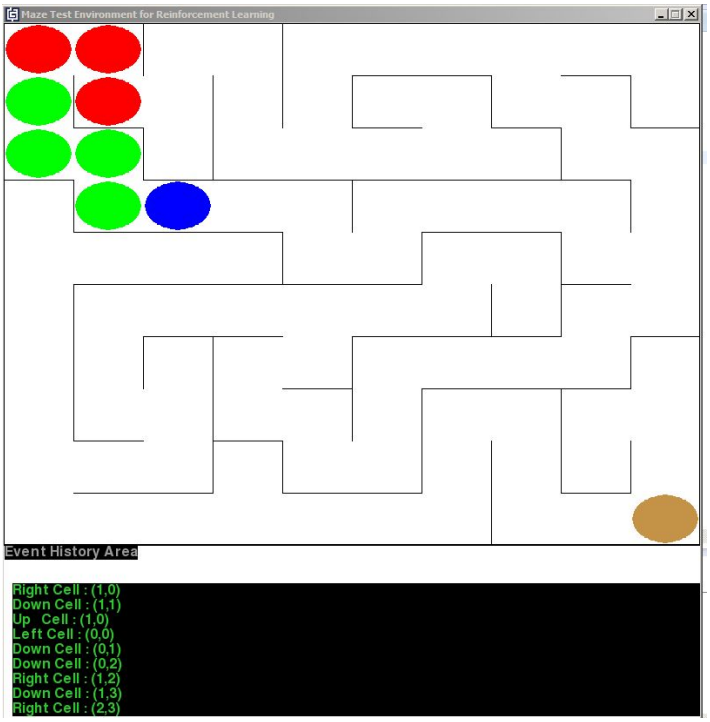


Figure D.9: Python maze simulation environment

If the maze has no loops, the LSR rule always gets you to the end of the maze. Pseudocode of LSR is illustrated in Algorithm 7.

Algorithm 7 Pseudocode of Wall Follower based on LSR Rule

```
repeat
    if Left is open then
        Turn Left
    else if Straight is open then
        Go Straight
    else if Right is open then
        Turn Right
    else
        Turn Back
    end if
until ¬not target point
```

D.4.2 Right Wall Follower (RSL Rule)

The right-hand rule, RSL rule, is similar to LSR, the only difference lies in the wall being followed. The RSL rule states Right direction has highest priority compared to Straight and Left directions while there are options for turns. Likewise, Straight has higher priority than Left turn. The order of precedence can be summarized as:

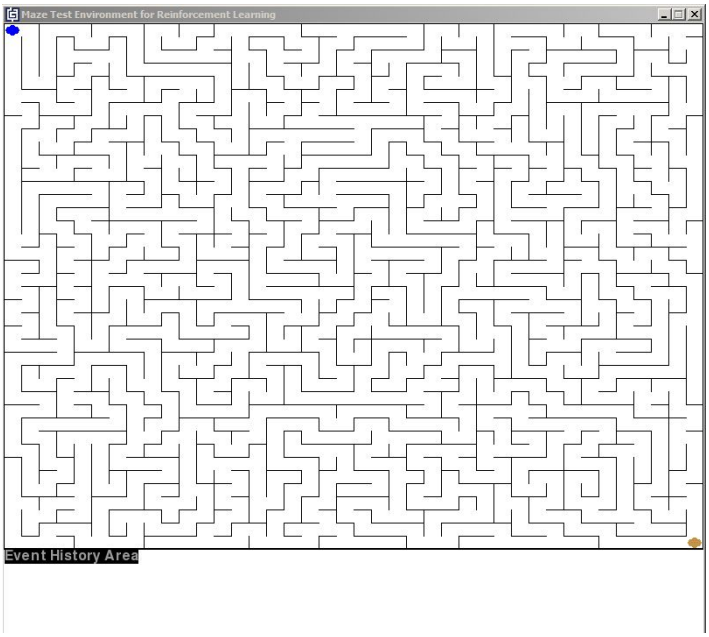


Figure D.10: Maze simulation environment with the size of 40 by 40 on Python

$$\text{Right} \Rightarrow \text{Straight} \Rightarrow \text{Left}$$

If the maze has no loops, the RSL rule always gets you to the end of the maze. Pseudocode of RSL is similar to Algorithm 7 except checking the order of possible turns.

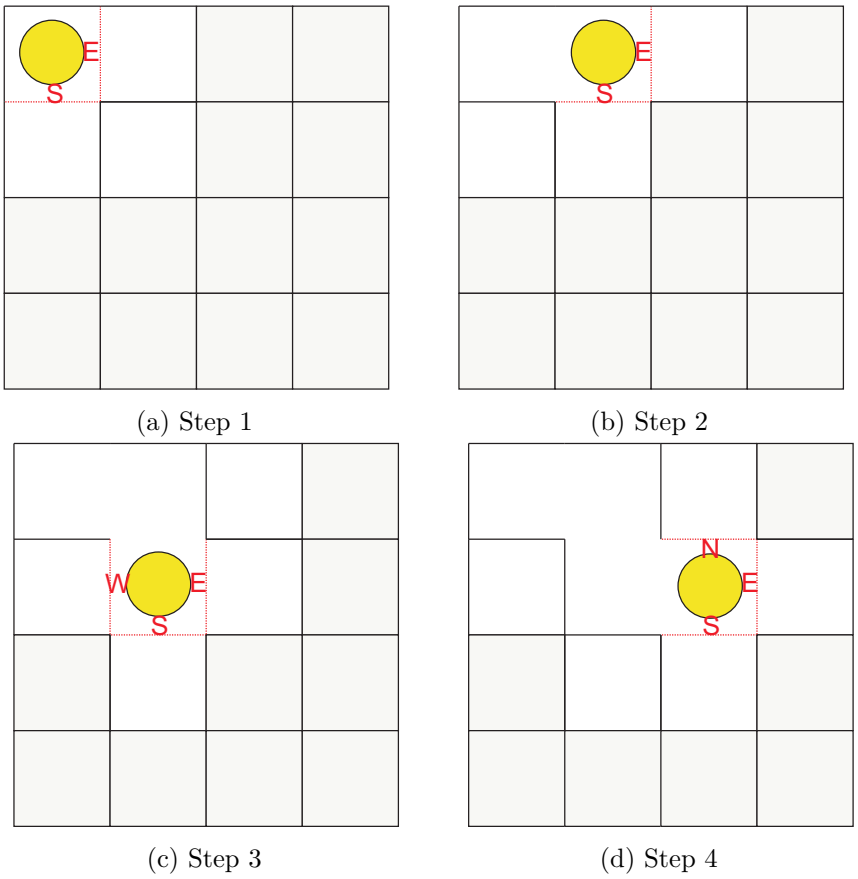
D.5 Example Screenshot from Workspace

D.6 Prerequisites

Scripts during all experiment executions are performed under the Python programming language which is also used for data analysis, file organizations of saved sessions and evaluation of the results.

The following Python-based software packages are used:

- Python installation: Python 2.7.
- SNN simulator: BRIAN2



(a) Step 1 (b) Step 2

(c) Step 3 (d) Step 4

Figure D.11: Maze generation steps based on Backtrack Recursion algorithm.

- Numerical computations: numpy as a part of the SciPy package, it is an efficient array/matrix representation, linear algebra functions (including matrix operations), etc.
- Visualization: matplotlib, pylab, brian2tools, spikeplot to visualize the results of analysis and simulations.

D.7 Software Contributions

Brian2 is Python based simulator for Spiking Neural Networks (SNNs):

`https://brian2.readthedocs.io/en/stable/introduction/release_notes.html`

Brian2tools is Python based visualization tool for Brian2:

`https://brian2tools.readthedocs.io/en/stable/release_notes.html`

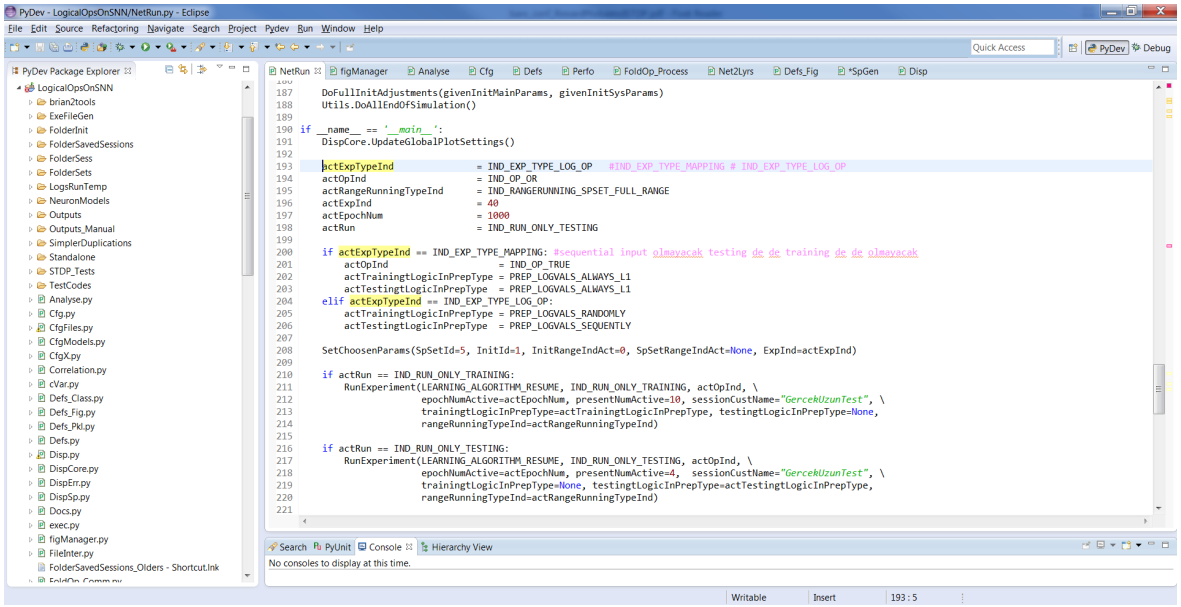


Figure D.12: Example screenshot from workspace

Appendix E

Credits

TexStudio 2.11.0 editor of LaTeXis used to edit and typeset this dissertation. The majority of diagrams used here created with Corel DRAW X8 and Corel PHOTO-PAINT X8. Also, Araxis Merge has been used to control files/folders/code through versions.

Main programming language used in the thesis is Python 2.7. Anaconda 64-bit distribution of Python 2.7 has been used. As a Spiking Neural Network simulator in the Python programming language, the latest stable version of Brian2 (2.0rc3) has been used. Furthermore, there are many additional used Python modules: NumPy, SciPy, Matplotlib, Pylab, Brian2tools, Spikeplot, PyGame, Tkinter and WxPython.

One of the popular IDE for developing software, Eclipse KEPLER is used with Python plug-in PyDev for Eclipse. Eclipse is an open-source platform that provides convenient code editing and debugging tools. PyDev is a Python IDE that runs on top of Eclipse.

Glossary

Abbreviations

STDP	Spike-timing Dependent Plasticity
Hebbian-STDP	Hebbian Spike-timing Dependent Plasticity
anti-Hebbian-STDP	anti-Hebbian Spike-timing Dependent Plasticity
Symmetric-STDP	Symmetric Spike-timing Dependent Plasticity
Asymmetric-STDP	Asymmetric Spike-timing Dependent Plasticity
BP	Back Propagation
SpikeProp	Spike Propagation
ReSuMe	Remote Supervised Method
DelReSuMe	Delayed Remote Supervised Method
R-STDP	Reward-modulated Spike-timing Dependent Plasticity
R-max	Reward-maximisation
VP	Victor & Purpura Distance
CF	Coincidence Factor
ScD	Schreiber Distance
vRD	van Rossum Distance
vRE	van Rossum Error
LCE	Logic Classification Error

WH	Widrow-Hoff
ML	Machine Learning
RL	Reinforcement Learning
MDP	Markov Decision Process
DTMC	Discrete Time Markov Chain
TD	Temporal-Difference
MCTS	Monte Carlo tree search
PSP	Post-synaptic Potential
EPSP	Excitatory Post-synaptic Potential
IPSP	Inhibitory Post-synaptic Potential
LTP	Long Term Potentiation
LTD	Long Term Depression
NMDAR-dependent	N-methyl-D-aspartate receptor dependent
DA	Dopamine
CNS	central nervous system
SD	standard deviation
MLP	Multi-Layer Perceptron
NN	Neural Network
ANN	Artificial Neural Network
SNN	Spiking Neural Network
FNN	Feed-forward Network
RNN	Recurrent Neural Network
HH	Hodgkin-Huxley
IF	Integrate-and-Fire

LIF	Leaky-Integrate-and-Fire
SRM	Spike Response Model
IM	Izhikevich model
NEF	Neural Engineering Framework
NEST	Neuron Simulation Tool
GENESIS	General Neural Simulation System
ISI	Inter-spike Interval
GUI	Graphical User Interface
SWIG	The Simplified Wrapper and Interface Generator
TRN	Training
TST	Testing
E	Epoch
P	Presentation
EXP	Experiment
Norm	Normalized
L	Left turn
R	Right turn
S	Straight (No turn)
U	U turn
LSR	Left-Straight-Right
RSL	Right-Straight-Left
RepRule	Replacement Rule
BscRepLSR	Basic Replacement Rules for LSR
ExtRepLSR	Extended Replacement Rules for LSR

BscRepRSL	Basic Replacement Rules for RSL
ExtRepRSL	Extended Replacement Rules for RSL
FP	False-Positive
FN	False-Negative
TP	True-Positive
TN	True-Pegative
SLAM	Simultaneous Localization and Mapping
sp	spike
ms	milisecond(s)
sec	second(s)
min	minute(s)

Bibliography

Abbott, L. F. (1999), 'Lapique's introduction of the integrate-and-fire model neuron (1907)', *Brain Research Bulletin*, 50(5-6), pp. 303–304.

Abbott, L. F., Fahri, E. & Gutmann, S. (1991), 'The path integral for dendritic trees', *Biol. Cybern.*, 66, pp. 49–60.

Adibi, P., Meybodi, M. R. & Safabakhsh, R. (2005), 'Unsupervised learning of synaptic delays based on learning automata in an rbf-like network of spiking neurons for data clustering', *Neurocomputing*, 64, pp. 335–357.

Adrian, E. D. (1926), 'The impulses produced by sensory nerve endings: Part I', *The Journal of Physiology*, 61(1), pp. 49–72.

Arias-Carrion, O. & Poppel, E. (2007), 'Dopamine, learning, and reward-seeking behavior.', *Acta neurobiologiae experimentalis*, 67(4), pp. 481–488.

Arias-Carrion, O., Stamelou, E., M. Murillo-Rodriguez, Menendez-Gonzalez, M. & Poppel1, E. (2010), 'Dopaminergic reward system : a short integrative review', *International Archives of Medicine*, 3(24).

Bailey, C. H., Giustetto, M., Huang, Y. Y., Hawkins, R. D. & Kandel, E. R. (2000), 'Is heterosynaptic modulation essential for stabilizing hebbian plasticity and memory?', *Nat. Rev. Neurosci.*, 1(1), pp. 11–20.

Bar-Yam, Y. (1997), Dynamics of Complex Systems, Addison-Wesley Press, Reading, Massachusetts, first edition edition.

Bekolay, T. (2011), Learning in large-scale spiking neural networks, Ph.D. thesis, the University of Waterloo.

Bellman, R. (1958), 'On a routing problem.', *Quarterly of Applied Mathematics*, 16, pp. 87–90.

Bellman, R. E. (1957), Dynamic Programming, Princeton University Press, Princeton, NJ, republished 2003: dover edition.

Ben Achour, S. & Pascual, O. (2010), 'Glia: the many ways to modulate synaptic plasticity', *Neurochem. Int.*, 57(4), pp. 440–445.

Beninger, R. (1983), 'The role of dopamine in locomotor activity and learning', *Brain Research Reviews*, 6, pp. 173–196.

Bi, G. & Poo, M. (2001), 'Synaptic modification by correlated activity: Hebb's postulate revisited', *Annu. Rev. Neurosci.*, 24, pp. 139–166.

Bi, G. Q. & Poo, M. M. (1998), 'Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type', *J. Neurosci.*, 18(24), pp. 10464–10472.

Bissiere, S., Humeau, Y. & Luthi, A. (2003), 'Dopamine gates ltp induction in lateral amygdala by suppressing feedforward inhibition', *Nat. Neurosci.*, 6(6), pp. 587–592.

Bliss, T. V. & Gardner-Medwin, A. R. (1973), 'Long-lasting potentiation of synaptic transmission in the dentate area of the unanaestetized rabbit following stimulation of the perforant path', *J. Physiol. (Lond.)*, 232(2), pp. 357–374.

Bliss, T. V. P. & Collingridge, G. L. (1993), 'A synaptic model of memory: long-term potentiation in the hippocampus', *Nature*, 361, pp. 31–39.

Blynel, J. & Floreano, D. (2002), 'Levels of dynamics and adaptive behavior in evolutionary neural controllers.', *Operations Research*, 50(1), pp. 272–281.

Bohte, S. M. (2004), 'The evidence for neural information processing with precise spike-times: A survey', *Natural Computing*, 3(2), pp. 195–206.

Bohte, S. M., Bohte, E. M., Poutre, H. L. & Kok, J. N. (2002a), 'Unsupervised clustering with spiking neurons by sparse temporal coding and multi-layer rbf networks', *IEEE Trans. Neural Netw*, 13, pp. 426–435.

Bohte, S. M., Kok, J. N. & Poutre, H. L. (2002b), 'Error-backpropagation in temporally encoded networks of spiking neurons', *Neurocomputing*, 48(1), pp. 17–37.

Booij, O. & Nguyen, H. (2005), 'A gradient descent rule for spiking neurons emitting multiple spikes', *Information Processing Letters*, 95(6), pp. 552–558.

Bower, J. M. & Beeman, D. (1998), The book of GENESIS: Exploring realistic neural models with the General Neural Simulation System, Springer, New York, second edition.

Bower, J. M. & Beeman, D. (2003), 'Genesis', in 'Exploring Realistic Neural Models with the GEneral NEural SImulation System', chapter Compartmental Modeling, Allan Wylde of TELOS, internet edition.

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., Diesmann, M., Morrison, A., Goodman, P. H., Harris, F. C., Zirpe, M., Natschlager, T., Pecevski, D., Ermentrout, B., Djurfeldt, M., Lansner, A., Rochel, O., Vieville, T., Muller, E., Davison, A. P., El Boustani, S. & Destexhe, A. (2007), 'Simulation of networks of spiking neurons: a review of tools and strategies', *J Comput Neurosci*, 23(3), pp. 349–398.

Brown, E., Richard & Milner, M., Peter (2003), 'The legacy of donald o. hebb: more than the hebb synapse', *Nature Reviews Neuroscience*, 4(12), pp. 1013–1019.

Burkitt, A. N. (2006), 'A review of the integrate-and-fire neuron model: I. homogeneous synaptic input', *Biol Cybern*, 95(1), pp. 1–19.

Butts, D. A., Weng, C., Jin, J., Yeh, C. I., Lesica, N. A., Alonso, J. M. & Stanley, G. B. (2007), 'Temporal precision in the neural code and the timescales of natural vision', *Nature*, 449(7158), pp. 92–95.

Carey, M. R., Medina, J. F. & Lisberger, S. G. (2005), 'Instructive signals for motor learning from visual cortical area mt', *Nat. Neurosci.*, 8(6), pp. 813–819.

Chistiakova, M., Bannon, N. M., Chen, J. Y., Bazhenov, M. & Volgushev, M. (2015), 'Homeostatic role of heterosynaptic plasticity: models and experiments', *Front Comput Neurosci*, 9, p. 89.

Citri, A. & Malenkas, R. C. (2008), 'Synaptic plasticity: Multiple forms, functions, and mechanisms', *Neuropsychopharmacology (2008)*, 33, pp. 18–41.

Coulom, R. (2006), 'Efficient selectivity and backup operators in monte-carlo tree search', in 'Proceedings of the 5th International Conference on Computers and Games', CG'06, pp. 72–83.

Crites, R. H. & Barto, A. (1998), 'Elevator group control using multiple reinforcement learning agents', *Machine Learning*, 33, pp. 235–262.

Davies, M. (2002), 'The neuron: size comparison', `http://www.ualberta.ca/~neuro/OnlineIntro/NeuronExample.htm/`, [Online; accessed 19-May-2014].

Dayan, P. & Abbott, L. F. (2005), Theoretical Neuroscience : Computational and Mathematical Modeling of Neural Systems, The MIT Press, London, UK, first edition.

Dayan, P. & Hinton, G. E. (1997), 'Using expectation-maximization for reinforcement learning', *Neural Computation*, 9, pp. 271–278.

DeFelipe, J. & Farinas, I. (1992), 'The pyramidal neuron of the cerebral cortex: morphological and chemical characteristics of the synaptic inputs', *Prog. Neurobiol.*, 39(6), pp. 563–607.

Deisenroth, M. P., Neumann, G. & Peters, J. (2011), 'A survey on policy search for robotics', *Foundations and Trends in Robotics*, 1-2(7), pp. 1–142.

Destexhe, A., Rudolph, M. & Pare, D. (2003), 'The high-conductance state of neocortical neurons in vivo', *Nat. Rev. Neurosci.*, 4(9), pp. 739–751.

DiCarlo, J. J., Zoccolan, D. & Rust, N. C. (2012), 'How does the brain solve visual object recognition?', *Neuron*, 73(3), pp. 415–434.

Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U. & Neftci, E. (2016), 'Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware', *arXiv*.

Dreyfus, S. (2002), 'Richard bellman on the birth of dynamic programming', *Operations Research*, 50(1), pp. 48–51.

Dudeney, H. E. (1970), Amusements in Mathematics, Dover Publications, New York, Dover, first edition.

El-Laithy, K. & Bogdan, M. (2011), 'A reinforcement learning framework for spiking networks with dynamic synapses', *Comput Intell Neurosci*, 2011, p. 869348.

Esser, S. K., Andreopoulos, E., Datta, P., Barch, D., Amir, A., Arthur, J., Cassidy, A., Flickner, M., Merolla, P., Ch, S., Basilico, N., Carpin, S., Zimmerman, T., Zee, F., Alvarez-icaza, R., Kusnitz, J. A., Wong, T. M., Risk, W. P., Mcquinn, E., Nayak, T. K., Singh, R. & Modha, D. S. (2013), 'Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores', .

Eurich, C. W., Pawelzik, K., Ernst, U., Thiel, A., Cowan, J. D. & Milton, J. G. (2000), 'Delay adaptation in the nervous system', *Neurocomputing*, 32-33, pp. 741–748, ISSN 0925-2312.

Even, S. (2011), Graph Algorithms, Cambridge University Press, Cambridge, second edition.

Farries, M. A. & Fairhall, A. L. (2007), 'Reinforcement learning with modulated spike timing dependent synaptic plasticity', *J. Neurophysiol.*, 98(6), pp. 3648–3665.

Fellous, J. M. & Suri, R. E. (2003), 'The hand book of brain theory and neural networks', in Michael, A. A., ed., 'The Roles of Dopamine', MIT Press, Cambridge, MA, second edition edition.

Fisher, R. A. (1936), 'The use of multiple measurements in taxonomic problems', *Annals of Eugenics*, 7(2), pp. 179–188.

Florian, R. V. (2003), 'Autonomous artificial intelligent agents.', Technical Report Coneural-03-01, Center for Cognitive and Neural Studies.

Florian, R. V. (2007), 'Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity', *Neural Comput*, 19(6), pp. 1468–1502.

Fremaux, N. & Gerstner, W. (2015), 'Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules', *Front Neural Circuits*, 9, p. 85.

Fremaux, N., Sprekeler, H. & Gerstner, W. (2010), 'Functional requirements for reward-modulated spike-timing-dependent plasticity', *The Journal of Neuroscience*, 30(40), pp. 13326–13337.

Fremaux, N., Sprekeler, H. & Gerstner, W. (2013), 'Reinforcement learning using a continuous time actor-critic framework with spiking neurons', *PLoS Comput. Biol.*, 9(4), p. e1003024.

Froemke, R. C., Poo, M. M. & Dan, Y. (2005), 'Spike-timing-dependent synaptic plasticity depends on dendritic location', *Nature*, 434(7030), pp. 221–225.

Gawne, T. J., Kjaer, T. W. & Richmond, B. J. (1996), 'Latency: another potential code for feature binding in striate cortex', *J. Neurophysiol.*, 76(2), pp. 1356–1360.

Geisler, C. & Goldberg, J. (1966), 'A stochastic model of repetitive activity of neurons', *Biophys. J.*, 6, pp. 53–69.

Gerstein, G. L. & Mandelbrot, B. (1964), 'Random walk models for the spike activity of a single neuron', *Biophys. J.*, 4, pp. 41–68.

Gerstner, W., Kempter, R., Hemmen, J. & Wagner, H. (1996), 'A neuronal learning rule for sub-millisecond temporal coding', *Nature*, 383, pp. 76–78.

Gerstner, W. & Kistler, W. M. (2002), Spiking Neuron Models : Single Neurons, Populations, Plasticity, Cambridge University Press, Cambridge, UK, first edition.

Gerstner, W., Kistler, W. M., Naud, R. & Paninski, L. (2014), Neuronal dynamics : From Single Neurons to Networks and Cognition, Cambridge University Press, Cambridge, first edition.

Gewaltig, M.-O. & Diesmann, M. (2007), 'Nest (neural simulation tool)', *Scholarpedia*, 2(4), pp. 14–30.

Gilson, M. & Fukai, T. (2011), 'Stability versus neuronal specialization for STDP: long-tail weight distributions solve the dilemma', *PLoS ONE*, 6(10), p. e25339.

Goodman, D. & Brette, R. (2008), 'Brian: a simulator for spiking neural networks in python', *Front Neuroinform*, 2, p. 5.

Gruning, A. & Sporea, I. (2012), 'Supervised learning of logical operations in layered spiking neural networks with spike train encoding', *Neural Processing Letters*, 36(2), pp. 117–134.

Gu, Q. (2002), 'Neuromodulatory transmitter systems in the cortex and their role in cortical plasticity', *Neuroscience*, 111(4), pp. 815–835.

Gutig, R., Aharonov, R., Rotter, S. & Sompolinsky, H. (2003), 'Learning input correlations through nonlinear temporally asymmetric hebbian plasticity', *J. Neurosci.*, 23(9), pp. 3697–3714.

Haas, J. S. & White, J. A. (2002), 'Frequency selectivity of layer ii stellate cells in the medial entorhinal cortex', *J. Neurophysiol.*, 88(5), pp. 2422–2429.

Hebb, D. O. (1949), The Organization of Behavior: A Neuropsychological Theory, Wiley and Sons, New York, first edition.

Hertz, J., Krogh, A. & Palmer, R. G. (1991), Introduction to the theory of neural computation, Addison-Wesley Pub. Co., first edition.

Hill, A. (1936), 'Excitation and accomodation in nerve', *Proc. R. Soc. Lond. B*, 119, pp. 305–355.

Hille, B. (2001), Ionic Channels of Excitable Membranes, Sinauer Associates, Sunderland, MA, third edition.

Hines, M. L. & Carnevale, N. T. (1997), 'The neuron simulation environment', *Neural Computation*, 9, pp. 1179–1209.

Hines, M. L. & Carnevale, N. T. (2001), 'Neuron: a tool for neuroscientists', *Neuroscientist*, 7(2), pp. 123–135.

Hirano, T. (2013), 'Long-term depression and other synaptic plasticity in the cerebellum', *Proc. Jpn. Acad., Ser. B, Phys. Biol. Sci.*, 89(5), pp. 183–195.

Hodgkin, A. L. & Huxley, A. F. (1952), 'A quantitative description of membrane current and its application to conduction and excitation in nerve', *Journal of Physiology*, 117, pp. 500–544.

Izhikevich, E. M. (2003), 'Simple model of spiking neurons', *IEEE Transactions on Neural Networks*, 14(6), pp. 1569–1572.

Izhikevich, E. M. (2004), 'Which model to use for cortical spiking neurons?', *IEEE Trans. Neural Networks*, 15(5), pp. 1063–1070.

Izhikevich, E. M. (2007a), Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting, The MIT Press, Cambridge, Massachusetts, first edition.

Izhikevich, E. M. (2007b), 'Solving the distal reward problem through linkage of stdp and dopamine signaling', *Cereb. Cortex*, 17(10), pp. 2443–2452.

Izhikevich, E. M., Gally, J. A. & Edelman, G. M. (2004), 'Spike-timing dynamics of neuronal groups', *Cereb. Cortex*, 14(8), pp. 933–944.

Jacobson, M. (1993), Foundations of Neuroscience, Plenum Press, New York, USA, first edition.

Johansen, J. P., Diaz-Mataix, L., Hamanaka, H., Ozawa, T., Ycu, E., Koivumaa, J., Kumar, A., Hou, M., Deisseroth, K., Boyden, E. S. & LeDoux, J. E. (2014), 'Hebbian and neuromodulatory mechanisms interact to trigger associative memory formation', *Proc. Natl. Acad. Sci. U.S.A.*, 111(51), pp. E5584–5592.

Joris, P. X., Schreiner, C. E. & Rees, A. (2004), 'Neural processing of amplitude-modulated sounds', *Physiol. Rev.*, 84(2), pp. 541–577.

Kaelbling, L. P., Littman, M. L. & Moore, A. W. (1996), 'Reinforcement learning : A survey', *Journal of Artifcial Intelligence Research*, 4, pp. 237–285.

Kandel, E. R., Schwartz, J. H. & Jessel, T. M. (2000), Principles of Neural Science, McGraw-Hill Companies, Montreal, Canada, fourth edition.

Kasabov, N. (2012), Evolving Spiking Neural Networks and Neurogenetic Systems for Spatio- and Spectro-Temporal Data Modelling and Pattern Recognition, pp. 234–260, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-642-30687-7.

Kasinski, A. & Ponulak, F. (2006), 'Comparison of supervised learning methods for spike time coding in spiking neural networks', *Int. J. Appl. Math. Comput. Sci*, 16(1), pp. 101–113.

Kempter, R., Gerstner, W. & van Hemmen, J. L. (1999), 'Spike-based compared to rate-based hebbian learning', in 'Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II', pp. 125–131, MIT Press, Cambridge, MA, USA.

Kerr, R. R., Burkitt, A. N., Thomas, D. A., Gilson, M. & Grayden, D. B. (2013), 'Delay selection by spike-timing-dependent plasticity in recurrent networks of spiking neurons receiving oscillatory inputs', *PLoS Comput. Biol.*, 9(2), p. e1002897.

Kistler, W. M. (2002), 'Spike-timing dependent synaptic plasticity: a phenomenological framework', *Biol Cybern*, 87(5-6), pp. 416–427.

Kistler, W. M., Gerstner, W. & van Hemmen, J. L. (1997), 'Reduction of the hodgkin-huxley equations to a single-variable threshold model', *Neural Computation*, 9(5), pp. 1015–1045.

Koch, C., Rapp, M. & Segev, I. (1996), 'A brief history of time (constants)', *Cereb. Cortex*, 6(2), pp. 93–101.

Koch, C. & Segev, I. (1999), Methods in neuronal modeling : from ions to networks, MIT Press, second edition.

Kriesel, D. (2014), 'A brief introduction to neural networks (zeta2-en)', `"http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf/"`, [Online; accessed 12-May-2014].

Law, G. (2013), 'Quantitative comparison of flood fill and modified flood fill algorithms', *International Journal of Computer Theory and Engineering*, 5(3), pp. 87–90.

Legenstein, R., Naeger, C. & Maass, W. (2005), 'What can a neuron learn with spike-timing-dependent plasticity?', *Neural Computation*, 17(11), pp. 2337–2382.

Legenstein, R., Pecevski, D. & Maass, W. (2008), 'A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback', *PLoS Comput. Biol.*, 4(10), p. e1000180.

Lipton, Z. C., Berkowitz, J. & Elkan, C. (2015), 'A critical review of recurrent neural networks for sequence learning', *arXiv preprint arXiv:1506.00019*.

Lomo, T. (2003), 'The discovery of long-term potentiation', *Philosophical Transactions of the Royal Society of London*, 358(1432), pp. 617–620.

Lynch, G. S., Dunwiddie, T. & Gribkoff, V. (1977), 'Heterosynaptic depression: a postsynaptic correlate of long-term potentiation', *Nature*, 266(5604), pp. 737–739.

Maass, W. (1996), 'Networks of spiking neurons: The third generation of neural network models', *NeuralNetworks*, 10(9), pp. 1659–1671.

Martin, S. J., Grimwood, P. D. & Morris, R. G. (2000), 'Synaptic plasticity and memory: an evaluation of the hypothesis.', *Neurosci.*, 23, pp. 649–711.

Masquelier, T., Guyonneau, R. & Thorpe, S. J. (2008), 'Spike timing dependent plasticity finds the start of repeating patterns in continuous spike trains', *PLoS ONE*, 3(1), p. e1377.

Massey, P. V. & Bashir, Z. I. (2007), 'Long-term depression: multiple forms and implications for brain function', *Trends Neurosci.*, 30(4), pp. 176–184.

Matignon, L., Laurent, G. & Piat, N. L. F. (2006), 'Reward function and initial values : Better choices for accelerated goal-directed reinforcement learning', *Artificial Neural Networks - ICANN 2006*, 4131(1), pp. 840–849.

Mishra, S. & Bande, P. (2008), 'Maze solving algorithms for micro mouse in signal image technology and internet based systems', *IEEE International Conference on 2008*, 1, pp. 86–93.

Moore, J. W. & Ramon, F. (1976), 'On numerical integration of the hodgkin and huxley equations for a membrane action potential', *Journal of Theoretical Biology*, 45, pp. 249–273.

Morrison, A., Diesmann, M. & Gerstner, W. (2008), 'Phenomenological models of synaptic plasticity based on spike timing', *Biol Cybern*, 98(6), pp. 459–478.

Mulder, W. D., Bethard, S. & Moens, M.-F. (2015), 'A survey on the application of recurrent neural networks to statistical language modeling', *Computer Speech and Language*, 30(1), pp. 61–98, ISSN 0885-2308.

Natschlager, T. & Ruf, B. (1998), 'Spatial and temporal pattern analysis via spiking neurons', *Network*, 9(3), pp. 319–332.

Naud, R., Marcille, N., Clopath, C. & Gerstner, W. (2008), 'Firing patterns in the adaptive exponential integrate-and-fire model', *Biol Cybern*, 99(4–5), pp. 335–347.

Nicholls, J., Martin, A. & Wallace, B. (1992), From Neuron to Brain, Sinauer Assoc., third edition.

Nicoll, R. A. (2017), 'A brief history of long-term potentiation', *Neuron*, 93(2), pp. 281–290.

Norris, J. R. (1997), Discrete-time Markov chains, pp. 1–59, Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press, Cambridge.

Nowe, A. & Brys, T. (2016), 'A gentle introduction to reinforcement learning', *Schockaert S., Senellart P. (eds) Scalable Uncertainty Management. SUM 2016*, 9858, pp. 18–32.

Otani, S., Daniel, H., Roisin, M. P. & Crepel, F. (2003), 'Dopaminergic modulation of long-term synaptic plasticity in rat prefrontal neurons', *Cereb Cortex.*, 13(11), pp. 1251–1256.

Paille, V., Fino, E., Du, K., Morera-Herreras, T., Perez, S., Kotaleski, J. H. & Venance, L. (2013), 'Gabaergic circuits control spike-timing-dependent plasticity', *J. Neurosci.*, 33(22), pp. 9353–9363.

Panzeri, S., Brunel, N., Logothetis, N. K. & Kayser, C. (2010), 'Sensory neural codes using multiplexed temporal scales', *Trends Neurosci.*, 33(3), pp. 111–120.

Pastalkova, E., Serrano, P., Pinkhasova, D., Wallace, E., Fenton, A. A. & Sacktor, T. C. (2006), 'Storage of spatial information by the maintenance mechanism of ltp.', *Science*, 313, pp. 1141–1144.

Paugam, H., Moisy & Bohte, S. M. (2012), 'Computing with spiking neuron networks.', in Rozenberg, G., Back, T. & Kok, J. N., eds., 'Handbook of Natural Computing', pp. 335–376, Springer.

Pawlak, V. & Kerr, J. N. (2008), 'Dopamine receptor activation is required for corticostriatal spike-timing-dependent plasticity', *J. Neurosci.*, 28(10), pp. 2435–2446.

Peters, J. & Schaal, S. (2008), 'Learning to control in operational space.', *International Journal of Robotics Research*, 27, pp. 197–212.

Pfister, J.-P., Toyoizumi, T., Barber, D. & Gerstner, W. (2006), 'Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning', *Neural Computation*, 18(6), pp. 1318–1348.

Ponulak, F. (2008), 'Analysis of the resume learning process for spiking neural networks', *Applied Mathematics and Computer Science*, 18(2), pp. 117–127.

Ponulak, F. & Kasinski, A. (2010), 'Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting', *Neural Comput*, 22(2), pp. 467–510.

Potjans, W., Morrison, A. & Diesmann, M. (2009), 'A spiking neural network model of an actor-critic learning agent', *Neural Computation*, 21(2), pp. 301–339.

Pouget, A., Dayan, P. & Zemel, R. (2000), 'Information processing with population codes', *Nat. Rev. Neurosci.*, 1(2), pp. 125–132.

Rall, W. (1959), 'Branching dendritic trees and motoneuron membrane resistivity', *Exp. Neurol.*, 1, pp. 491–527.

Rangel, A., Camerer, C. & Montague, P. R. (2008), 'A framework for studying the neurobiology of value-based decision making', *Nat. Rev. Neurosci.*, 9(7), pp. 545–556.

Rasmusson, D. D. (2000), 'The role of acetylcholine in cortical synaptic plasticity', *Behav. Brain Res.*, 115(2), pp. 205–218.

Rescorla, R. & Wagner, A. (1972), 'Classical conditioning ii', in 'Current Theory and Research', chapter A theory of pavlovian conditioning: variations in the effectiveness of reinforcement and nonreinforcement, pp. 64–99, Prokasy AH, Black WF, eds., New York: Appleton Century Crofts.

Reynolds, J. N., Hyland, B. I. & Wickens, J. R. (2001), 'A cellular mechanism of reward-related learning', *Nature*, 413(6851), pp. 67–70.

Risken, H. (1989), The Fokker-Planck equation, Springer-Verlag, Berlin, second edition.

Roberts, P. D. & Bell, C. C. (2002), 'Spike timing dependent synaptic plasticity in biological systems', *Biol Cybern*, 87(5-6), pp. 392–403.

Rojas, R. (1996), Neural Networks: A Systematic Introduction, Springer, first edition.

Rubin, J., Lee, D. D. & Sompolinsky, H. (2001), 'Equilibrium properties of temporally asymmetric Hebbian plasticity', *Phys. Rev. Lett.*, 86(2), pp. 364–367.

Rusu, C. V. & Florian, R. V. (2014), 'A new class of metrics for spike trains', *Neural Comput*, 26(2), pp. 306–348.

Sabatini, B. L. & Regehr, W. G. (1999), 'Timing of synaptic transmission', *Annual Review of Physiology*, 61(1), pp. 521–542.

Salvioni, P., Murray, M. M., Kalmbach, L. & Bueti, D. (2013), 'How the visual brain encodes and keeps track of time', *J. Neurosci.*, 33(30), pp. 12423–12429.

Schmidt, R. A. & Lee, T. D. (1999), Motor control and learning: a behavioural emphasis, Champaign: Human Kinetics, third edition.

Schrauwen, B. & Campenhout, J. V. (2004), 'Extending spikeprop', *Proceedings of the International Joint Conference on Neural Networks*, 1, pp. 471–476.

Schreiber, S., Fellous, J. M., Whitmer, D., Tiesinga, P. & Sejnowski, T. J. (2003), 'A new correlation-based measure of spike timing reliability', *Neurocomputing*, 52-54, pp. 925–931.

Schultz, W. (1998), 'Predictive reward signal of dopamine neurons', *J Neurophysiol.*, 80(1), pp. 1–27.

Schultz, W. (2007), 'Behavioral dopamine signals', *Trends in neurosciences*, 30(5), pp. 203–210.

Schultz, W., Dayan, P. & Montague, P. R. (1997), 'A neural substrate of prediction and reward', *Science*, 275(5306), pp. 1593–1599.

Senn, W., Markram, H. & Tsodyks, M. (2001), 'An algorithm for modifying neurotransmitter release probability based on pre- and postsynaptic spike timing', *Neural Comput*, 13(1), pp. 35–67.

Senn, W. & Pfister, J. (2014), 'Reinforcement learning in cortical networks', in 'Encyclopedia of Computational Neuroscience', Springer.

Seol, G. H., Ziburkus, J., Huang, S., Song, L., Kim, I. T., Takamiya, K., Huganir, R. L., Lee, H. K. & Kirkwood, A. (2007), 'Neuromodulators control the polarity of spike-timing-dependent synaptic plasticity', *Neuron*, 55(6), pp. 919–929.

Seung, H. S. (2003), 'Learning in spiking neural networks by reinforcement of stochastic synaptic transmission', *Neuron*, 40(6), pp. 1063–1073.

Shmiel, T., Drori, R., Shmiel, O., Ben-Shaul, Y., Nadasdy, Z., Shemesh, M., Teicher, M. & Abeles, M. (2005), 'Neurons of the cerebral cortex exhibit precise interspike timing in correspondence to behavior', *Proc. Natl. Acad. Sci. U.S.A.*, 102(51), pp. 18655–18657.

Silver, D. (2009), Reinforcement learning and simulation-based search in computer Go, Ph.D. thesis, University of Alberta, Edmonton, Alta., Canada.

Singh, S. & Bertsekas, D. P. (1997), 'Reinforcement learning for dynamic channel allocation in cellular telephone systems', *In Advances in Neural Information Processing Systems*, 9, pp. 974–980.

Sjostrom, J. & Gerstner, W. (2012), Spike-Timing-Dependent Plasticity : Synaptic Neuroscience, Frontiers Media SA, New York, USA, first edition.

Sjostrom, P. J., Rancz, E. A., Roth, A. & Hausser, M. (2008), 'Dendritic excitability and synaptic plasticity', *Physiol. Rev.*, 88(2), pp. 769–840.

Sjostrom, P. J., Turrigiano, G. G. & Nelson, S. B. (2001), 'Rate, timing, and cooperativity jointly determine cortical synaptic plasticity', *Neuron*, 32(6), pp. 1149–1164.

Song, S., Miller, K. & Abbott, L. (2000), 'Competitive hebbian learning through spike-timing-dependent synaptic plasticity', *Nat Neurosci*, 3(9), pp. 919–926.

Sporea, I. & Gruning, A. (2013), 'Supervised learning in multilayer spiking neural networks', *Neural Computation*, 25(2), pp. 473–509.

Squire, L. R. & Kandel, E. R. (1999), Memory: From Mind to Molecules (Scientific American Library), W H Freeman & Co, New York, first edition.

Stein, R. (1965), 'A theoretical analysis of neuronal variability', *Biophys. J.*, 5, pp. 173–194.

Stimberg, M., Goodman, D. F., Benichoux, V. & Brette, R. (2014), 'Equation-oriented specification of neural models for simulations', *Front Neuroinform*, 8, p. 6.

Sutton, R. S. (1988), 'Learning to predict by the methods of temporal differences', *Machine Learning.*, 3(1), pp. 9–44.

Sutton, R. S. & Barto, A. G. (1998), Reinforcement Learning: An Introduction, MIT Press, Cambridge MA, first edition.

Sutton, R. S., McAllester, D., Singh, S. & Mansour, Y. (1999), 'Policy gradient methods for reinforcement learning with function approximation', in 'Proceedings of the 12th International Conference on Neural Information Processing Systems', NIPS 99, pp. 1057–1063, MIT Press, Cambridge, MA, USA.

Taherkhani, A., Belatreche, A., Li, Y. & Maguire, L. P. (2015), 'DL-ReSuMe: A Delay Learning-Based Remote Supervised Method for Spiking Neurons', *IEEE Trans Neural Netw Learn Syst*, 26(12), pp. 3137–3149.

Tesauro, G. (1995), 'Temporal difference learning and td-gammon', *Commun. ACM*, 38(3), pp. 58–68.

Thorpe, S., Delorme, A. & Van Rullen, R. (2001), 'Spike-based strategies for rapid processing', *Neural Networks*, 14(6-7), pp. 715–725.

Thorpe, S., Fize, D. & Marlot, C. (1996), 'Speed of processing in the human visual system', *Nature*, 381(6582), pp. 520–522.

Thorpe, S. & Gautrais, J. (1998), Rank Order Coding, pp. 113–118, Springer US, Boston, MA, ISBN 978-1-4615-4831-7.

Thorpe, S. J. (1990), 'Spike arrival tiems: A highly efficient coding scheme for neural networks', in Eckmiller, G. & Hauske, G., eds., 'Parallel processing in neural systems', pp. 91–94, Elsevier.

Thrun, S. (2002), 'Robotic mapping: A survey', in Lakemeyer, G. & Nebel, B., eds., 'Exploring Artificial Intelligence in the New Millenium', Morgan Kaufmann, to appear.

Tiesinga, P., Fellous, J. M. & Sejnowski, T. J. (2008), 'Regulation of spike timing in visual cortical circuits', *Nat. Rev. Neurosci.*, 9(2), pp. 97–107.

Trappenberg, T. (2010), Fundamentals of Computational Neuroscience, Oxford University Press, New York, second edition.

Tuckwell, H. (1988), Introduction to theoretic neurobiology, Cambridge University Press, Cambridge, first edition.

Turrigiano, G. (1999), 'Homeostatic plasticity in neuronal networks: the more things change, the more they stay the same', *Trends Neurosci.*, 22(5), pp. 221–227.

Turrigiano, G. (2012), 'Homeostatic synaptic plasticity: local and global mechanisms for stabilizing neuronal function', *Cold Spring Harb Perspect Biol*, 4(1), p. a005736.

Turrigiano, G. G. (2008), 'The self-tuning neuron: synaptic scaling of excitatory synapses', *Cell*, 135(3), pp. 422–435.

van Rossum, M. C., Bi, G. Q. & Turrigiano, G. G. (2000), 'Stable hebbian learning from spike timing-dependent plasticity', *J. Neurosci.*, 20(23), pp. 8812–8821.

van Rossum, M. C. W. (2001), 'A novel spike distance', *Neural Computation 13*, 20, pp. 751–763.

Vasilaki, E., Fremaux, N., Urbanczik, R., Senn, W. & Gerstner, W. (2009), 'Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail', *PLoS Comput. Biol.*, 5(12), p. e1000586.

Venkata, V. P., Chitra, D. J., Karpagam, P. & Manju, P. D. (2011), 'Knowledge based reinforcement learning robot in maze environment', *International Journal of Computer Applications*, 14, pp. 975–8887.

Victor, J. D. & Purpura, K. P. (1996), 'Nature and precision of temporal coding in visual cortex: a metric-space analysis', *J. Neurophysiol.*, 76(2), pp. 1310–1326.

Victor, J. D. & Purpura, K. P. (1997), 'Metric-space analysis of spike trains: Theory, algorithms and application', *Network: Comput. Neural Syst.*, 8(2), pp. 127–164.

Vodopivec, T., Samothrakis, S. & Ster, B. (2017), 'On monte carlo tree search and reinforcement learning', *Journal of Artificial Intelligence Research*, 60, pp. 881–936.

Watt, A. & Desai, N. (2010), 'Homeostatic plasticity and stdp: keeping a neuron's cool in a fluctuating world', *Frontiers in Synaptic Neuroscience*, 2, p. 5.

Wickens, J. R. (2009), 'Synaptic plasticity in the basal ganglia', *Behav Brain Res.*, 199(1), pp. 119–128.

Widrow, M., B. Lehr (1990), '30 years of adaptive neural networks: perceptron, madaline, and backpropagation', *Proceedings of the IEEE*, 78(9), pp. 1415–1442.

Wikipedia, t. f. e. (2016), 'Artificial neuron model', `https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png`, [Online; accessed 13-Sep-2016].

Witten, I. H. (1977), 'An adaptive optimal controller for discrete-time markov environments', *Information and Control*, 34(4), pp. 286–295, ISSN 0019-9958.

Xie, X. & Seung, H. S. (2004), 'Learning in neural networks by reinforcement of irregular spiking', *Physical Review*, 69.

Xu, T. X. & Yao, W. D. (2010), 'D1 and d2 dopamine receptors in separate circuits cooperate to drive associative long-term potentiation in the prefrontal cortex', *Proc. Natl. Acad. Sci. U.S.A.*, 107(37), pp. 16366–16371.

Xu, Y., Zeng, X., Han, L. & Yang, J. (2013), 'A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks', *Neural Netw*, 43, pp. 99–113.

Yamauchi, B. M. & Beer, R. D. (1994), 'Sequential behavior and learning in evolved dynamical neural networks.', *Adaptive Behavior*, 2, pp. 219–246.

Zenke, F. & Gerstner, W. (2017), 'Hebbian plasticity requires compensatory processes on multiple timescales', *Philos. Trans. R. Soc. Lond., B, Biol. Sci.*, 372(1715).

Zhao, Y., Kosorok, M. R. & Zeng, D. (2009), 'Reinforcement learning design for cancer clinical trials', *Statistics in Medicine*, 28, pp. 3294–3315.

Zucker, R. S. & Regehr, W. (2002), 'Short-term synaptic plasticity.', *Annu. Rev. Physiol.*, 64, pp. 355–405.