UNIVERSITY OF SOUTHAMPTON

Multiprocessing Neural Network Simulator

by

Anton Kulakov

A thesis submitted for the
degree of Doctor of Philosophy

in the

Faculty of Engineering and Applied Science
Department of Electronics and Computer Science

January 2013

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE
DEPARTMENT OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

Multiprocessing Neural Network Simulator

by Anton Kulakov

Over the last few years tremendous progress has been made in neuroscience by employing simulation tools for investigating neural network behaviour. Many simulators have been created during last few decades, and their number and set of features continually grows due to persistent interest from groups of researchers and engineers.

A simulation software that is able to simulate a large-scale neural network has been developed and presented in this work. Based on a highly abstract *integrate-and-fire* neuron model a clock-driven *sequential* simulator has been developed in C++. The created program is able to associate the input patterns with the output patterns. The novel biologically plausible learning mechanism uses *Long Term Potentiation* and *Long Term Depression* to change the strength of the connections between the neurons based on a global binary feedback.

Later, the sequentially executed model has been extended to a multi-processor system, which executes the described learning algorithm using the event-driven technique on a parallel *distributed* framework, simulating a neural network *asynchronously*. This allows the simulation to manage larger scale neural networks being immune to processor failure and communication problems.

The multi-processor neural network simulator has been created, the main benefit of which is the possibility to simulate large scale neural networks using high-parallel distributed computing. For that reason the design of the simulator has been implemented considering an efficient weight-adjusting algorithm and an efficient way for asynchronous local communication between processors.

Contents

Acknowledgements **11**

1 Introduction **13**

1.1 Background 13
1.2 Aim and Motivation 16
1.3 Objectives 16
1.4 Contributions 17
1.5 Thesis Structure 18

2 Biological Principles **20**

2.1 Neuron Structure 21
2.2 Action Potential 23
2.2.1 Formation 24
2.2.2 Propagation 25
2.2.3 Propagation Delay 26
2.2.4 Active Properties of Dendrites 27
2.3 Synaptic Plasticity 28
2.3.1 Long-term Plasticity 28
2.3.1.1 Long-term Potentiation 28
2.3.1.2 Long-term Depression 29
2.3.2 Short-term Plasticity 30
2.3.2.1 Facilitation of Transmitter Release 30
2.3.2.2 Post-tetanic Potentiation 30
2.3.2.3 Depression of Transmitter Release 30
2.4 Plasticity Mechanisms 31
2.4.1 Hebbian-type Learning Rules 32
2.4.2 Spike-timing-dependent Plasticity 33
2.5 Summary 36

3 Modelling and simulation **38**

3.1 Information Sources 39
3.1.1 Biological Sources 39
3.1.2 Computational Sources 40
3.2 Neuron Models 41
3.2.1 Biophysically Detailed Models 43

3.2.2 Computationally Efficient Models . . . 44
3.3 Simulation Strategies . . . 50
3.3.1 Network Structure . . . 50
3.3.2 Activity Propagation . . . 51
3.3.2.1 Feed-forward Neural Network . . . 52
3.3.2.2 Recurrent Neural Network . . . 53
3.3.2.3 Reservoir Computing . . . 54
3.3.3 Modelling Techniques . . . 55
3.3.3.1 Time-driven Technique . . . 56
3.3.3.2 Event-driven Technique . . . 57
3.3.3.3 Temporal Precision . . . 59
3.3.4 Learning Paradigms . . . 60
3.3.4.1 Supervised Learning . . . 60
3.3.4.2 Reinforcement Learning . . . 61
3.3.4.3 Unsupervised Learning . . . 61
3.4 Distributed Computing . . . 62
Need of Communication Network . . . 62
Faster Execution . . . 62
Computational Reliability . . . 63
Scalability . . . 63
Enough On-chip Memory . . . 63
3.4.1 Distributed Simulation Challenges . . . 63
3.5 Available Architectures . . . 64
3.5.1 Parallel Computers . . . 64
3.5.1.1 Single Program Multiple Data . . . 65
3.5.2 Application-Specific Integrated Circuit . . . 65
3.5.3 Field Programmable Gate Array . . . 66
3.5.4 Graphics Processing Units . . . 66
3.5.5 Summary . . . 67
3.6 Existing Simulation Environments . . . 67
3.6.1 NEURON . . . 68
3.6.2 GENESIS . . . 69
3.6.3 NEST . . . 70
3.6.4 BRIAN . . . 70
3.6.5 NCS . . . 71
3.6.6 PCSIM . . . 72
3.6.7 SPLIT . . . 72
3.6.8 HHSIM . . . 73
3.6.9 MvaSpike . . . 73
3.6.10 SpikeNET . . . 73
3.6.11 KInNeSS . . . 74
3.6.12 LENS . . . 75
3.6.13 JavaNNS . . . 75
3.6.14 NeuroJet . . . 76

3.6.15 Nengo 76
3.6.16 MOOSE 77
3.6.17 FANN 77
3.6.18 CX3D 78
3.6.19 Review Conclusions 78
3.7 Summary 79

4 Sequential Neural Network Simulation 81
4.1 Network Model 82
4.1.1 Network Architecture 82
4.1.1.1 Connectivity 82
4.1.1.2 Introducing Dilution 82
4.1.1.3 Diluted Network Importance 83
4.1.1.4 Network Structure 84
4.1.2 Neuron Model 85
4.1.2.1 Rationale for Chosen Model 85
4.1.2.2 Model Parameters 86
4.1.2.3 Model Dynamics 87
4.2 Learning 88
4.2.1 What is Learning? 88
4.2.2 Biological Inspiration 88
4.2.2.1 Morris Water Maze 88
4.2.2.2 Representation of Biological Learning 89
4.2.3 Learning Mechanism 90
4.2.3.1 Existing Techniques 90
4.2.3.2 Hebbian Postulate 91
4.2.3.3 Rosenblatt Principle 91
4.2.3.4 Weight Update Rule 92
4.3 Program Implementation 93
4.3.1 Initialisation Phase 93
4.3.2 Algorithm 95
4.4 Model Weaknesses 97
4.4.1 Active Paths Interference 97
4.4.2 Reasons of Active Paths Interference 98
4.5 Simulation Results and Discussions 99
4.6 Biologically-inspired Agent 103
4.6.1 Environment 104
4.6.2 Agent and Food 104
4.6.3 Food Searching Algorithm 106
4.6.4 Decision Making Mechanism 108
4.6.5 Agent Simulation Results 108
4.7 Summary 111

5 Parallel Distributed Simulation 113

5.1 Objectives . . . 114
5.2 Mapping . . . 114
5.2.1 Neuron Number Based Mapping . . . 114
5.2.2 Delay Based Mapping . . . 114
5.2.3 Average Activity Based Mapping . . . 115
5.2.4 Neuron Connection Based Mapping . . . 115
5.2.5 Conclusion . . . 115
5.3 Distributed Time Management . . . 116
5.3.1 Synchronisation . . . 116
5.3.1.1 Synchronous Simulation . . . 117
5.3.1.2 Asynchronous Simulation . . . 118
5.3.2 Deadlock . . . 119
5.3.2.1 Conservative Synchronization . . . 120
5.3.2.2 Optimistic Synchronization . . . 121
5.4 Simulator Structure . . . 122
5.4.1 Communication Mode . . . 123
5.4.2 Spike Message . . . 124
5.4.3 Message Passing Interface . . . 125
5.4.4 Master Processor . . . 125
5.4.4.1 Interaction with Environment . . . 126
5.4.4.2 Initialization Stage . . . 126
5.4.4.3 Simulation Flow . . . 127
5.4.4.4 Master-Slave Interaction . . . 127
5.4.4.5 Slave Termination . . . 128
5.4.5 Slave's Workload . . . 129
5.4.5.1 Applying Learning Rules . . . 130
5.5 Results and Discussions . . . 130
5.5.1 Spiking Pattern . . . 130
5.5.2 Weight Alteration . . . 130
5.5.3 Performance Estimation . . . 136
5.5.4 Fault-tolerance . . . 138
5.6 Summary . . . 141

6 Conclusions and future work 142
6.1 Thesis Summary . . . 142
6.2 Future Work . . . 144

List of Figures

2.1 The schematic diagram of the neuron [1]. 22

2.2 Schematic and real view of an action potential [2]. 24

2.3 The propagation of an action potential [3]. a) The depolarised region on the far left causes sodium channels to open, further depolarising the region. b) At certain point, the sodium channels become inactive and potassium channels open, which temporary depolarise the membrane. c) The process repeats as the wave of depolarisation propagates down the axon. 25

2.4 LTP and LTD dependence on the difference between the spike arrival and new spike generation times [4]. *SCR* (synaptic change rate) describes the change of the absolute synaptic efficacy. The parameter A represents absolute synaptic efficacy, T_i describes the size of the learning window, SAT shows spike arrival time from pre-synaptic neuron and FT shows firing time of post-synaptic neuron. The dashed line demonstrates the change of the parameter A based on the variation of pre- and post-synaptic firing time lag. 29

2.5 Variation in amplitude of excitatory postsynaptic current ($EPSC$) as a function of action potential arrival times. $\Delta t = t_{post} - t_{pre}$, the sending times of postsynaptic and pre-synaptic neurons [5]. 34

3.1 Two interconnected cortical pyramidal neurons and in vitro recorded spike. 39

3.2 Simplified neuron model. If the sum of the inputs weighted by w_{ji} exceeds the threshold θ, then the output y_j of the neuron is different from 0. 41

3.3 Action potential generation in the Hodgkin-Huxley model. 43

3.4 Neuronal responses obtained for different values of the four model parameters a, b, c, and d (see Fig. 3.5). Each is labelled with the corresponding biological behavior. [6]. 47

3.5 Choice of a, b, c, and d parameters for representation of regular spiking (RS), intrinsically bursting (IB), chattering (CH), fast spiking (FS), thalamo-cortical (TC), resonator (RZ), and low-threshold spiking (LTS). [7]. 48

3.6 Comparison of the architecture of a feed-forward (left hand side) and a recurrent neural network (right hand side); the grey arrows sketch the direction of computation. 52

3.7 Various types of connections in a recurrent networks. 53

3.8 Reservoir computing. 54
3.9 Approximating the value of the membrane potential in time-driven simulation with every time step Δt. 56
3.10 Change of state in time-driven (left) and event-driven (right) simulation. 57
3.11 **A.** Comparison between time-driven (cd, top: low temporal resolution; middle: higher resolution) and event-driven (ed) technique. **B.** Impact of simulation strategy on facilitation and depression of synapses. **C.** Differences in the synchronous event occurrence for various temporal resolutions. **D.** Small differences in spike times can accumulate and lead to severe delays (top, arrow) or even cancellation (bottom, arrow) of spikes (modified from [8]). 59

4.1 The effect of dilution: fully connected network (*left*) and with some missing connections (*right*). 83
4.2 Example of different subsets of neurons in the network. 85
4.3 Neuron model parameters. 86
4.4 Influence of excitatory (EPSP) and inhibitory (IPSP) synapses on the membrane potential, resulting in a spike (an action potential) if the membrane potential crosses the threshold level [9]. 87
4.5 Morris water maze. 89
4.6 Simulation initialization stage. 94
4.7 Simplified representation of main program classes. 95
4.8 Path interference of A-C-D and A-B-D neuron paths [10]. 98
4.9 The performance of the neural network of 200 neurons while increasing the number of patterns to be learned. 100
4.10 The capacity of the network to learn as the number of patterns increases until the system is no longer capable of learning. The solid lines represent the network of 2000 neurons and the dashed lines represent the network of 200 neurons. 100
4.11 Dependence of the number of learning steps required to learn 20 neurons by 500-neuron size network on the faults number injected at the network level during the learning process. The level of faults is presented in percents. 101
4.12 The number of learning steps requried to learn 20 different input patterns depending on the level of connectivity in the network of 500 neurons. 102
4.13 Dependence of the number of learning steps on faulty components (the red line for faulty nodes and the green one for faulty synapses) in the network of 500 neurons. 102
4.14 Environment view. 104
4.15 The agent design framework. 105
4.16 The food search algorithm. 107
4.17 Food chasing success rate versus the performed learning steps. . . . 109
4.18 Dependence of the neural network performance on the learned patterns number. 110

4.19 Comparison of the performance with and without un-learning technique as a function of the learned patterns number. 110
4.20 The influence of the corrupted level of vision on the quality of produced output. 111

5.1 Distributed time management techniques. 117
5.2 An example of synchronous simulation with a controlling processor not participating (a) and participating (b) in simulation. Thick line (blue) represents processing state, fine line (green) – idle, arrows (in red) – communication. 118
5.3 An example of asynchronous simulation. Thick line (blue) represents processing state, fine line (green) – idle, arrows (in red) – communication. 119
5.4 Secure window in conservatively synchronised simulation. 121
5.5 Sender-oriented and target-oriented communication mode. 123
5.6 A sample spike train obtained simulating a network of 40 neurons. . 131
5.7 Total activity obtained simulating a network of 40 neurons. 131
5.8 Training efficiency dependence on the number of equilibration steps applied prior training. During the training phase 4 different patterns were learned. The network size was 2000 neurons with 5 input and 5 output neurons and 90 % connectivity. 132
5.9 Weight distribution of input-hidden connections at the initial phase (a), after the equilibration phase (b), and after the learning phase (c). 134
5.10 Weight distribution of hidden-output connections at the initial phase (a), after the equilibration phase (b), and after the learning phase (c). 134
5.11 Weight alteration of connections between input and hidden layers during the simulation progress. 135
5.12 Weight alteration of connections between hidden and output layers during the simulation progress. 135
5.13 The time required for simulation of a network with $1 \cdot 10^4$ neurons. 137
5.14 The mean value of rate of change in the synaptic weights of sequential (dashed red line) and parallel (solid blue line) simulation. . . . 137
5.15 The quality of output against the amount of faulty neural network's nodes while recalling pre-learned 20 patterns using the network of 2000 neurons. 140
5.16 The quality of output against the amount of faulty neural network's nodes while recalling pre-learned 20 patterns using the network of 2000 neurons (zoomed version of Fig. 5.15). 140

List of Tables

2.1 Summary of experimental evidence of axonal conduction delays in different neurons and species. 27

3.1 Comparison of the computational properties of neuron models, after [6]. Sign “+” means ability to implement and “-” means inability to implement the given feature by certain model. 49

5.1 The weight matrices of synaptic weights after learning process. . . . 138

Declaration of Authorship

I, Anton Kulakov, declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

Multiprocessing Neural Network Simulator

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. None of this work has been published before submission.

Signed:

Date:

Acknowledgements

I would like to thank all those who helped me throughout my research. I am particularly thankful to my supervisor, Prof. Mark Zwolinski, for giving me the opportunity to undertake this research assignment, and, most importantly, for his invaluable advice, continuous encouragement, and patient guidance.

I would also like to thank my wife Olga for her invaluable support, thoughtful care, and endless love. To her and our son Anton I dedicate this work.

"Our brains keep working despite frequent failures of their component neurons, and this 'fault-tolerant' characteristic is of great interest to engineers who wish to make computers more reliable."

Stephen B. Furber

Chapter 1

Introduction

The brain's structure and its organisation have been attracting attention throughout all history. The brain has a very complex and intricate structure. For that reason its study has always been an incredibly difficult and challenging task.

1.1 Background

Over the past century tremendous progress has been made in neuroscience, when technological advances made it possible to move beyond description to explorations of brain systems. The achievements have been made by employing classical and optical electrophysiological techniques (e.g. sharp electrode and planar patch clamp techniques or bioelectric recognition assay method). It was discovered that the basic biological control component is an electrically excitable cell called the *neuron* (also known as a *nerve cell*). Neurons communicate between each other by sending electrochemical messages (i.e. using chemicals to produce an electrical signal) via special connections with other cells called *synapses.* The fundamental process that triggers synaptic activity is the *action potential.* It is an electrical discharge (or *spike*), occurring in the cell's membrane potential. Spikes travel along the membrane down the axon and through synapses to other neurons, rapidly carrying information.

In spite of the recent achievements made due to the astonishing level of current experimental and measurement techniques (i.e. Hodgkin-Huxley mathematical model on transmission of electrical signals in neurons, [11] and Kandel *et al.* work on a biochemical analysis of changes in neurons associated with learning and memory storage, [12]), investigators are unable to discover and comprehend the brain's working details. And there is not any widely accepted theory about the brain's functionality, explaining the way brains reorganise and reinforce themselves in response to new stimuli and learning experience. Using conventional laboratory techniques (physical intervention of intracellular electrodes or influence of fluorescent dyes) alters the activity of the neurons and often produces inaccurate results. Difficulties, associated with obtaining experimental data, limit the progress of understanding the principles and mechanisms of brain activity. In this situation neural network modelling becomes more valuable by providing an alternative way to make the experiments via computer simulation.

Motivated by outstanding biological achievements, engineers and scientists have made various attempts to mimic the structure of the brain. A steadily growing number of simulation environments endow computational neuroscience with tools, allowing simulation of neural systems with increasing complexity and employing various neuron models [13].

With the vast improvement in computational capacity, the chances to implement the complex system of neural network noticeably increase. Computational power requirements, necessary to perform real-time simulation, can be met by employing a massively parallel architecture. The communication load, caused by the communication between the neurons, could be partly reduced by employing various sophisticated algorithms [14]. The simulation of the mechanisms implicated in information processing at the network level already brings important achievements validating or disproving the various neuroscientific hypotheses [15, 16]. Unfortunately, due to insufficient knowledge about the processes occurring inside the neuron, these simulations face many challenging problems, especially when the simulator models are large, containing thousands of neurons and millions of synapses.

One of the most important obstacles is the absence of a neuron model that would precisely describe neuronal activity on ion channel level and at the same time would be computationally efficient for large-scale simulation and amenable to mathematical analysis. There are hundreds of models in existence, but none of them fully meets these requirements [17]. On the one side, detailed biophysical models fully mimic the neuron by reproducing ion channels activity on the tree-like spatial structure of the neuron cell (Hodgkin-Huxley model, [11]). Although they provide very effective and biologically plausible neural networks (being able to predict changes in the chemical environment or the influence of temperature dependence), they are computationally demanding and thus expensive to implement. On the other side, the abstract mathematical models provide only a weak link to the underlying biophysical causes of electrical activity by neglecting neuronal morphology and reducing the neuron to an extensionless mathematical construct (Izhikevich model, [7]). They can be easily implemented, they are able to simulate a large network and do not require expensive simulation equipment.

Several software tools have been developed which are able to simulate limited aggregates of the neurons. The main aspects to take into account are:

1. the speed of simulation;
2. the memory requirements;
3. the accuracy of simulation.

These aspects are usually in conflict, such as the speed of simulation is inversely proportional to the simulation's accuracy and memory consumption. But a trade-off can be found depending on the goal of simulation that indicates the flexibility of the program.

For the research purposes there was a choice of either developing specific tools or using an already adapted simulation environment following a designated procedure. There were a number of possible off-the-shelf simulators available to use for our experiments (e.g. "Neuron", [18] or "Genesis", [19]). However, they had been designed to be all-purpose simulators (allowing different simulation modes and many other options). They were, therefore, large programs with very complicated inner-working. Unfortunately, although intended as all-purpose feature-rich simulation tools, none of them had all the built-in facilities that could serve all our purposes. Modifications would have been needed to allow all the different operations in this study (such as reinforcement learning technique combined with Hebbian learning, all damage experiments and analysis procedures to be described later). The way the adapted simulation environments are structured does not allow modifying them straightforwardly. Readjusting such large pieces of code would have been more time-consuming and error-prone process than writing a new tool. It was decided at an early stage that it would be easier overall to make the necessary day-to-day changes to a smaller, custom-written program that contained only the facilities necessary for our experiments.

Also, eventually there was an intention to launch the created simulator on the *SpiNNaker* massively-parallel neuromorphic computing platform, which is capable of simulating millions of neurons in real time. The distinctive feature of *SpiNNaker* is based on distributed communication-centric computation and non-deterministic communication, which provides the potential to investigate new principles of massively parallel computation.

1.2 Aim and Motivation

The aim of this thesis is to develop a neural network simulator. The *motivation* for doing so is two-fold. Firstly, we hope that such models may contribute to our understanding of mechanisms by which real brains operate. Secondly, the simulator is designed to exploit the potential of the *SpiNNaker* neuromorphic distributed computing machine, gaining insights into the nature of neural computation itself [20].

1.3 Objectives

Bearing both of these aims in mind, the research is directed towards achieving the following objectives:

- **Biological plausibility:** the approach we take in this thesis is not to limit the models under study to those that precisely resemble their biological counterparts, but nonetheless to use models which are *biologically inspired* both in architecture and learning mechanism. We belief that focusing on building models consistent with every known fact about the brain hinders finding general principles of biological computation processes.

  Therefore, to achieve a biologically realistic simulation of a neural network, our approach is to employ an abstract neuron model, enriched with physiologically motivated functionality (e.g. adaptable synaptic weights, fixed threshold potentials) and to create a neural network simulator with a flexible, reusable, event-driven, platform-independent and scalable framework, based on such a neuron model.

  We want to build a model that is able to search for a correct output by itself without a feedback indicating an approximated error. The model searches for the optimal set of weights and makes rewarding associations more probable based on the binary feedback indicating the correctness of the current solution. Our hope is that we do not have to use any additional artificial mechanisms, which were not found in biological networks. For the same reason we avoid using biologically implausible learning methods, where the information about the states and the properties of the involved neurons, which directly affects their synaptic alteration, is spread around the network (e.g. error backpropagation algorithm). (Chapter 4)

- **Multipurpose:** it is not our aim to create a simulator that is optimised for some particular task. Our purpose is to simulate the biologically inspired network of neurons and to investigate the way it "learns". The intention is to study a network that is able to re-arrange its connection for a correct output because it is biologically plausible. (Chapter 4)
- **Fault-tolerance:** the SpiNNaker platform has been constructed as a dynamically reconfigurable and fault-tolerant computing system. However, the system depends on the software to make full use of these features. Therefore in case of a node failure, the simulator should be able to dynamically redistribute the working task in this way isolating the failed node from further use. This emulates a biological neural system, where neurons die frequently and spontaneously (one per second in adult human) without significantly affecting the overall brain performance. (Chapter 5)
- **Scalability:** SpiNNaker is a scalable computing system, so that linking the SpiNNaker multi-processor chips together it is possible to assemble a system of almost any desired scale. Correspondingly, the simulator should satisfy the same criteria being able to handle growing amount of network size in a capable manner by employing new resources in the case of necessity. For such a large systems it is necessary to apply an efficient weight-adjusting algorithm and corresponding computational-communication balance (due to increased communication increasing computational power reduces the speedup gains) for parallel distributed execution. (Chapter 5)

A simulator with these features has been designed during the project. The program is implemented in C++. Issues of computation efficiency and memory requirements were carefully considered. The software architecture and algorithm have been presented. The implementation details have been introduced and explained.

1.4 Contributions

The research has contributed in providing:

- A novel parallel distributed event-driven simulation model of neural network has been developed.
- A fault-tolerant mechanism immune to processor failure and communication problems during a learning phase on a parallel distributed hardware has been created.

1.5 Thesis Structure

The thesis is composed of six chapters. The **first chapter** presents an introduction along with the motivation and objectives for the project. It includes an explanation of the general background of the research, statement of the research aim and objectives, in addition to the importance of the study.

The **second chapter** aims to introduce brief fundamentals of the synaptic plasticity along with the biological characteristics of the neuron, which are involved in an action potential formation and propagation. It presents the recent biological knowledge of neural network plasticity to give an idea about the field of research.

The **third chapter** gives a comprehensive literature review of existing neuron models together with an overview of simulation strategies and evaluation of the existing neural network simulators. It also discusses important obstacles that designer encounters while implementing large scale neural network.

The purpose of the second and third chapters is to develop a good understanding of the relevant previous research work and investigation that took place in the similar areas of interest.

The **fourth chapter** introduces and explains an approach for building and executing a biological plausible neural network on a single processor, capable of simulating large-scale networks. It is capable of searching for correct patterns, storing and reproducing them later. The algorithm and architecture of the simulation software are illustrated. The model is functionally verified and experimental results are included. It is tested by an agent application, which represents a learning system. The agent is able to accumulate the vital experience from a dynamic ambient environment based on its own observations in an unsupervised manner. An analysis of the performance as well as simulation results of the application are presented and discussed at the end of the chapter.

The **fifth chapter** gives introduction to the field of *distributed* computation and discusses commonly addressed issues along with distributed simulation techniques. It also discusses the benefits and challenges of using the distributed hardware for neural network simulation. It extends the sequentially executed model presented in Chapter 4 to a concurrently executed model with calculations distributed over a multi-processor machine. This makes the program more scalable and tolerant to hardware failures. The algorithm and architecture of the simulation software are illustrated. Discussion of challenging problems encountered while implementing the system and the undertaken solutions are presented.

The **sixth chapter** is the last chapter of the thesis. It concludes the thesis with a succinct summarization, discusses the results and produces conclusions of the research. It also contains several possible directions for the future work of research as well as the changes to the software that could be made in order to enrich and improve the program's performance.

Chapter 2

Biological Principles

The brain structure and its organisation have been attracting attention through all history. Because of the brain's incredible connectivity and the microscopic scale of its interconnection, there is lack consistent theories about neural coding and computation in modern neuroscience. The investigation of neuron structure is an incredibly difficult and complex task that yields relatively low rewards in terms of information from biological forms. The structure and connectivity of even the simplest invertebrates are almost impossible to establish with standard laboratory techniques. That is why at present knowledge about the operational principles of the brain is far from complete, so simulation attempts must employ a great deal of assumption and guesswork to fill the gaps in the experimental evidence.

Understanding of the principles and mechanisms of brain activity could benefit the human development. First of all, imitation could allow expanding of computing capabilities. Also, understanding such a phenomenon as memory capacity, an animal's planning or reasoning, thought or consciousness (particularly mammals) would benefit the progress of the mankind. Finally, the hope is that by emulating the brain, it will be possible to capture some of its computing capabilities.

The main feature of the brain is the ability to learn. We understand *learning* as the probability for certain behaviour to happen in response to a certain event. The basis for most models of learning is a synaptic plasticity. Plasticity refers to the changes that occur in the organisation of the brain as a result of experience. Several underlying mechanisms cooperate to achieve plasticity, forming cognition and memory formation.

This chapter aims to introduce the biological structure of a neuron and its characteristics (Section 2.1) that are involved in an action potential formation and propagation (Section 2.2) along with the fundamentals of plasticity (Section 2.3).

Section 2.4 describes the contemporary learning rules based on a synaptical plasticity.

2.1 Neuron Structure

In order to imitate biological neural networks it is important to understand the nature of biological neurons, which are the building blocks of neural networks. This section introduces biological neurons and explains how they work.

The investigation of the neuron behaviour is an incredibly difficult and complex task. It is almost impossible to establish the brain's structure and connectivity of even an elementary animal using conventional laboratory techniques. Recent research employed alternative ways of investigations using electro-physiological techniques and computer simulation experiments [21]. They unveiled the internal structure of the brain and allowed the scientists to mimic its behaviour.

Although there are all kinds of different neurons, the basic structure is the same. As an ordinary cell, a neuron typically consists of a *soma* (or *cell body*), a *dendric tree* and an *axon.* The majority of the neurons in vertebrate organisms input the signal through the synapses on the dendrites, transmit it along the cell body outputting via the synapses located on the axon. However, there is some heterogeneity throughout the nervous system of different species in the size, shape and function of neurons.

The key parts of the neuron are shown in Fig. 2.1 and their functionalities are described below.

- The central part of the neuron is the **soma**. It houses the normal metabolic systems required to maintain the cell, such as nucleus, mitochondria and other organelles. All internal organelles are surrounded by a cell membrane and suspended in intracellular fluid, known as cytoplasm.

- The cellular extensions with many treelike branches are referred to as a **dendritic tree**. The dendrites' role is to receive signals from other neurons through the synapses. The backflow of a neuron is inhibited. Such an inhibition happens first of all because the axon does not contain any chemoreceptors and, secondly, because the dendrites are unable to secrete the neurotransmitter chemicals. This unidirectionality of a chemical synapse explains conduction of impulses only in one direction.

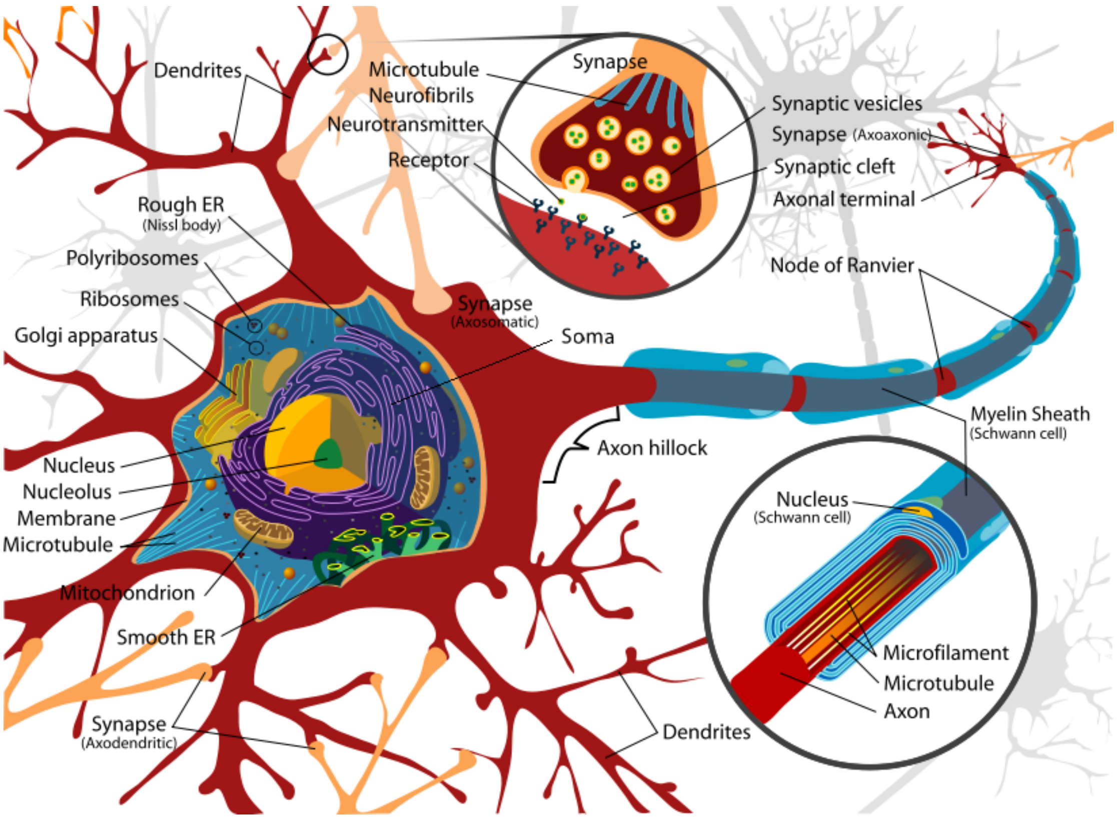


FIGURE 2.1: The schematic diagram of the neuron [1].

- The **axon hillock** connects the cell body to the axon. It contains the greatest density of voltage-dependent sodium channels. This makes it the most easily-excited part of the neuron and the spike initiation area for the axon.

- The **axon** represents a finer, slender, cable-like extension, the length of which can be tens, hundreds, thousands, or even tens of thousands times the soma's diameter. The axon edge has its own specialised structure for passing information across the synapse to other cells in the nervous system.

- Some axons contain a fatty sheath around the axon called **myelin**, which provides electrical insulation for the covered sections of the axon membrane from the extracellular fluid with the purpose of accelerating the propagation of an action potential along the axon. The areas between the consecutive myelin sections are the *nodes of Ranvier*, which cause regeneration of electrical signals.

- A **synapse** is a neuron's protrusion, which acts as contact points where two neurons can communicate with each other. It consists of two elements, separated by a synaptic cleft: the presynaptic terminal and the postsynaptic receptor site, which may be located on the axon, soma or dendrite of the neuron cell. The synapse is one of the most important controlling factors within the central neural system due to its role in controlling the signalling process. Depending on whether a synapse raises or lowers the membrane potential of its neuron, one differs between an *excitatory synapse* and an *inhibitory synapse* respectively.

- The **axon terminal** contains synapses where neurotransmitter chemicals are released in order to communicate with the target neurons. The impulse charge, generated at the axon hillock, arrives at the chemical synapses of the axon terminals. There the ionic charge is converted back into a chemical signal and transmitted further to the postsynaptic neuron. At the same time the axon terminals take up neuron growth factors (neurotrophin), released by the postsynaptic dendrites, which are transported back to the nucleus.

- The axon terminal stores **neurotransmitters**. Neurotransmitter is a chemical that is used to relay, amplify and modulate a signal between the neuron and another cell. It uses the energy propagated by the action potential to fuse with the membrane of the cell and diffuses across the synaptic cleft.

2.2 Action Potential

A neuron is responsible for propagating electrical signals through the neural system. Neurons communicate by the chemical and electrical synapses in a process known as a synaptic transmission. After receiving signals from the presynaptic neurons, the cell makes "decision". Based on the activity at its input, the neuron either does or does not generate a spike. On average, the action potentials of individual neuron are rare and occur at rates of 1-50 Hz. At the same time the rate of incoming action potentials is much higher and is approaching 100 kHz due to some 10^4 incoming connections [22]. The synaptic transmission occurs when an *action potential* is initialised in the presynaptic neuron and is characterized by a sudden change in membrane potential of the postsynaptic neuron.

2.2.1 Formation

The action potential (or AP, also known as a *wave of depolarisation* or a *spike*) is an electrical impulse travelling along several types of the cell membranes (Fig. 2.2). The process starts when a neuron is excited by more than one synapse in a short time, or when the same synapse is repeatedly active. The cell's interior voltage rises comparing to the cell's exterior voltage. Voltage-sensitive *sodium* channels expand and allow sodium current entering into the axon, further depolarising the membrane. When the voltage reaches its threshold level, the neuron fires. This process initiates a positive feedback loop, raising the voltage further up. However, after the spike releases, the membrane voltage is restored to its resting value (between -40mV and -90mV) because the channels, responsible for the initial inward current, are deactivating. Meanwhile the raised voltage opens *potassium* voltage-sensitive channels that pass through the outward current.

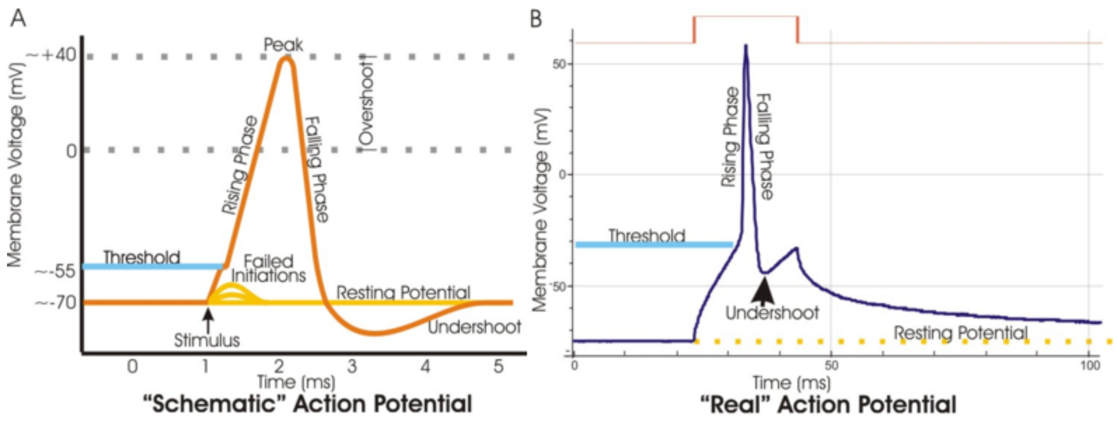


FIGURE 2.2: Schematic and real view of an action potential [2].

The passage of an action potential can leave the ion channels in non-equilibrium state, making them more difficult to open, and thus inhibiting another action potential at the same spot (resting potential in Fig. 2.2). Such a state is said to be *refractory*. The refractory period can be divided into two phases. In the first phase with a duration of about 1 ms to 5 ms, called the *absolute refractory period*, it is not possible for the neuron to fire another spike (i.e. the threshold is said to be infinite). In the second phase, called the *relative refractory period* with a length of 2 ms to 20 ms, the threshold slowly returns to its normal level. During this phase a spike could be emitted, but a stronger depolarisation of the membrane potential is required.

2.2.2 Propagation

The triggered action potential propagates through the axon without fading out because the signal is regenerated at each patch of the membrane. Due to the myelin sheath the action potential travels further before being regenerated at the areas between the consecutive myelin sections known as the *nodes of Ranvier*, Fig. 2.1. This accelerates the action potential propagation along the axon, since it only needs to be regenerated at the uncovered sections rather than continuously along the length of the axon. An action potential at one patch raises the voltage at nearby patches of the axon, depolarising them and provoking a new action potential there. The diagram in Fig. 2.3 shows a section of an axon, which is conducting an action potential. The action potential propagates through the axon and causes a back-propagation low amplitude pulse in the dendrites [23].

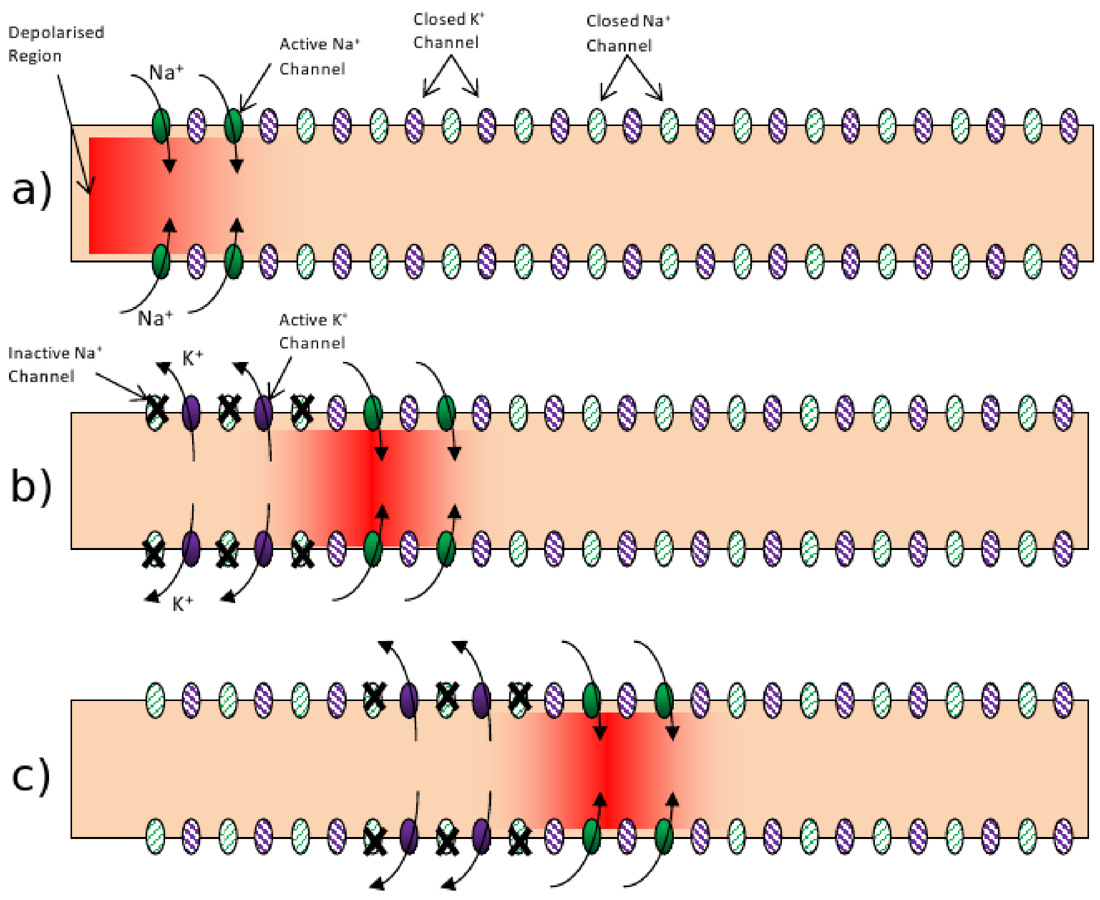


FIGURE 2.3: The propagation of an action potential [3]. a) The depolarised region on the far left causes sodium channels to open, further depolarising the region. b) At certain point, the sodium channels become inactive and potassium channels open, which temporary depolarise the membrane. c) The process repeats as the wave of depolarisation propagates down the axon.

The action potential stops at the end of the axon and causes the secretion of the neurotransmitter at the synapses that are found there, Fig. 2.1. Released neurotransmitter binds to the receptors on adjacent cells, which are ion channels themselves. In contrast to the axonal channels, they are generally opened by the presence of the neurotransmitter, rather than by changes in the voltage. The opening of these ion channels can cause polarisation or depolarisation of the postsynaptic neuron based on the type of neurotransmitters at the synapse. An excitatory neurotransmitter generates a postsynaptic potential depolarisation, while a depolarising inhibitory neurotransmitter causes postsynaptic potential hyperpolarisation. The generated potential depends on the amount of neurotransmitters released into the synaptic cleft and the number of accepting receptors. Sufficiently strong depolarisation can provoke another action potential in the new cell.

2.2.3 Propagation Delay

An action potential does not propagate between neurons instantaneously. First of all, the delay is caused due to the cable properties of a neuron. In transmission along the axon and dendrites a delay may vary from a half of a millisecond in a short axon to tens of milliseconds in a very long one. A myelin sheath affects the speed of action potential propagation, significantly increasing it (see Section 2.1). Myelinated axons demonstrate the spike transmission velocity of around 1 m/s, while in the non-myelinated axon it is only around 0.15 m/s. [24].

Another cause for transmission delay is a synapse. The time is required for a neurotransmitter to be released by presynaptic terminal, diffuse across the synaptic cleft (approximately 0.05 ms), and bind to a receptor site on the post-synaptic ending (about 0.15 ms). After having reached the dendritic tree, the synaptic input propagates to the soma. Depending on the physical location of a synapse within the tree, it can take up to 10 ms [25]. However, the typical delay time varies from 0.3 to 4.0 ms between the onset of the action potential on a presynaptic neuron and action potential at the post-synaptic site. Delays could also be instantiated by “chained” connections through other neurons.

Propagation delays demonstrate high variability in biological neural systems. This also applies to connections that run in parallel. Different propagation delays can be observed in two connections, which both originate and terminate in similar brain areas and follow alike paths. Conversely, some connection types can produce similar transmission delays irrespective of the length of a connection (e.g. thalamo-cortical). Nevertheless, the delay in any given connection is consistently reproducible with sub-millisecond precision despite the high variability between connections in many brain areas. Some experimental data is collected in Table 2.1.

TABLE 2.1: Summary of experimental evidence of axonal conduction delays in different neurons and species.

Connection	Delays (ms)	Animal	Reference
layer 6 $\longrightarrow$ LGN	1 - 44	cat	[26]
	1.7 - 32	rabbit	[27]
cortico-cortical	1 - 35	rabbit	[28]
	1.2 - 19	rabbit	[27]
cortico-(ipsi)cortical	2.2 - 32.5	rabbit	[27]
layer 5 $\longrightarrow$ LGN	0.6 - 2.3	rabbit	[27]
cortico-collicular	0 - 3	cat	[26]
VB $\longrightarrow$ layer 4	2	mice	[29]

Apparently, biological neural networks exert purposeful control over the duration of delays. This indicates the computational significance of a propagation delay. But the delay time is not consistent and can vary over time depending on different factors. For instance, the increasing activity of a neuron population can shorten the delay time. This leads to the increasing susceptibility of the network area due to the faster reaction of post-synaptic neurons to stimuli. In this way, delay variations improve short-term memory and, thus, the overall learning process, which occurs in this part of the neuron network [30].

2.2.4 Active Properties of Dendrites

Not only the synapses play the primary role in propagation of the signal and processing of information. Due to a small amplitude of synaptic potentials, summation of multiple synaptic inputs is required to reach action potential firing threshold. Dendrites are responsible for synaptic summation and for the changes in synaptic strength that take place as a function of the activity of the neuron. Dendritic voltage-gated ion channels can alter the local input resistance and time constant, which in turn would influence both spatial and temporal summation of excitatory and inhibitory postsynaptic potentials. Multiple excitatory postsynaptic potentials occurring on the same branch and within a narrow time window might activate voltage-gated channels and produce a much bigger response than would occur if they were on separate branches or occurred outside this time window.

2.3 Synaptic Plasticity

The concept of plasticity is very broad and can involve many levels of organisation. Any changes in neural system activity can be attributed to some sort of plasticity. We refer to *plasticity* as the ability of the synapse between two neurons to change in strength. Therefore, we name this phenomenon as a *synaptic plasticity.*

The synaptic plasticity is not an unitary process but rather many processes on the different time scales. There are several underlying mechanisms that cooperate to form the synaptic plasticity operating at various time scales:

- within several hours (or days), known as *Long Term Potentiation* and *Long Term Depression*, inducing long-persisting changes.
- within seconds, known as *Short Term Potentiation* and *Short Term Depression* for retaining a small amount of information in an active state for a short period of time.

It seems likely that new, perhaps unanticipated, forms of plasticity remain to be discovered. The most important, known today, are reviewed below.

2.3.1 Long-term Plasticity

Long-term plasticity represents the ability of chemical synapses to change their strength, which preserves for days, months, or years.

2.3.1.1 Long-term Potentiation

Synaptic strength increments when a neuron repeatedly fires and depolarises the postsynaptic neuron, producing a process called **L**ong-**T**erm **P**otentiation (LTP). It causes the tendency of the postsynaptic neuron to be active simultaneously with the pre-synaptic neuron. Subsequent stimuli applied to one cell are more likely to elicit action potential in the cell to which it is connected. LTP improves the receptors' sensitivity to the neurotransmitter concentration in the synaptic cleft [31]. It is achieved in large part because of increasing activity of existing receptors and by increasing the number of receptors on the postsynaptic cell surface as well as the dendrite's growth into more branches, causing the presynaptic axon to form more synapses.

2.3.1.2 Long-term Depression

If synapses simply continued to increase in strength as a result of LTP, eventually they probably would reach some maximum efficacy level. After this, the new information encoding would be difficult, if not impossible. To make the synaptic strengthening useful, another process must selectively weaken specific sets of the synapses. **L**ong-**T**erm **D**epression (LTD) is such a process. LTD is the weakening of synapse strength lasting from hours to days. The same mechanism of calcium influx causes both LTP and LTD. Low calcium influx leads to LTD, and calcium entry above the certain threshold leads to LTP [32]. The threshold rises after the synapse has already been subjected to LTP. This provides a *negative feedback* system to maintain synaptic plasticity. The threshold level is on a sliding scale and depends on the history of the synapse. According to [33], timing is another factor affecting LTP and LTD mechanisms. When the presynaptic neuron fires just before the postsynaptic neuron fires, the connection will be strengthened. However, when it fires slightly after the postsynaptic neuron, the connection will be weakened.

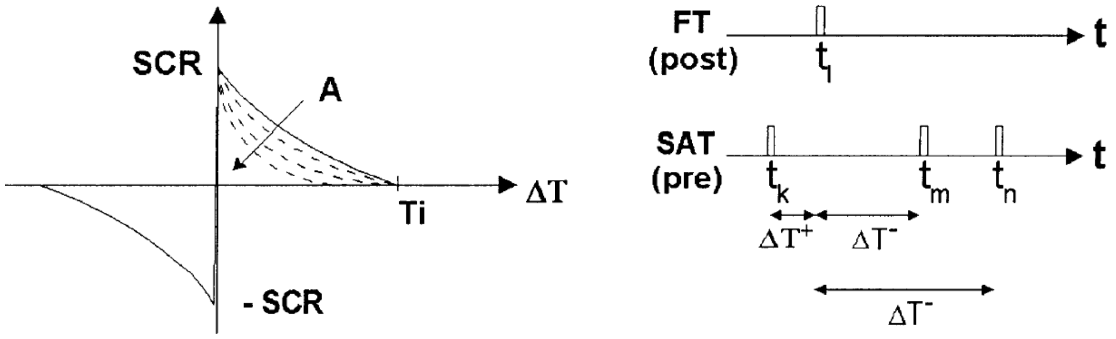


Figure 2.4: LTP and LTD dependence on the difference between the spike arrival and new spike generation times [4]. *SCR* (synaptic change rate) describes the change of the absolute synaptic efficacy. The parameter A represents absolute synaptic efficacy, T_i describes the size of the learning window, *SAT* shows spike arrival time from pre-synaptic neuron and *FT* shows firing time of post-synaptic neuron. The dashed line demonstrates the change of the parameter A based on the variation of pre- and post-synaptic firing time lag.

The dependency of LTP and LTD on time is presented in Fig. 2.4. Absolute synaptic efficacy depends on the time difference between the last firing time of the postsynaptic neuron and the spike arrival time from the presynaptic neuron. In this way the synaptic strength of a strong synapse is altered only insignificantly as opposed to a weak one. This mechanism prevents the fast saturation of the synaptic strength and stabilizes the network activity.

2.3.2 Short-term Plasticity

Paired-pulse facilitation, *paired-pulse depression* and *post-tetanic potentiation* are often referred as a short-term plasticity because of the duration of phenomenal existence. They vary strongly according to the interval between the conditioning and induced pulses. These mechanisms cannot provide the basis for memories that persist even for several hours. Instead, they uphold a small amount of information in an active state for a short period of time. To retain the information for longer, the content of information must be periodically rehearsed.

2.3.2.1 Facilitation of Transmitter Release

Synaptic enhancement that is prominent on the hundreds of milliseconds time scale is referred to as **P**aired-**P**ulse **F**acilitation (PPF), as was explained in [34]. Facilitation occurs solely at the pre-synaptic side of a synapse, where synaptic vesicles containing neurotransmitter are released from a pool. It is the result of an increase in the probability of the transmitter release. PPF can be seen with the pairs of stimuli, in which the second postsynaptic pulse, following the first one, can be up to five times the size of the first.

2.3.2.2 Post-tetanic Potentiation

When action potentials arrive close together in time, influxed calcium builds up within the terminal. This calcium causes release of more neurotransmitter by a subsequent presynaptic action potential. A high-frequency burst of the presynaptic action potentials (sometimes referred as *tetanus*) causes elevation of presynaptic calcium level, giving rise to another form of synaptic enhancement called **P**ost-**T**etanic **P**otentiation (PTP). It usually continues a few minutes after the train of the stimuli ends. The difference in duration distinguishes PTP from PPF.

2.3.2.3 Depression of Transmitter Release

Often the periods of the elevated activity lead to a decline in postsynaptic signal amplitude during repeated stimulations. This process is known as **P**aired-**P**ulse **D**epression (PPD). This state continues from seconds to minutes before the synaptic strength recovers. As explained in [34], the mechanism is caused by presynaptic decrease in the release of neurotransmitters that reflects a depletion of a release-ready pool of vesicles. The phenomenon gains strength also due to release of *adenosine* from the activated presynaptic terminals, postsynaptic cells, or neighbouring cells. This modulatory substance reduces the number of vesicles. Finally, desensitization of ion receptors makes the target neuron less sensitive to neurotransmitters.

Summarizing, synapses exhibit many forms of activity that occur over a broad temporal range. At the shortest time (seconds to minutes), PPF, PTP, and PPD provide rapid but transient modifications. Longer-lasting forms of synaptic plasticity such as LTP and LTD are more enduring forms of plasticity and can yield persistent changes in synaptic strength (hours to days or longer).

2.4 Plasticity Mechanisms

For a long time scientists believed that memories were stored in the synapses, however, they never found a synaptic model of plasticity to corroborate their theory.

In 1949 the Canadian physiologist Donald Hebb suggested the basic mechanism of synaptic plasticity in his postulate, which has been an important milestone for both neurophysiology and computer science. The suggested mechanism was the first and the only plausible learning rule for artificial neuron networks. It proposes the basis of a memory formation and storage process. The postulate describes how the connection between a presynaptic neuron A and a postsynaptic neuron B should be modified. The rule states that synapses increase their transmission efficacy if the presynaptic spike arrives before the postsynaptic neuron is activated. An often used simplification is *those who fire together, wire together.*

Basic Hebbian learning rule can be described by the following relation:

$$\tau_\omega \frac{dw}{dt} = vu \tag{2.1}$$

where τ_ω is the time constant controlling the rate of weight change, v is the postsynaptic activity evoked directly by the presynaptic activity u (the product of uv represents the nature of interaction between pre- and post-synaptic spiking activities), while w is a vector defining all the synaptic weights in the network. The synaptic weight is a suitable abstract representation of "synaptic strength“. Commonly it is treated as a dimensionless variable, as its relation to biophysical variables is not specified.

Hebb's prediction was verified decades later with such an advanced technique as the extracellular micro-electrodes and the explosion of interest in the hippocampus [35]. The direct evidence of activity-dependent long-term potentiation and its contribution to behavioural conditioning has been provided in [36]. Synaptic modifications caused by correlated activity of the involved neurons also has been observed electrophysiologically in [37]. The influence, long suspected and now proven biologically gave rise to the rule of synaptic plasticity based on the arrival times of pre- and postsynaptic action potentials [38].

One further important finding is, that for this type of learning all information is available at the location of the synapse itself. A synapse is modified as a function of the activity of only the two neurons it connects. No external teacher is necessary to employ Hebbian learning: every synapse has information whether a spike arrived from the presynaptic *neuron A* or not. An action potential also traverses backwards up the dendritic tree of the firing neuron [23]. So the information whether the *neuron B* has fired is also available to all of its synapses. Thus, every synapse has all the information of pre- and postsynaptic firing available at its location. This makes Hebbian rule a likely candidate for the biological plasticity mechanism.

On a cellular level, the induction of synaptic strengthening is triggered by activation of NMDA channels (glutamate receptors), caused by the pre-synaptic cellular depolarization. Strong depolarization of the post-synaptic cell due to post-synaptic spike completely displaces the magnesium ions (Mg^{2+}). Both neurotransmitter (glutamate) penetration and removal of the Mg^{2+} block allow brief and strong Ca^{2+} influx through the NMDA channels. This leads to suddenly rising level of post-synaptic Ca^{2+} concentration, which initiate molecular process leading to synaptic enhancement.

2.4.1 Hebbian-type Learning Rules

Several shortcomings arise while applying the Hebbian rule in its present form. Firstly, Hebb did not specify in details what the term "*near enough to excite*" exactly refers to. This addresses the question of when are two neurons considered as being active together. For a long time it was mostly interpreted in terms of the firing rates of the two neurons involved. During the last decades many varieties of Hebbian-type learning rules evolved: conjunctive-type rules, that simply require pre- and postsynaptic activity at the same time or correlational-type rules, where synaptic strengthening is dependent on statistical measures like (e.g. covariance [39]). In contrast to conjunctive-type rules, where a coincident activity of a pair of neurons to be sufficient to cause synaptic weight modification, a modification in correlation-type rules takes place only in the case when neuron A *consistently* takes part in firing neuron B.

Secondly, what is a mechanism of decreasing the synaptic weight. During a correlation of pre-synaptic and post-synaptic neurons' activity, weight growth causes a higher post-synaptic potential and therefore even more weight growth. This induces an exponential weight growth and leads to destabilisation of the entire neuron network. New memories could not be stored and old ones would be obliterated. To solve the described constraints, some modifications of the standard Hebbian rule were proposed. The most known were to introduce a sliding threshold in BCM rule (**B**ienenstock, **C**ooper, and **M**unro rule [40]), to introduce a weight decay term in Oja rule [41], and in generalised Hebbian Algorithm (also known as Sanger's rule [42]), which is similar to Oja's rule, but converges to ordered principal components.

2.4.2 Spike-timing-dependent Plasticity

With recent advancements in technology became possible more precisely measure the spike timing of neurons. Evidence has been collected that the synaptic connection between two neurons is more likely to strengthen if the presynaptic neuron fires off shortly before the postsynaptic neuron. Thus, more recent types of Hebbian learning rules incorporate exact time differences between single pre- and postsynaptic spikes. It has also been possible to measure the direction and extend of synaptic changes as a function of the exact activity timing of the presynaptic and postsynaptic neurons [43].

Using dual patch clamping techniques, Markram showed that the strength of a synapse increases while repeatedly activating the pre-synaptic neuron 10 milliseconds prior the post-synaptic target neuron and decreases if post-synaptic neuron is activated first [44]. Bi and Poo [5] continued the mapping of the entire time course relating pre- and postsynaptic activity and synaptic change and observing similar results within a time window of 20 milliseconds. They stated, that repetitive postsynaptic spiking within this time window after presynaptic activation results in LTP, whereas postsynaptic spiking within this time window before the repetitive presynaptic activation leads to LTD, Fig 2.5. Additionally, the initial synaptic efficacy also plays a role: significant LTP occurs only at synapses with relatively low initial strength, in contrast to LTD where this does not seem to have any influence. Recent work of Dan has advanced study of the phenomenon using *in vivo* whole-cell recordings [45].

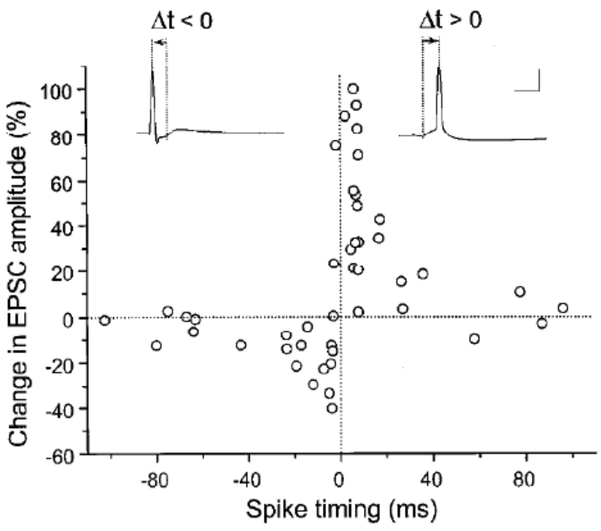


FIGURE 2.5: Variation in amplitude of excitatory postsynaptic current (*EPSC*) as a function of action potential arrival times. $\Delta t = t_{post} - t_{pre}$, the sending times of postsynaptic and pre-synaptic neurons [5].

It was established that Ca^{2+} plays a crucial signalling role both in synaptic weight potentiation and depression [46]. But it is a rise time and a decay time as well as a peak level of the Ca^{2+} influx, which determine plasticity outcomes. After pre – post spike pairing the calcium ions enter through NMDA receptors (which rapidly allow high volume of ions) and yield synaptic weight potentiation. Post – pre spike pairing leads to Ca^{2+} influx through post-synaptic voltage-gated channels (which allows a moderate volume of ions) and induces depression of synaptic weight. Biochemically, this picture of Ca^{2+}-level determinism is based on the differential activation thresholds of CaMKII kinase enzyme and phosphatase (calcineuron), both responsible for inducing the opposite synaptic modifications.

Due to such spatiotemporal dynamics, the model is called **S**pike-**T**iming-**D**ependent **P**lasticity (STDP). It enables the system to assign credit to those synapses that are actually responsible for generating the postsynaptic spike. This is implemented through setting the sign and magnitude of synaptic modification based on the precise timing of pre-synaptic and post-synaptic spikes. The rule is directly in line with the Hebbian hypothesis. Besides, it supplements the hypothesis by addressing the questions of when are two neurons considered as being active together and determining an exact amount of synaptic modification.

Different types of computational models has been proposed based on STDP, and differ by their implementation (in particular how synapse weights saturate to their minimal and maximal values). All computational models can be divided into biophysical models and computationally-efficient models.

Biophysical models follow exact biophysical and biochemical pathways and are crucial to understand the biological mechanisms underlying synaptic plasticity. They give an interpretation of internal variables in terms of the internal state of NMDA receptors and secondary messengers (e.g. [47]) as well as calcium concentration and backpropagating traces of action potentials (e.g. [48]). Due to detailed reproduction, they are able to reconstruct more complex paradigms of interaction between pairs of pre – post spikes (e.g. nonlinear interactions of spikes and frequency-dependence of STDP).

On the other hand, computationally-efficient models reproduce the dynamics of STDP without necessary referring to biological mechanism. They are amenable to mathematical analysis and used in analytical and simulation study. One of simplifications is an additive plasticity, where spike pairs are considered equally: based on the time between the spikes all the synaptic changes are calculated and summed linearly (e.g. [49]). Spike pairs contribute until an upper or lower bound is reached, leading to a bi-modal weight distribution (e.g. [50]). Although some of the models use multiplicative approach, where the magnitude of the synaptic change depends on the current synaptic weight. In this way weight contributions diminish as weight approaches its limit, providing a smoother form of saturation and a stable, unimodal distribution of synaptic weights (e.g. [51]).

Nowotny *et al.* [52] proposed a learning rule based on the experimental recordings of Bi and Poo [5]. According to [52], the weight change of a synapse $\Delta w(\Delta t)$ is defined by the equations:

$$\Delta w(\Delta t) = A_{+}\frac{\Delta t}{\tau_{+}} * e^{\frac{-\Delta t}{\tau_{+}}},\ where\ \Delta t \geq 0 \tag{2.2}$$

$$\Delta w(\Delta t) = A_{-}\frac{\Delta t}{\tau_{-}} * e^{\frac{-\Delta t}{\tau_{-}}},\ where\ \Delta t < 0 \tag{2.3}$$

where $\Delta t = t_{post} - t_{pre}$ is the difference in postsynaptic and presynaptic spike times, τ_{+} and τ_{-} determine the width of the learning windows for potentiation and depression (often chosen $\tau_{+} = \tau_{-} = 20$ ms), respectively, and the amplitudes A_{+} and A_{-} determine the magnitude of synaptic change per spike pair (often chosen $A_{+} = 0.1$ and $A_{-} = -0.12$).

The value of A_{-} often taken larger than the value of A_{+}, so that depression is stronger than the potentiation to ensure that the weights of uncorrelated pre- and post-synaptic connections go slowly to zero, while the weights of strongly correlated connections are strengthened. More about the choice of STDP parameters can be found in [50].

Obviously the only consideration of the temporality of the action potential cannot be sufficient to fully explain the phenomenon of memory, which is associated with synaptic plasticity [53]. The modulation of STDP by a third factors (e.g. dopamine) turns the learning mechanism from unsupervised learning into a reward-based learning paradigm. The synaptic strength defines the ease with which a signal traverses the synapse and is represented by a number and size of synaptic receptors and the amount of neurotransmitter, participating in a process. The amount of neurotransmitters released into the synapse varies depending on the amount of available neurotransmitters and the storage capacity of vesicles. All interacting parameters cause variations in the forms of impact and operate at various time scales.

2.5 Summary

In order to simulate a neural network, it is important to understand the dynamics of biological neurons and their collective behaviour as a population of neurons interconnected in a network.

Like many biological cells, a neuron has a cell body, which contains a nucleus and the metabolic structures required to maintain the cell. There are two sets of processes extending from the cell body, which are collectively referred to as neurites. These consist of the dendrites, which receive signals and the axon, which transmits them.

The electrical signals in the neural system are called action potentials. They are caused by the ions (sodium and potassium) flowing through specialised channels to change the voltage across the membrane. Neurons communicate through the specialised structures called synapses, which can be either excitatory or inhibitory.

The ability of the synapse linking two neurons to change in strength is one of the important neurochemical foundations of learning and memory. It is not an unitary process but rather many processes on the different time scales representing long-term and short-term plasticity.

Hebbian theory describes the mechanism for synaptic plasticity. However, it cannot be sufficient to fully explain the phenomenon of learning and many varieties of Hebbian-type learning rule evolved. Recent research revealed spike-timing-dependent plasticity, which in a sense is an extension to the Hebbian postulate. It allows modelling more biologically plausible and sophisticated neural networks and provides the answers to shortcomings of the Hebbian rule, supplementing it (e.g.

precisely defining when two neurons are considered as being active together, specifying a mechanism of decreasing the synaptic weight, and evaluating an amount of synaptic weight modification).

Nowadays the STDP process represents a tentative candidate for a hypothesis that fully explains the development of an individual's brain. However, the complexity of described plasticity mechanism has certain restrictions on our ability to model, firstly, due to the lack of available information and secondly, due to the lack of computing power required for simulation.

Chapter 3

Modelling and simulation

The beginning of this chapter describes the different sources of information about the biophysical parameters and functional behaviour of neurons (Section 3.1). Section 3.2 explains the basic models of neurons that are used in most experiments. Their advantages and disadvantages are outlined and the best models for creation of large-scale neural networks are selected. However, the focus is only on those features of neurons and synapses that are most relevant for understanding of the computational models used in the coming chapters.

Later, Section 3.3 overviews the simulation strategies and techniques, particularly paying attention to the information propagation mode in various network models (Section 3.3.2) and the influence of modelling techniques on the efficiency of network simulation (Section 3.3.3).

Section 3.4 gives introduction to the field of distributed computation and discusses commonly addressed issues along the benefits and challenges of using the distributed hardware for neural network simulation. A review of available architectures for distributed neural network simulation is presented in Section 3.5, emphasizing the benefits and shortcomings of each of them. Finally, there is an overview of the existing simulation environments in Section 3.6.

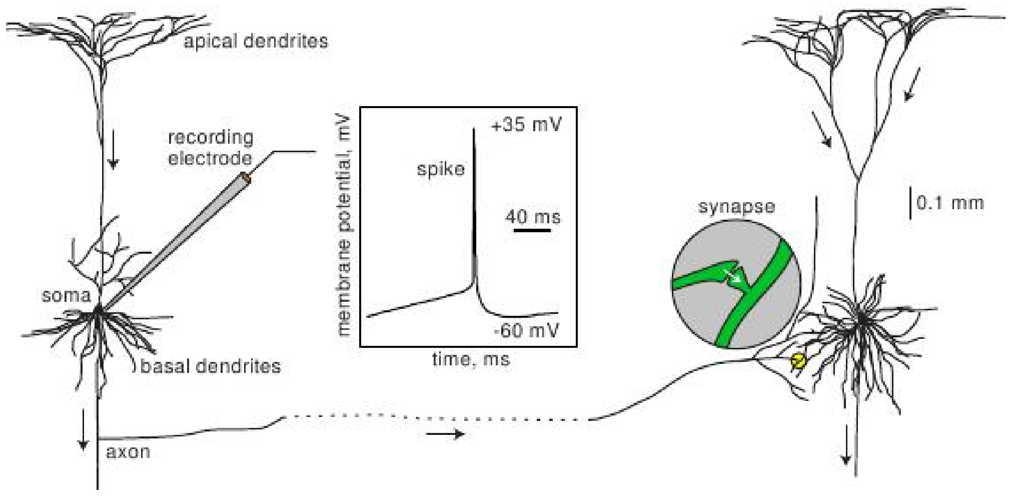


FIGURE 3.1: Two interconnected cortical pyramidal neurons and in vitro recorded spike.

3.1 Information Sources

While making an attempt to implement the neural network simulator one faces the challenging problem of selecting the most appropriate neuron model to use. On the one side, neurobiologists are trying to understand the principles of neural activity. For instance, how it is structured, how it develops, how it works and malfunctions, and how it can be changed. On the other side, computer scientists are questioning what computational models could describe the activity of a neural system the best. What problems can be solved with certain architecture of a neural network with certain types of neuron models connected to each other? What tasks can be solved therewith? Does a certain model give an explanation for various activities, for example in a human brain? These problems are especially acute while implementing a neural system with a large number of highly-connected neurons.

3.1.1 Biological Sources

Neurobiologists study a neural behaviour based on observation of changes in neurons firing rate during experimentally controlled stimulation. Through the emergence of new powerful measurement techniques neural behaviour can be obtained in several ways:

- by detecting the action potential issued by a neuron using an *intracellular electrode*, as shown in Fig. 3.1;

- by **M**ulti**E**lectrode **A**rray (MEA), or microelectrode array, where neuron signals are obtained through multiple plates. They are grouped into implantable MEAs (used *in vivo*) and non-implantable MEAs (used *in vitro*);
- by **L**ocal **F**ield **P**otential (LFP), where the neuron's potential changes in a restricted area of the brain are recorded using an extracellular electrode;
- by **E**lectro**E**ncephalo**G**rams (EEG), where electrodes are placed on a scalp recording the potential changes in large areas of the brain;
- by **f**unctional **M**agnetic **R**esonance **I**maging (fMRI), where observations of changes in brain blood flow indicate the changes in electrical activity of underlying cortical areas, which is caused mainly by the exchange of action potentials between the neurons.

Even with such kind of advances, it is still difficult to determine parameters of biological networks, for instance, such as connectivity or synaptic weighting. Behavioural modelling can be a powerful ally for studying the nervous system.

3.1.2 Computational Sources

Many different models of neurons have been created despite the fact that the understanding of the neuron's structure and particularly the neuron's function at the network level are still limited. Notwithstanding the fact that many of them provide a weak link to the underlying biophysical processes, they are still valuable. These models allow performance of certain experiments via computer simulation that would be impossible to carry out *in vitro*.

The pioneering work of Turing [54], McCulloch and Pitts [55] proposed a neuron model (called the McCulloch-Pitts neuron) trying to prove that neurons are able to handle basic logic functions. They assumed that complex functional capacity observed in the brain (such as attention or consciousness) could emerge from a proper arrangement of a large number of neurons each performing a basic logical gate operation. The state of the neuron can be either "active" or "inactive" and is computed as the sum of all neurons' states connected to the input of the neuron, as shown in Fig. 3.2. Remarkably, networks of such a simple connected communication elements can implement a range of mathematical functions, relating input states to output states. Using algorithms for setting the weights between neurons, these artificial neural networks can "learn".

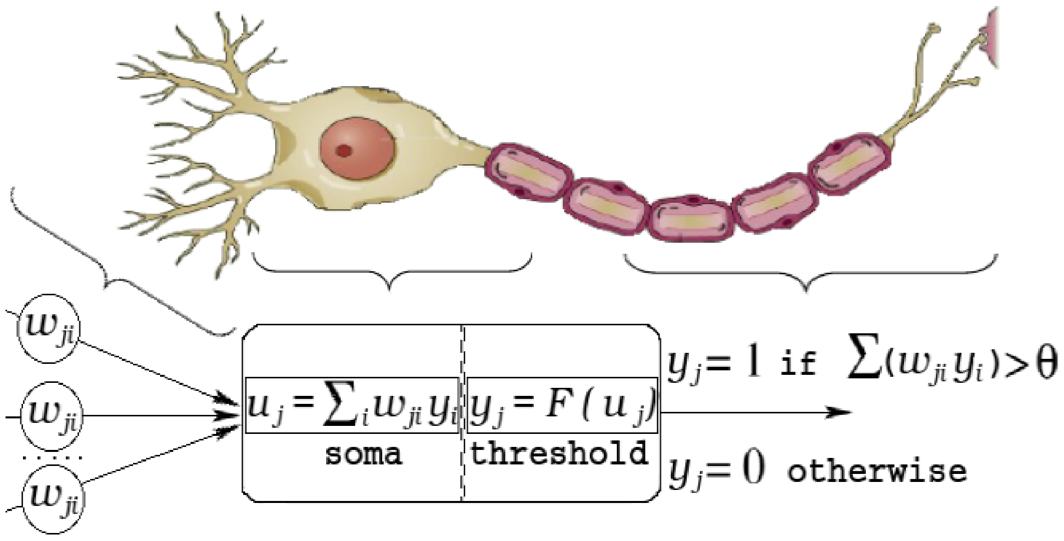


FIGURE 3.2: Simplified neuron model. If the sum of the inputs weighted by w_{ji} exceeds the threshold θ, then the output y_j of the neuron is different from 0.

However, the limitations of these early networks were amply recognized, such as an inability to classify patterns that are not linearly separable in the input space, e.g. XOR funcion [56]. To alleviate these issues, the models with a sigmoidal shape transformation function were developed, creating a graded response instead of original binary thresholding computation. The diversity and complexity of neural processing were reduced to the mere notion of a rate at which a neuron generates spikes. The idea is that when a neuron receives an increasing number of spikes, the higher number of spikes is more likely to be emitted by this neuron. Taking this approach, some of the deficiencies of the earlier neural networks were overcome, such as an ability to learn computing the XOR function, produce pattern recognition and classification or perform unsupervised clustering. The importance of including the changes in neurons firing rate in visual processing has been shown in the work of Hubel and Wiesel, which describes the way signals from the eye generate building blocks of the visual scene (e.g. edge detectors, motion detectors, stereoscopic depth detectors and color detectors) [57].

3.2 Neuron Models

Currently, all neuron models can be divided into two major categories according their plausibility and computational efficiency:

- Biophysically detailed;

- Computationally efficient.

A biophysically detailed model (as example, Hodgkin-Huxley model [11] or Morris-Lecar model [58]) plausibly models the real neuron and accurately reproduces all the processes that are taking place inside of the neuron. Their parameters (such as membrane potential, conductance gating variables or concentration of the intracellular substances) are biophysically meaningful and measurable. But their major problem is that a significant computation is involved in managing synaptic messages and simulation of post-synaptic currents. That is why implementation of such models for large-scale neural networks is extremely expensive and simulation time is greatly prolonged. For this reason they will not be extensively described in this work.

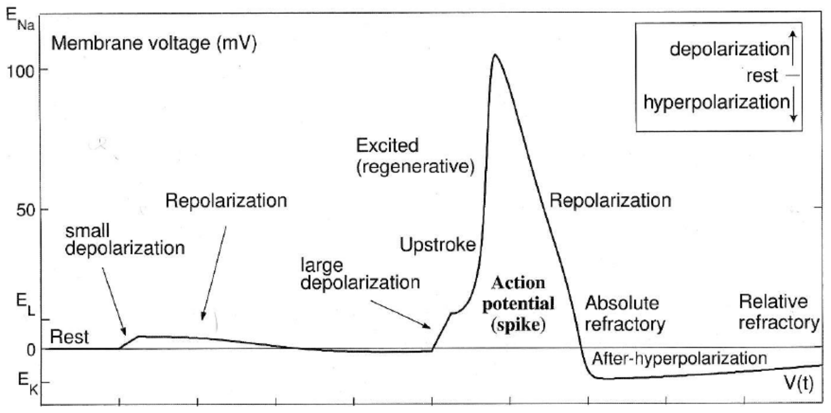
Neurophysiological scientists are using biophysically detailed models to simulate small neural networks in order to most plausibly reconstruct the biological processes occurring in a central nervous system (i.e. brain and spinal cord), revealing the functions and behaviour of different parts of the system. Utilizing biophysically detailed neuron models provides more flexible and realistic behaviour of artificial neural networks. However, at the same stage certain challenges arise:

- **Processing time** is long even for a relatively small networks due to intensive computations required while simulating biologically realistic network. However, this challenge is partly solved with employing distributed system running simulation concurrently.

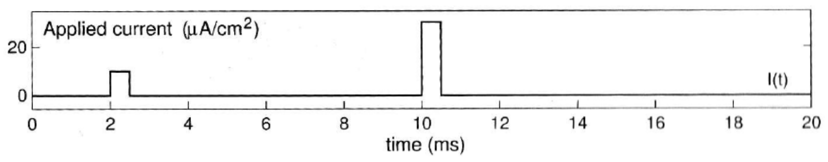
- **Complex dynamics** is involved in the complex behaviour of the network, which should be understood and effectively managed. This is a complex issue as a network dynamics is constantly changing at some level without have a 'stable' state.

- **Limited knowledge** significantly restricts researcher's efforts as an artificial neural network is relatively recent development. Neurophysiological knowledge cannot be easily integrated into the existing models and still many methodologies should be developed and adapted before providing opportunity to construct realistic brain-like model of a neural network.

Computationally efficient models do not have the computational drawback. This is achieved at the price of direct mapping between biophysical parameters of a real neuron and the corresponding parameters of the model. Despite their poor biophysical conformity, they are very popular among engineers due to computational



(a) Membrane potential.



(b) Applied current.

FIGURE 3.3: Action potential generation in the Hodgkin-Huxley model.

efficiency. Computer scientists and engineers leave a lot of biological complexity behind and work with more abstract neural models in order to construct large-scale neural networks. The most significant biophysically detailed and computationally efficient models and their applicability to large-scale network simulations are discussed below.

3.2.1 Biophysically Detailed Models

In 1952 Hodgkin and Huxley [11] performed experiments on the axon of the giant squid and proposed a neuron model, derived from studies on the mechanisms responsible for generation of an action potential in a neuron, which earned them the Nobel Prize (HH model). This neuron model is the most complex but also the most precise existing to date. It consists of four coupled differential equations expressing the dynamics of the membrane potential V_m of the neuron. This corresponds to the potential difference between the neuron and the external environment.

V_m is a function of input current I_{inject} applied to the neuron when it is stimulated, as shown in Fig. 3.3. Currents I_K and I_{Na} are generated by the movement of K^+ and Na^+ ions through the membrane and the leakage current I_l, representing movements of Cl^-. Each of these currents is based on the difference between the membrane potential V_m and the reversal potential E_{Na}, E_K and E_l. The membrane potential of a neuron is described by the equation [11]:

$$\begin{aligned} C_m \frac{dV_m}{dt} &= -I_{Na} - I_K - I_l + I_{inject} \\ I_{Na} &= g_{Na} m^3 h (V_m - E_{Na}) \\ I_K &= g_K n^4 (V_m - E_K) \\ I_l &= g_l (V_m - E_l) \end{aligned} \tag{3.1}$$

where C_m is the specific membrane capacitance, $g_i, i \in \{K, Na, l\}$ are constants and the parameters h, m and n describe the probability of opening/closing of ion channels: sodium h and m and potassium n.

The big drawback of this model is its complexity. Indeed, the coupling of four differential equations makes it extremely difficult to build large networks from such a neuron model. For that reason the HH model usually is used for simulation of one or a couple of neurons. It is necessary to simplify the HH model for simulating at least several hundred neurons for the purpose of studying their dynamics or the plausibility of interaction.

3.2.2 Computationally Efficient Models

One of the simplest and most widely used one-dimensional mathematically efficient models is the **L**eaky **I**ntegrate-and-**F**ire (LIF) neuron model [6]. It acts as a simple incoming spike integrator and compares the sum with a fixed threshold. Comparing to the HH model, LIF model provides the action potential's representation by an instantaneous pulse. Such an instantaneous pulse marks a certain point on the time axis but does not contain any information either in its height or in its shape. In mathematical notation, an action potential is given by a delta pulse at a certain time point $\delta(t - t_i)$.

$$v' = I + a - bv, if \ v \geq v_{thresh}, \ then \ v \leftarrow c \tag{3.2}$$

where v is the membrane potential of the neuron, I is the input current representing the post-synaptic potential received by the neuron (and/or current applied to the neuron), v_{thresh} is a threshold voltage and a, b, c are constant parameters.

This model needs only five mathematical operations to perform one iteration.

However, the model does not provide realistic behaviour close to the threshold and reproduces only some characteristics of a conductance based neuron(e.g. phasic spiking, any kind of bursting, rebound responses, any threshold variability, bistability of attractors, or autonomous chaotic dynamics) [59].

Some improvements to that model have been made at the cost of a second equation describing activation dynamics and adding five mathematical operations to implement spike frequency adaptation and after-potential depolarization [6]:

$$\begin{aligned} v' &= I + a - bv + g(d - v) \\ g' &= \frac{(e\delta(t) - g)}{\tau} \end{aligned} \tag{3.3}$$

The idea is to use an activation gate g that increases after each neuron's firing via the Dirac delta function δ and produces an outward current, slowing down tonic spiking frequency.

Bursting properties and some other implementations, like phasic spiking, rebound spiking and bistability of resting and spiking states have been added by Smith and co-authors in the **L**eaky **I**ntegrate-and-**F**ire-or-**B**urst neuron model [60]:

$$\begin{aligned} v' &= I + a - bv + gH(v - v_h)h(v_T - v) \\ &if\ v = v_{thresh},\ then\ v \leftarrow c \\ h' &= \begin{cases} \frac{-h}{\tau^-} & ,\ if\ v > v_h \\ \frac{(1-h)}{\tau^+} & ,\ if\ v < v_h \end{cases} \end{aligned} \tag{3.4}$$

Here $h\,'$ shows the dynamics of the calcium current, H is a Heaviside function and g, v_h, v_T, τ^+, and τ^- are parameters describing the current dynamics. The price for that is thirteen mathematical operations per iteration, which could be a crucial factor for a very large-scale network.

A simple and efficient analogue of the leaky integrate-and-fire neuron model was presented by Izhikevich [61]. It belongs to a resonator type of neuron models. In contrast to integrators (found among cortical neurons), which perform temporal integration of the incoming spike trains until the threshold is not reached, resonators (found among thalamic neurons) produce a response when stimulated at the resonant frequency. The **R**esonate-and-**F**ire model is described by the following equations:

$$\begin{aligned} v' &= -\frac{v - v_{eq}}{\tau} - cw + I \\ w' &= \frac{a}{\tau_w}(v - v_{eq}) - \frac{w}{\tau_w} \end{aligned} \tag{3.5}$$

Here, v_{eq} is the equilibrium potential, v represents the membrane potential, which is reset whenever it reaches a firing threshold. The recovery variable w characterises the membrane dynamics, τ_ω is the time scale of the dynamics of the w variable, and τ, a and c are parameters. This model is able to represent subthreshold damped oscillations of membrane potential (threshold depends on the prior activity of the neurons) and resonance (neurons can respond selectively to the inputs of similar frequency with subthreshold oscillations) as well as autonomous chaotic activity. To implement a resonate-and-fire neuron model one needs 10 floating operations per one iteration.

Another alternative is the **Q**uadratic **I**ntegrate-and-**F**ire (QIF) neuron model, also known as the theta neuron [62]:

$$\begin{aligned} v' &= I + a(v - v_{rest})(v - v_{thresh}) \\ if\ v &= v_{peak}, \quad then \ \ v \leftarrow v_{reset} \end{aligned} \tag{3.6}$$

where v_{rest} is the resting membrane potential and v_{thresh} is the threshold value of the membrane potential. This model can represent spike latency, threshold variability (which is v_{thresh} only if $I = 0$) and bistability of resting and tonic spiking modes. This is quite a limited model in terms of plausibility and it needs seven mathematical operations for performing one iteration. Quadratic integrate-and-fire model could be a reasonable substitution for the basic integrate-and-fire model implementing a large-scale network.

Izhikevich presented another model [7], successfully reproducing the behaviour observed in many biological neurons. His model is based on the Fitzhugh-Nagumo model [63, 64] and is described by the following coupled differential equations:

$$\begin{aligned} v' &= 0.04v^2 + 5v + 140 - u + I \\ u' &= a(bv - u) \end{aligned} \tag{3.7}$$

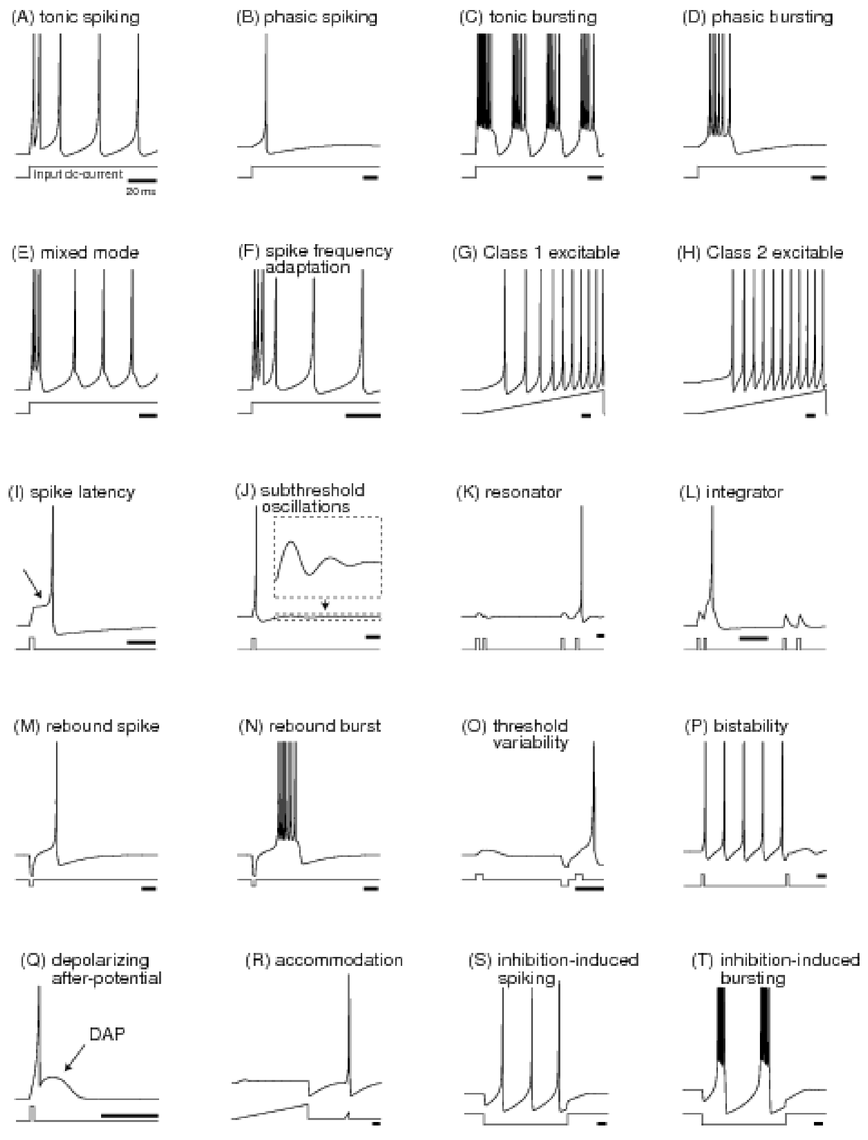


FIGURE 3.4: Neuronal responses obtained for different values of the four model parameters a, b, c, and d (see Fig. 3.5). Each is labelled with the corresponding biological behavior. [6].

with the auxiliary after-spike resetting

$$if\ v \geq +30mV,\ then \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{3.8}$$

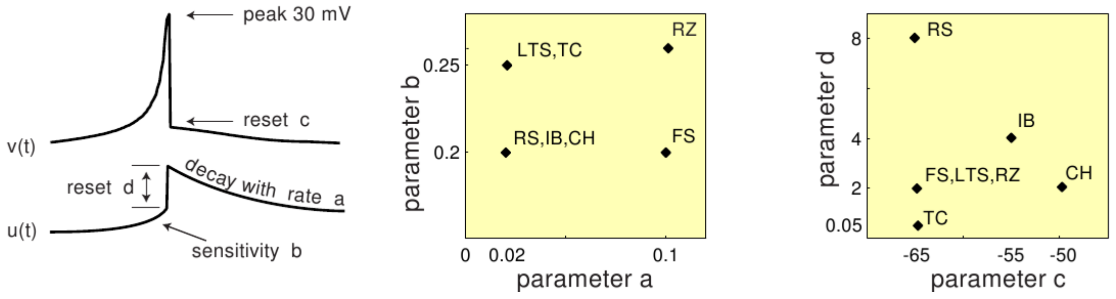


FIGURE 3.5: Choice of a, b, c, and d parameters for representation of regular spiking (RS), intrinsically bursting (IB), chattering (CH), fast spiking (FS), thalamo-cortical (TC), resonator (RZ), and low-threshold spiking (LTS). [7].

where v and u are dimensionless variables, which correspond to the membrane potential and the recovery of the membrane. The Eq. 3.7 and Eq. 3.8 represent the behaviour of the membrane's voltage and the recovery variable after reaching the action potential's peak value (+30 mV) or any higher value. The rest value is dynamic and is located between -70 and -50 mV. After the potential reaches a threshold ($+30\ mV$), the membrane potential and the recovery variable are reset, as shown in Eq. 3.8. The part $0.04v^2 + 5v + 140 - u + I$ is chosen empirically to scale v to mV and time to ms [6].

Various choices of the a, b, c and d parameters result in various intrinsic firing patterns, Fig. 3.5. Izhikevich states in [7] that his model can exhibit all known types of firing patterns in case of the proper choice of a, b, c and d parameters, Fig. 3.4. It is necessary to perform 13 floating-point operations to execute one iteration. The computational load is similar to the integrate-and-fire-or-burst model, but the behavioural properties produced by the Izhikevich model are closer to those observed in Hodgkin-Huxley-type models and therefore it is a more desirable model for neural network simulations.

Despite the coupling of its two differential equations, this model is relatively simple compared with HH-type models and is particularly suitable for simulations of spiking neural networks [6].

As a result of different computationally efficient neurons' models review, it is necessary to emphasize that only few of them are widely used: one-dimensional *leaky integrate-and-fire model* [6], requiring only 5 mathematical operations per one iteration and opposite both in computational complexity and the variety of the behavioural properties to the *Izhikevich model* [7], requiring 13 mathematical operations per one iteration. The former one is widely used due to its facilitated mathematical analysis of neurons population and its computational efficiency. Comparison of the all above described models is in the Table 3.1.

Models	LIF [6]	LIF with adaptation [6]	LIF-or-burst [60]	Resonate-and-fire [61]	Quadratic LIF [62]	Izhikevich [7]
tonic spiking	+	+	+	+	+	+
phasic spiking	-	-	+	+	-	+
tonic bursting	-	-		-	-	+
phasic bursting	-	-	+	-	-	+
mixed mode (bursting then spiking)	-	-	-	-	-	+
spike frequency adaptation	-	+	+	-	-	+
class 1 excitable	+	+	+	+	+	+
class 2 excitable	-	-	-	+	-	+
spike latency	-	-	-	-	+	+
subthreshold oscillations	-	-	-	+	-	+
resonator	-	-	-	+	-	+
integrator	+	+	+	+	+	+
rebound spike	-	-	+	+	-	+
rebound burst	-	-	+	-	-	+
threshold variability	-	-	-	-	+	+
bistability	-	-	+	+	+	+
depolarizing after-potential	-	+	+	+	-	+
accommodation	-	-	-	+	-	+
inhibition-induced spiking	-	-	-	-	-	+
inhibition-induced bursting	-	-	-	-	-	+
chaotic activity	-	-		+	-	+
# of FLOPS	**5**	**10**	**13**	**10**	**7**	**13**

Table 3.1: Comparison of the computational properties of neuron models, after [6]. Sign "+" means ability to implement and "-" means inability to implement the given feature by certain model.

3.3 Simulation Strategies

Simulation is the connecting link between neurophysiological measurements and theoretical studies. It helps us to understand and to reproduce the behaviour of biological systems of neurons and to verify functional behaviour of network models. It also validates or refutes certain assumptions made in neuroscience by employing appropriate network models. As an example, the study of the network dynamics (the way a network changes the weights over time) or the distribution of weights after learning offers an interesting perspective to explore new opportunities of the brain.

This section reviews the common simulation techniques and approaches used today for neural network simulation. It starts with a review of connectivity rules, overviews the signal propagation modes and completes with a discussion about the ways to trail the flow of simulation.

3.3.1 Network Structure

To interconnect neurons in a network, certain connectivity defining rule can be applied for the entire network or a few rules can coexist in a simulation. One of the intuitive ways is a fully-connected network, where every neuron is connected to all other neurons of the network. Each connection is one-way only, so there are two connections between every pair of neurons. A neuron i through its axon is connected to a dendrite or a cell body of a neuron j, while the neuron j is connected to a dendrite or a cell body of the neuron i. A network of N neurons connected in this way has $N(N\text{-}1)$ connections, with the condition that self-connections are omitted.

The connectivity can be of many other types, such as connectivity controlled by the Gaussian distribution (for instance, receptive fields of sensory neurons in the visual system), interconnecting all neurons or randomly connecting neurons in a network. One popular architecture is the layered network, where the N neurons are grouped in a number of layers. Starting from the simple concept of a neuron "layer", it is possible to define a network model as specific as necessary.

The choice of the term "layer" should not mislead here because it can be considered as unordered set of neurons or as one-dimensional, two-dimensional or three-dimensional vector. An interconnection is called an intra-layer if the pre- and post-connections are in the same layer. Only the applied rules and the connectivity graphic representation (if any) of the network could visually define the pattern of connectivity.

An appropriate way of dividing a network into sub-networks and interconnecting them properly could partially solve the problem of network complexity. This approach is particularly worth attention in view of the fact that the study of cortical neuroanatomy shows the repetition of small basic unit topologies in the cortex. They differentiate by the types of neurons and the neural densities inside. Some of them are further divided into 'sub-layers'. Although their function is still unknown to date [65, 66], the entire neocortex consists of a juxtaposition of these units, called cortical columns [67, 68, 69]. It is a group of neurons, stetching vertically through the layers of cortex. Each column encode similar feature and has afferent input (e.g. from receptive field), efferent output (e.g. to motor neurons), and intra-cortical circuitry with both excitatory and inhibitory neurons. This kind of structure has been found in dogs, cats, monkeys, and also in other mammals both as an anatomical and functional structure [70].

It must be noted that in biological networks neuron inter-connections have a high level of diffusion and mixing. A single neuron does not have a large influence on any other neuron and only a group of firing neurons can cause a post-synaptic neuron to fire. However, it is not a case for all brain areas. For example, in the cerebellar cortex there is only one cerebellar climbing fibre (one of the two axon types that enter the cerebellum) per Purkinje cell. This sole climbing fibre is powerful enough to activate the target Purkinje cell.

While setting up a network, a definition of an optional network structure should be possible for a generic set of connectivity rules. For instance, it can be defined in a XML configuration file. This gives an opportunity of creating a wide range of various network topologies, such as a feedforward topology of a multilayer perceptron, a recurrent topology of a Hopfield network, or a convertible topology of a self-organizing map.

3.3.2 Activity Propagation

Biological neurons are the actual processing units of the brain. The computation they do is slower compared to a silicon chip. A network of a large quantity of these simple units however proves to be very powerful. A silicon-based computer usually has one or a few processing units, while in a neural network all the neurons work in parallel. To form this network, on average every neuron is connected with thousands of other neurons.

Since the early 80s, many network models based on models of non-spiking neurons were developed and studied. These network models can be applied to spiking neurons and their effectiveness has been proven [71]. Generally there are *feed-forward* and *recurrent* networks with several variations.

3.3.2.1 Feed-forward Neural Network

One popular architecture is a feed-forward network, in which activity flows from input to output and the network topology contains no cycles. The network built according to this model often is called a layered network because its processing is based on a succession of neurons layers, as shown in Fig. 3.6.

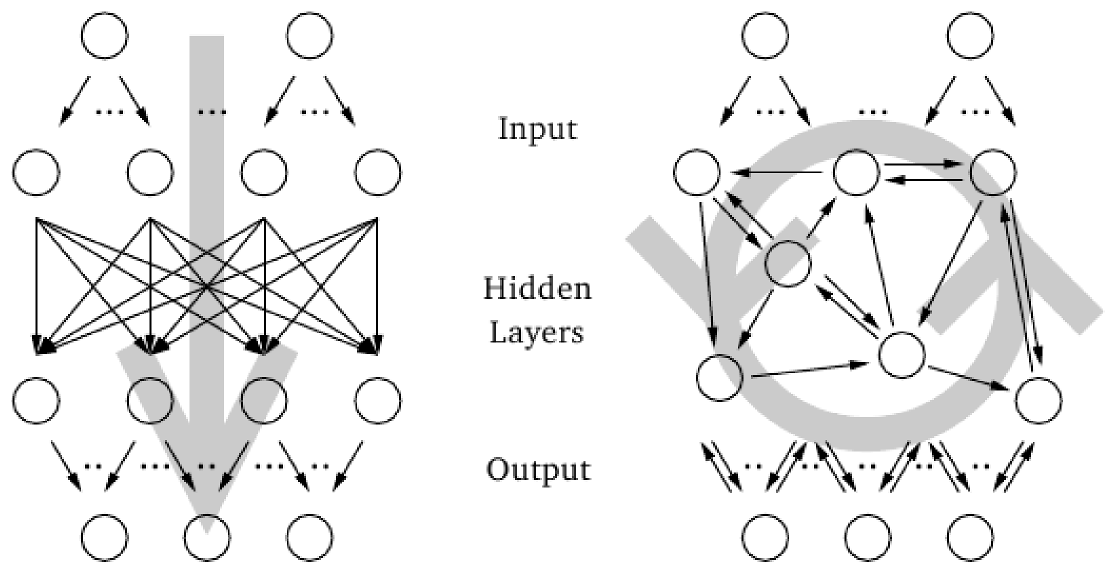


FIGURE 3.6: Comparison of the architecture of a feed-forward (left hand side) and a recurrent neural network (right hand side); the grey arrows sketch the direction of computation.

In the layered network there is an input layer consisting of N_I neurons, an output layer consisting of N_O neurons and a number of hidden layers in between, each hidden layer consisting of N_H neurons. If the layers are sequentially numbered starting with the input layer and ending with the output layer, then feed-forward means that every neuron in layer i has only connections to the neurons in layer j if $j > i$. In most cases there are only connections to the layer next to the current layer: a neuron in layer i has only connections to neurons in layer j if $j = i + 1$ (most often every neuron connects to all neurons in the subsequent layer).

Biological observations suggest that the visual cortex of a human brain spreads neuronal activation as a feed-forward wave [72]. After a stimulus is applied to receptive fields of the retina, activation spreads through the cascade of feed-forward connections between the successive hierarchical levels. Short latencies between hierarchical levels (about 10 ms [73]) suffice only for a single spike generation per cortical neuron before activation of the next level [74], [75], leaving no time for any lateral or feedback connections to exert their effect. The propagation is completed within approximately 100 ms, which is not enough for detailed scene awareness but suffices for high-level category-selective information processing (such as object recognition or categorization) [72].

The radial Basis Function [76] and the Multilayer perceptron [77] in general fall in the category of feed-forward networks.

3.3.2.2 Recurrent Neural Network

Contrary to feed-forward networks, recurrent neural networks are models with a bi-directional activity flow. A network is said to be recurrent if there exists at least one cycle in the network topology. For example, starting from a feed-forward network, you can add feedback connections, in this way modulating inputs using the previous outputs (Fig. 3.7 a). This can also lead to connections within a single layer known as lateral connections (Fig. 3.7 b).

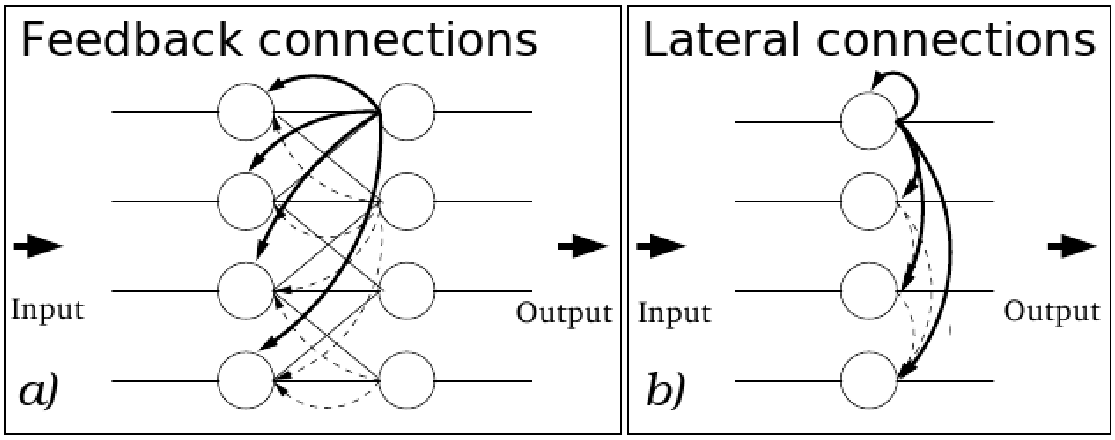


Figure 3.7: Various types of connections in a recurrent networks.

With the recurrency, the network can retain certain level of activity over time, even in the absence of input. This provides a sort of dynamical memory within the network, enabling to process temporal context information and to compute functions that are more complex than just simple reactive input-output mappings. Self-sustained activation dynamics makes it a *dynamical* system, whereas feed-forward networks are *functions*. This allows a system to incorporate a much richer range of dynamical behavior.

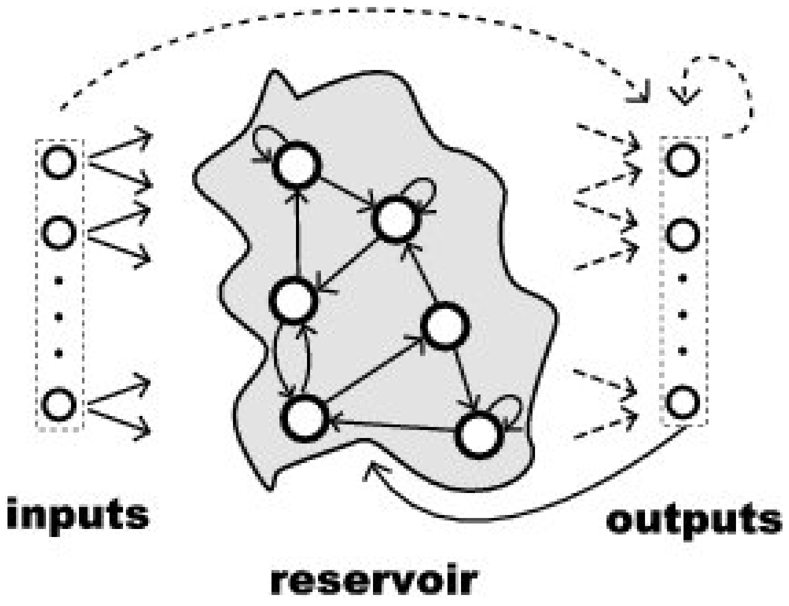


Figure 3.8: Reservoir computing.

In the narrower context of spiking neuron models, many recent works focus on better understanding of how to handle these new neuron models to take advantage of their potential computational power. Many approaches have been elaborated on this topic. Among them are simple recurrent networks, such as the Elman network (recurrency through a feedback from the hidden layer, maintaining a copy of the previous values of the hidden units, see [78]) and the Jordan network (similar to the Elman network but with recurrency through a feedback from the output layer, see [79]), self-organizing map (using the competitive learning model with a neighborhood constraint on the output units in order to produce a certain topographic map in response to a corresponding input pattern, see [80]), and associative memory models, such as Hopfield network (using binary threshold units with symmetric connections, accepting only stationary input and guaranteed to converge to a local minimum, see [81]) and interactive activation model (consisting of competitive pools of processing units with excitatory bidirectional connections among the different pools and inhibitory unidirectional connections within the same pool, see [82]).

3.3.2.3 Reservoir Computing

Some approaches attempt to express the behavior of neuron populations using mathematical equations [17, 83]. Other studies use detailed neural networks models applying a neuron reservoir computing approach, based on treating the recurrent part (the *reservoir*) separately than the readout nodes. *Echo State Networks*

[84], *Liquid State Machine* [85], and *Backpropagation–Decorrelation* [86] are the main paradigms of reservoir computing models which significantly facilitate the application of recurrent neural networks.

The reservoir consists of a collection of recurrently connected nodes. The connectivity structure is usually randomly created, and the units are usually non-linear. The weights for the connections within the dynamic reservoir are chosen at the beginning with a random distribution. The overall dynamics of the reservoir is passively excited by the input signal and also is affected by the past input history. The desired output signal is generated as a combination of the input and excited reservoir activities. A rich collection of dynamical input-output mapping is a crucial advantage over simple time delay neural networks. Various dynamics can be observed while applying a large enough reservoir with sparse connections. This can be used to compute many complex output functions [87].

The idea behind this is that one does not try to set the weights of the connections within the pool of neurons but instead reduces learning to setting the weights of the readout neurons. This facilitates learning dramatically and much simpler supervised learning algorithms can be applied, for example minimizing the mean square error in relation to a desired output. Also one of the many available linear regression algorithms can be used instead of specialized gradient descent based algorithms, which aim at iteratively reducing the training error and typically are slow, computationally expensive and can end up in a local minimum.

3.3.3 Modelling Techniques

Despite a steady improvement of computational hardware and ever-growing knowledge about functionality of a biological neuron and its modelling techniques, numerical simulation results are still tightly bound to the selected modelling approach.

Conventionally, a time-driven approach was used for neural network simulation. In such an approach, the temporal interval is chosen, during which a system recalculates its state. Such an algorithm can be easily coded and applied to any model. But recently simulation based on event-driven approach has gained recognition [88], [89]. Modern event-driven methods can be applied to almost every neuron model and therefore challenge the traditional time-driven methods. Each strategy has its own assets and constraints.

3.3.3.1 Time-driven Technique

In the time-driven approach, the temporal interval is chosen, during which a system recalculates its state, i.e. scrolling all the neurons and synapses of the network at each time step. Therefore, the precision of simulation depends on the chosen temporal interval. If the interval is very large, the system can miss some neural activity and thus the simulation result will not be realistic. Conversely, if the interval is too small, the computations will overload the system and slow down the operational speed [8]. Spike timing is aligned to a time grid, thus the simulation precision is approximate even if the differential equations are computed accurately. Modern time-driven simulators mostly use 1 ms or less time interval in order to find the trade-off and to achieve the accurate simulation result [6, 88, 90, 91]. Another drawback of this strategy is the linear dependence of the computational load on the number of the neurons in the system. Threshold conditions are checked only at the ticks of the clock and that can lead to some spikes being missed [8]. Time-driven technique can be easily coded and applied to any model.

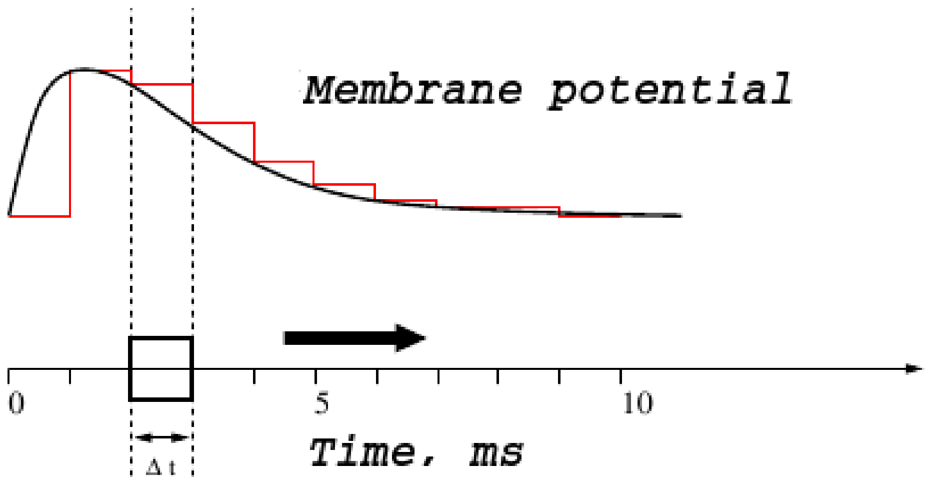


FIGURE 3.9: Approximating the value of the membrane potential in time-driven simulation with every time step Δt.

Standard Euler or second-order Runge-Kutta numerical integration algorithms could be used for spike prediction when the differential system is used for describing the dynamics of the neurons [59]. However, the spike can occur earlier or later if, due to the approximation error, the predicted value of the membrane potential reaches the firing threshold level at an inappropriate time. The overall impact of such an errors is difficult to determine [92], [93].

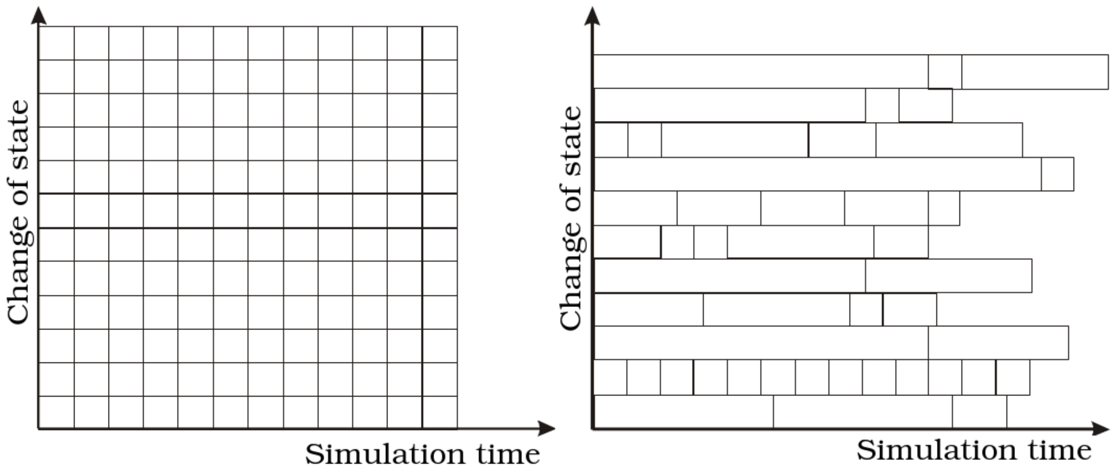


FIGURE 3.10: Change of state in time-driven (left) and event-driven (right) simulation.

3.3.3.2 Event-driven Technique

In contrast to the time-driven approach, event-driven simulation is free from dependence on the temporal resolution. There is no need for a discretisation of time other than that enforced by the finite precision of the floating-point arithmetic of the computer. But it is significantly more complex to implement and less universal. The simulation environment does not progress step-by-step through time, but instead jumps ahead to the next *event* scheduled to occur, as shown in Fig. 3.10. An event is any occurrence that changes the state of the model.

In a neural network simulation, events include firing of neurons and an arrival of spikes at synapses. Events cause a change in the state of the system and often result in the scheduling of other events for some future time. A cell recomputes its state after every spiking event occurs in the network. Such a method allows time saving due to skipping of the calculation update step when a neuron does not get any event.

However, in a large neural networks with high level of connectivity (about 10^4 targets per neuron) an event-driven strategy leads to an immense overhead. A single spike generates up to 10^4 even objects, which have to be allocated, queued, sorted, delivered and deallocated. This results in decelerated simulation execution.

While a high activity period of simulation benefits to the time-driven approach due to the gain in speed when the temporal resolution integrates several events, all low activity periods advantage the event-driven approach. According to biological observations, the neurons of a spiking neural network are sparsely and irregularly connected in space (network topology), and the variability of spike flows implies they communicate irregularly in time (network dynamics) with a low average activity.

Since the activity of an spiking neural network can be fully described by emissions of dated spikes from pre-synaptic neurons towards post-synaptic neurons, an event-driven approach is clearly suitable for sequential simulations of spiking neural networks [94], [88], [95], [89]. Furthermore, a time-driven simulation loses the order of emission of spikes emitted at the same time interval, which is important for spike-timing-dependent plasticity and can change the behaviour of the simulation [96]. In an event-driven simulation, temporal precision only depends on the precision of the variable used for time stamps and clocks, and even a simple 16 bit variable gives a suitable precision [97].

Modern event-driven methods can be applied to almost every neuron model and therefore challenge the traditional time-driven methods. However, an event-driven simulation does have some drawbacks. The state of neurons is only actualized when events are processed, ignoring the subthreshold variations of the membrane potential. Although it is possible to make a prediction of the spike emission date when the behaviour of the neuron is described by several differential equations and/or when synaptic impacts are not instantaneous [95]. Such a prediction implies heavy computation costs, and a mechanism must control *a posteriori* the correctness of the prediction.

Event-driven simulations are most appropriate where all activity can be broken down into events that take place instantaneously, allowing the simulation to bypass all the time between events when nothing changes. This is the case of pulse-coded integrate-and-fire models, but not the Hodgkin-Huxley type of models. Thus, the event-driven approach substantially reduces the computational cost of simulators that control exchanges of dated events between event-driven cells, without checking each cell at each time step. However, the system's average activity increases with the number of neurons employed in the network. This is why the number of spiking events is large in the large-scale networks as network activity increases with growing number of neurons. Use of the exact time of the event can noticeably complicate the system's state computation. That fact can be crucial for large-scale networks. However, applying massively parallel systems can reduce the computational cost by processing a high number of events in parallel when activity is distributed among the processing units. Another crucial component is the event queue management.

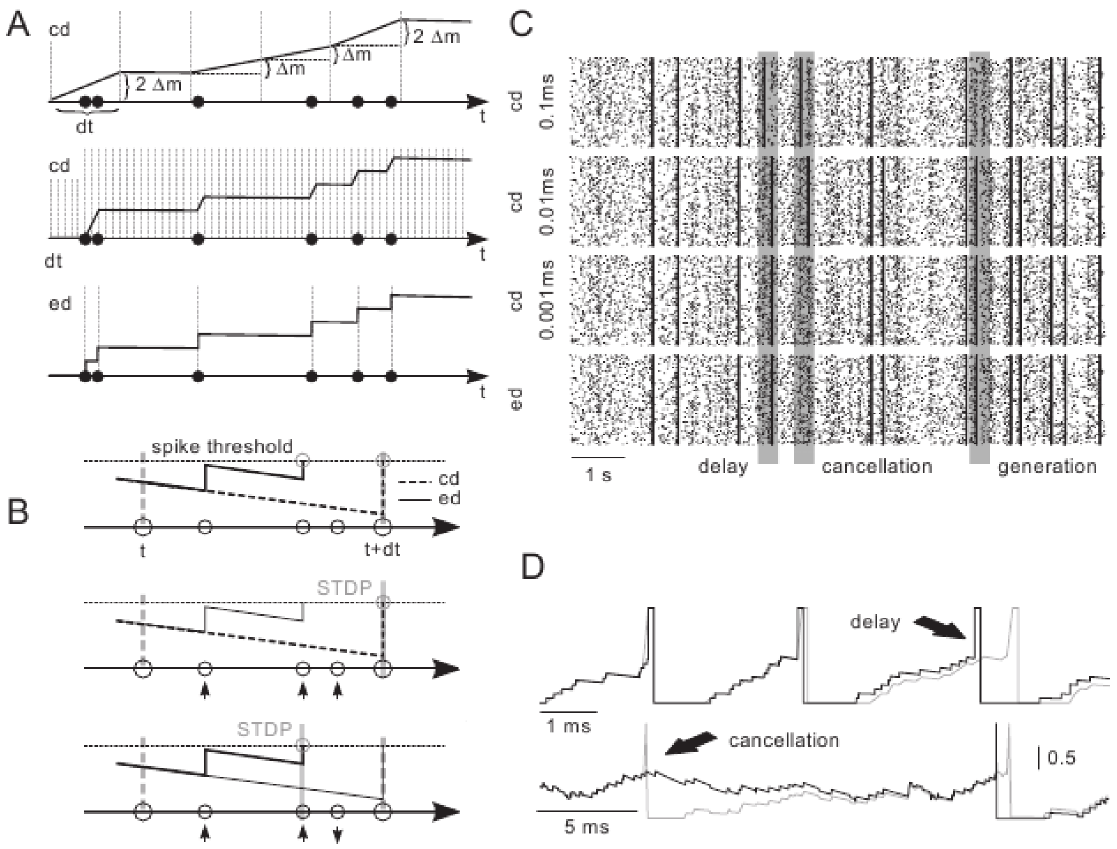


Figure 3.11: **A.** Comparison between time-driven (cd, top: low temporal resolution; middle: higher resolution) and event-driven (ed) technique. **B.** Impact of simulation strategy on facilitation and depression of synapses. **C.** Differences in the synchronous event occurrence for various temporal resolutions. **D.** Small differences in spike times can accumulate and lead to severe delays (top, arrow) or even cancellation (bottom, arrow) of spikes (modified from [8]).

3.3.3.3 Temporal Precision

Rudolph and Destexhe [8] made their experiments with the temporal development of individual synaptic weights employing both time-driven and event-driven strategies. They stated that a an inappropriately chosen temporal interval can significantly influence the accuracy of simulation, producing severe delays that eventually lead to additional spikes or spike cancellations. Comparing the weight development for specific synaptic channels, the authors report that both simulation strategies present dramatically contrasting results. Time-driven simulation leads to about a 10 % higher firing rate compared to the event-driven simulations over the whole simulated window (applying 0.1 ms temporal precision). Fig. 3.11 shows that small differences in the precision of synaptic events can have severe impacts.

Concluding this section, it is important to point out that any network can be divided into sub-networks, or layers, with extra-layer and intra-layer connections. The network can propagate data in a feed-forward mode sending data from input to output without any cycles or in a recurrent mode with a bi-direction data flow, which provides a sort of memory within the network. Reservoir computing is an example of a recurrent network, the dynamics of which is driven by the input and is also affected by the past. Despite discrete event-based simulation's drawbacks, this approach is more advanced, because of the higher precision of simulation (every event is fixed during simulation) and lighter computation (spiking activity is not a uniform process and on average the number of spiking events is smaller than in the time-driven approach). Also, the computational cost can be significantly reduced for the networks with uniformly distributed activity, by implementing on massively parallel systems.

3.3.4 Learning Paradigms

There are several learning paradigms organized into a taxonomy corresponding to a particular abstract learning task.

3.3.4.1 Supervised Learning

Supervised learning is widely used in conventional neural networks based on threshold or sigmoid neurons (Perceptron [98], Adaline [99], backpropagation multilayer perceptron [100]). The idea is to infer a function from pairs of input object (typically a vector) and a desired output supervisory signal, which constitute a training data. The inferred function evaluates the difference between a calculated output vector and expected output vector and is either a classifier (for discrete output) or regression function (for continuous output). The least squares or gradient descent methods are the most convenient for supervised learning.

The simulations using supervised learning are generally divided into two phases:

1. A learning phase, during which the weight vector is modified according to the chosen method applying the example set of patterns.
2. A generalization phase, during which the weight vector is set to the values, obtained after the learning phase and the network is applied to the processing of the non-learned patterns.

The achieved learning quality is evaluated by the network's ability to generalize [101]. For instance, in classification, the examples used during the learning phase must be determined so as to enable the network to efficiently approximate the various classes. The neural network performs a separation of possible entries into several clusters whose form and number depends on the used type of neurons and the network size.

The main disadvantage of this type of learning is that although a network will be able to generalize to achieve the task in the particular manner, it will not be able to explore novel ways to achieve its goals.

3.3.4.2 Reinforcement Learning

Reinforcement learning is another training method that is capable of learning goal-directed behaviour. It differs from standard supervised learning in that correct input and output pairs are never presented nor sub-optimal actions are explicitly corrected. The network is guided by a reward function, which is unalterable and defines what is objectively "good" and "bad" so that learning is conducted through trial and error.

Reinforcement learning can be categorized into three classes: actor-only networks, critic-only networks, and actor-critic networks. Actor-only networks [102, 103] learn the policy directly by adjusting their parameters as indicated by the reinforcements signal. Critic-only networks [104] learn an approximation of the value function, which is then used to derive the policy. Actor-critic networks combine the above two by feeding the output of the critic network to the actor network [105].

Reinforcement learning is particularly well suited to problems, which include long-term versus short-term reward trade-off. The method has been used successfully to train neural networks in a number of domains such as controlling an inverted pendulum [106] and landing an aircraft [104].

3.3.4.3 Unsupervised Learning

Unsupervised learning includes a wide spectrum of learning methods. The processed data in unsupervised learning is not associated with desired outputs, unlike supervised or reinforcement learning. Some forms of unsupervised learning observed in biological neural networks are outlined in the Hebbian law. The law states that the joint activation of two connected neurons strengthens their connections and thus facilitates their joint activation in the future (see 2.3).

Hebbian law has been implemented in various forms, start from the Hopfield network [81], self-organizing map [80] and adaptive resonance theory [107]. More recently, this law has often been revised in a temporal context: modelling of spiking neural networks directly exploit these principles to model a form of synaptic plasticity as a rule of unsupervised learning [49].

Summarizing, learning algorithms can be grouped into supervised, reinforcement and unsupervised learning. A supervised learning algorithm generalizes from the training data to unseen situations in a “reasonable” way. Reinforcement can be defined as the trial-and-error “law-of-effect” learning, which balances between *exploration* of unknown states and actions and *exploitation* of current knowledge. Unsupervised learning does not require pre-defined output data and usually is applied for the tasks with known learning input samples while searching for relations among system’s stimuli and responses. In our project we combine reinforcement and unsupervised techniques based on the biological inspirations described below.

3.4 Distributed Computing

Although the use of a single-core computer is possible in principle, execution of the simulation on multiple cores is beneficial for a very large neural network for the following reasons.

Need of Communication Network First of all, the very nature of the neural network requires the use of a communication network that connects several computational units. This happens because the data is produced in one physical location and is sent to another. Therefore it is more cost-efficient to obtain the desired level of performance by using a cluster of chips, each making its own computation in comparison with a single chip performing all the computation itself. One can reduce the execution time by up to a factor equal to the number of processors that are used [108].

Faster Execution Biological systems are slow comparing with modern computers, operating at several GHz. However, the massive population of neurons, high density of connectivity, and high frequency of interactions compensate their natural slowness. When implemented on distributed hardware, neuron networks take full advantage of their inherent parallelism and run orders of magnitude faster thus becoming appropriate for real-time applications.

Computational Reliability Another reason is increased tolerance to failures and thus computation reliability, enabling a system to continue operating properly in the event of one or more faults within some of its components. A distributed system is more reliable than a non-distributed one, as there is no single point of failure. If one processor fails, it may be possible for other processors to continue the simulation provided critical elements do not reside on the failed processors or dynamically redistribute to the working nodes.

Scalability Additional interest for supporting such a type of computing gives high scalability due to the fact that a distributed system may be easier to expand and manage than a monolithic uniprocessor system.

Enough On-chip Memory Finally, there is a prospect of using a number of processors, each with a high enough on-chip memory. This makes possible to simulate large networks with high accuracy of the neuron model, giving the opportunity to implement and investigate learning dynamics.

3.4.1 Distributed Simulation Challenges

Distributed computation of separate network segments comes at the cost of additional distribution-related communication. It is obvious that a distribution benefits only from a certain network size as the layout of the network generally takes computing time for communication between processors. Moreover, an uneven computation load distribution among sub-processing units further burdens the simulation speed. Evenly distributing the computational load and using the techniques to reduce communication among the involved processing units significantly improves simulation speed and facilitates program maintenance.

Another issue concerns a distribution of the information, required for computation. In a sequential system with a single processing node all information is stored in the system memory. Conversely, in distributed system with multiple processing nodes keeping some part of information remotely from a processing node causes inefficiency and optimization is necessary before mapping.

In the case of a neural network simulation, however, the physical mapping problem is simplified because each individual neuron is a self-contained unit, and all communication is limited to the short-lasting impulses carrying no information in their shape or size, but the time of spikegeneration. But issues such as storage, communication mode and synchronization still should be addressed before workload distribution by mapping a neural network onto different hardware processing units.

3.5 Available Architectures

The last 50 years have witnessed considerable research in the area of neural network implementation on distributed systems, resulting in a range of various architectures. Here we overview the most popular of them.

3.5.1 Parallel Computers

Parallel computers are typically classified into two categories, namely **S**ingle **I**nstruction, **M**ultiple **D**ata (SIMD) and **M**ultiple **I**nstruction, **M**ultiple **D**ata (MIMD).

textbfSIMD usually has a high number (10^4) of very limited-functionality processing elements that all execute the same instruction (i.e. program). In this way such machines exploit data level parallelism. This type of computer was dominating the scene until the MIMD class of computers became more powerful, and interest in SIMD waned.

MIMD are usually built with a moderate number (10^2) of general-purpose processors that function asynchronously and independently. At any time, different processors execute different instructions on different pieces of data. MIMD machines can be of either *shared memory* or *distributed memory* categories. These classifications are based on how MIMD processors access memory.

In the **shared memory MIMD model** (MIMD-SM) all processors are connected to a “globally available” memory. Shared memory model is less flexible than the distributed memory model (MIMD-DM), where each processor has its own individual memory location and no direct knowledge about other processors’ memory.

In the **distributed memory MIMD model** (MIMD-DM) a message must be passed from one processor to another in order to share some data. Since there is no shared memory, data contention is not as great a problem with these machines. Thus, in recent years interest in applications exploiting MIMD-DM architectures is growing. The communication between the nodes is explicitely included in the program by calls to a communication API (e.g. MPI or PVM). The most common form of MIMD-DM network architecture is a workstation, where each machine has its own physical memory.

A combination of shared and distributed memory is possible when each machine's processing unit is operated as such in the designed distributed system. It is also possible to use distinct light-weight multiple processes (or *threads*) on each machine. In this case, some messages are sent over the network to other processors while other will be sent via the local memory between the threads [109].

3.5.1.1 Single Program Multiple Data

Most often tasks are split up and run simultaneously on multiple autonomous processors executing different portions of program on different data. Each processor executes part of the code according to its identification number. Different instances of the program communicate with each other by exchanging messages. This mode, which uses the same program executing different data on each processor is named ***S**ingle **P**rogram **M**ultiple **D**ata* (or SPMD) and is a subcategory of MIMD.

3.5.2 Application-Specific Integrated Circuit

Application-**S**pecific **I**ntegrated **C**ircuit (ASIC) traditionally referred to as neuroprocessor or neuron chip [110, 111].

A notable example of an ASIC is the Blue Brain Project, using the IBM Blue Gene supercomputer with 8000 CPUs to simulate neurons with STDP learning in software [112]. The distinctive feature of the project is its goal to simulate the ion channels and processes of neurons at the fine-grain compartmental level with a high degree of biological realism. A 10000 neuron model of a neocortical column of a 2-week-old rat was successfully simulated on the Blue Gene platform. This part of the somatosensory cortex is the smallest functional unit of the neocortex, which is thought to be responsible for higher functions, such as conscious thought.

Another notable example of a flexible application-specific integrated circuit is the SpiNNaker neuromorphic hardware, which provides a universal platform to simulate large networks in biological real time [113]. As well as IBM Blue Brain Project, SpiNNaker represents a new breed of cluster computers where the number of processors, instead of the computational performance of each individual processor, is the key to high performance. It uses highly-parallel computing distributed on 20 identical ARM968 processing subsystems with high-bandwidth inter-process communication through the self-timed links. The key feature of SpiNNaker design is fault tolerance. The basic approach is a combination of redundancy and reconfigurability, so that the work of any failing processor can be dynamically re-mapped without harm to the whole system.

However, developing custom ASIC devices for neural networks is both time consuming and expensive. These devices are also inflexible as a modification of the basic neuron model requires a new development cycle to be undertaken; unless a flexible neuromimetic system is developed, such as SpiNNaker.

3.5.3 Field Programmable Gate Array

Another type of dedicated hardware architecture is the **F**ield **P**rogrammable **G**ate **A**rray (FPGA), which compromise a large number of logic cells with rich interconnectivity resources. Due to the reconfigurable nature of the device, the employed neuron model can easily be modified and a new bitstream generated and downloaded an unlimited number of times. Several concepts for neural network implementation have been created on FPGAs ([114], [115], [116]).

The fundamental problem limiting the size of neural network realization on the FPGA is the cost of implementing the multiplications associated with the synaptic connections. A fully parallel neural network requires a large number of multipliers, which limits the size of the network. Several techniques were proposed to enlarge the density of the neural network realization on FPGA hardware. The use of weights with values limited to powers-of-two has been proposed to avoid the use of multipliers [117]. Distributed arithmetic and lookup tables are applied to improve the efficiency of synapse multiplication [118]. Dynamic adaptive memories are proposed in order to reduce memory requirement [119]. Finally, a multiplexed architecture implementation was suggested, minimizing the resource requirement [120]. These techniques significantly reduce the cost of implementing neural networks of reasonably small size, but an efficient implementation of neural networks with a large number of neurons on FPGA remains a challenging task.

3.5.4 Graphics Processing Units

Graphics **P**rocessing **U**nits (GPUs) were originally designed to exploit parallel shared memory-based floating point computation applied to computer graphics. The single instruction, multiple data architecture (SIMD) of GPUs is gaining popularity because of their high performance, programmability, and price. Graphics processing units were primarily designed for high-performance rendering where repeated operations are common, due to that fact they outperform general purpose

CPUs in utilizing parallelism.

Many neural network algorithms have been re-implemented on GPUs. Among them are classification of the pixels of input images with the feature extraction and pattern recognition stage [121], and character recognition using a neural network based on the Izhikevich neuron model [122]. Also, a number of neural network simulators were created for execution on GPU. Among them are simulators based on spiking integrate-and-fire neuron model, however omitting axonal delays and synaptic learning [123], implementing Izhikevich neurons [124], and modelling both Hodgkin-Huxley and Izhikevich neuron models [125].

The implementation using GPU involves several issues. First of all, the developer encounters tight constraints, as a graphic card is mainly designed to render a scene and display it. Also, the programmer should master the fundamentals of the graphics shading languages that require the prior knowledge of computer graphics. Lastly, GPUs underperform when either a significant overhead in calculations is incurred or the algorithm is not sufficiently parallel.

3.5.5 Summary

We overviewed approaches based on custom shared memory solutions, cluster message passing computing systems, integrated circuits, programmable logic, and GPU-based computing. Applying SPMD by one-to-one placement of the neurons or group of neurons into physical components mostly suits our aim,taking into account the benefits of the architecture, as well as considering cost and time spent on development.

3.6 Existing Simulation Environments

Two solutions are available to neural network researchers for their experiment: either developing specific tools or using an already adapted simulator following a designated procedure. However, the problem of having a biologically realistic size and complexity in a neural network simulation has been underestimated for a long time. This reflected in the limited number of powerful neural network simulation environments used both in industry and academia.

It is difficult to make a proper comparison between different simulation environments because each is optimal for a given problem. It is interesting to note that many simulators are able to simulate the same models, allowing simulation results to be cross-checked between different simulators and providing greater confidence in their correctness. However, the codes are not always compatible with each other, making it difficult to incorporate the findings of others in a different model. This underlines the need for a more transparent communication channel between simulators.

Various common programming interfaces acquire popularity in order to reduce or eliminate the problem of simulator diversity. PyNN Python-based common simulator interface is one of them, already adapted by several large-scale simulation projects, such as NEURON, NEST, PCSIM, and BRIAN [126]. Nevertheless, many widely-used simulator tools still are not compatible with any common programming interface.

Below is presented an overview of presently most available simulation environments.

3.6.1 NEURON

NEURON is a highly reputable general purpose simulation environment for modelling both individual neurons and networks of neurons [18]. The program was introduced as a single-cell modelling domain but in the early 1990s it was enhanced and applied to network models containing large number of both computationally-efficient and biophysically-detailed neurons, or a combination of both. Later, it was extended for a distributed execution over parallel hardware. The simulator can run under Windows, Linux and Mac OS operational systems and is available free of charge.

The simulator is well-suited to conductance-based models of individual neurons with complex anatomical and biophysical properties, with extracellular potential near the membrane, and biophysical properties (i.e. multiple channel types, inhomogeneous channel distribution, ionic accumulation and diffusion, and second messengers).

NEURON offers several different numerical integration methods, such as Euler's method, the Crank-Nicolson method (providing higher accuracy at little additional cost), and IDA adaptive integration method (providing the highest accuracy at the cost of reduced runtime) [127]. The NEURON library is extensible through the NMODL and Hoc programming languages, which allow users to add new features to existing mechanisms (i.e. voltage- and ligand-gated ion channels, diffusion and buffering), as well as developing new descriptions for neuron models that have analytical solutions [128].

The simulator earned a reputation for having very friendly interface with a large number of tools for model construction (such as channel, cell, network and linear circuit builders), analyzing (such as Import3D, Model View, Impedance) and simulation control (such as Variable Step Control, Multiple Run Fitter). Besides, it has an extensive user base with more than 860 scientific articles and books reporting work that was done with NEURON.

3.6.2 GENESIS

The **GE**neral **NE**ural **SI**mulation **S**ystem (GENESIS) is a further example of a reputable general purpose simulation platform. It was the first broad scale modelling system in computational neuroscience. The last version (GENESIS version 2.3) is an open source program for UNIX-based systems.

The system was developed to simulate neural systems ranging from sub-cellular components and biochemical reactions to complex models of single neurons, simulations of large networks, and system-level models [19]. It is widely used both for single cell modelling (over 600 research publications) as well as for large network models (over 20 large-scale neural network GENESIS simulations were reported, many of which exceed 10,000 neurons per network).

GENESIS is an object-oriented simulation system with a "building block" approach. Basic building elements receive inputs, perform calculations and communicate by sending messages to each other. The message content differs according to the neuron's own variables and the method used to perform its calculations. A wide range of neuron models is built in into the program, from abstract integrate-and-fire model and the simplified Izhikevich model, to the biophysically realistic and plausible Hodgkin-Huxley model. Additionally, new user-defined GENESIS object types, or script language commands can be easily added, which is central to the generality and flexibility of the system. The simulator uses precompiled object types, rather than re-compiling all the scripts each time. This allows modifying a simulation while it is running without a significant change in speed.

The Parallel GENESIS (PGENESIS) was created as an extension to GENESIS. It runs on any parallel cluster, SMP, supercomputer, or network of workstations where MPI or PVM is supported, and on which serial GENESIS itself is able to run.

3.6.3 NEST

Neural **S**imulation **T**echnology (NEST) is a simulation framework for large, structured networks of neural systems, at the same time maintaining an appropriate degree of biological detail [129]. The main focus of this simulator is the dynamics behaviour of large networks, rather than the detailed morphological and biophysical properties of individual neurons [130]. It is a freely available software able to run on UNIX, Linux, Mac OS and Windows.

NEST is written in an object-oriented style in C++ and a network is regarded as a hierarchical structure, consisting on abstract atomic and compaund components. The simulator can be easily extended by adding new models of components (i.e. neurons and sub-neuronal compartments), or writing new libraries and routines. New components can be loaded during run-time as the NEST code is modularized.

The program package does not have any built-in graphical interface and data analysis happens off-line. The analysis can be done using the compatible mathematical tools, such as Matlab, Mathematica or Python.

The NEST simulation kernel supports parallelization by a POSIX multi-threading and message passing interface. On spike occurence, the pre-synaptic node simply notifies the post-synaptic node because the dynamics of synapses are placed on the post-synaptic side of the connections. In order to ensure causality, all communication is performed in intervals of the minimum propagation delay between neurons.

3.6.4 BRIAN

BRIAN is a clock-driven simulator for modelling networks consisting of spiking neurons, primarily aimed at modelling single compartmental model neurons. It is an open source Python extension package using vector-based computation. Despite the overheads of an interpreted language, vectorisation techniques allow efficient simulations.

In BRIAN all the internal variables of the simulator can be directly accessed in order to initialise the network or control it as it runs. Both short-term plasticity and STDP learning are included for integrate-and-fire (with adjustable threshold and reset) and Hodgkin-Huxley type neuron models. Linear, nonlinear, and stochastic neuron models can be easily defined by differential equations using standard mathematical notation. Runge-Kutta, linear and exponential (for non-linear models) Euler methods are implemented for numerical integration. The connectivity of a network can be specified either by directly describing connections of each pair of neurons, or applying all-to-all or random connectivity. Each of the synaptic connections can have a specific value of the propagation delay.

Brian includes several functions for spike train statistics. Python scientific libraries allow fast data analysis and processing with the NumPy and SciPy numerical and scientific computing packages and the PyLab graphics package, which mirrors the syntax of the Matlab plotting commands. There is a number of third-party packages available for graphic interfaces. The CherryPy package can be applied for creation HTML interfaces to Brian simulations, which can run locally or on a web server. Parts of the simulator can optionally be run using C code.

The popularity of Brian is due to the ease of learning and using the software and minimal development time necessary to construct a neural model. Also, the simulator does not require learning custom scripting languages (such as Hoc and NMODL for NEURON, NEST's SLI, and Genesis' SLI (the last two being different languages with the same name)). Parallel Python package can be used to run independent simulations with different parameter values on a cluster or on different processors.

3.6.5 NCS

The **N**eo**C**ortical **S**imulator (NCS) is a simulation platform optimized for modelling the horizontally dispersed and vertically layered distribution of neurons [131]. It performs both sequential and parallel simulation on Linux clusters (including Linux emulation) and on Mac OS systems.

NCS is written in C++ using object-oriented design principles. Each object can represent a cell, a compartment, a channel and the like, which model the corresponding cortical entities. The simulator employes integrate-and-fire and Hodgkin-Huxley neuron models using a clock-based approach [132]. Nonlinear specifications (like the Izhikevich model) are not supported.

Learning is implemented by using a look-up table, which contains values of synaptic weight modification (both positive and negative) based on the time between incoming and outcoming spikes occurence.

NCS delivers reports on any fraction of neural cell groups, at any specified interval. The report contains membrane voltages (current clump mode), currents (voltage clamp), spike-event-only timings (event-trigged), calcium concentration, synaptic dynamic parameter states and any Hodgkin-Huxley channel parameters. However, simulation does not provide any visualization software.

3.6.6 PCSIM

The **P**arallel **C**ircuit **SIM**ulator (CSIM) is a tool for simulating heterogeneous networks composed of spiking neurons [133]. The tool is written in C++ using object-oriented design and is compatible with Python, which provides data analysis and visualization. It is a freely distributed software able to run on Windows and Linux-like platforms. PCSIM can also be used as a package within Python.

PCSIM uses standard linear leaky integrate-and-fire and non-linear Izhikevich, as well as Hodgkin-Huxley models of a neuron. Also the framework simulates hybrid network models, which consist of both spiking and analog neural network components. Simulation is accelerated by dividing all synapses into the idle and active groups. Only the spike-receiving synapses are updated during simulation fixed time step. A synapse becomes idle again after its post-synaptic response has vanished.

The simulator has two forms of inputs, which are spike trains and analog signals. Correspondingly, communication consists on sending discrete and analogue messages (i.e. spikes, firing rates, or membrane voltages) between the connected neurons using either a multi-threaded approach, the MPI communication protocol, or both.

3.6.7 SPLIT

SPLIT is a simulation library specialized for efficient simulation of networks with a moderate number of multicompartmental models based on Hodgkin-Huxley formalism [134]. The model is written in the C++ language and supports parallelism using MPI.

SPLIT provides conductance-based synaptic interactions with short-term plasticity (facilitation and depression). Long-term plasticity (i.e. STDP) and integrate-and-fire types of neuron model are not implemented. It is a “general-purpose” simulator, where no specific topology or neuronal organisation is taken for granted beforehand.

A user needs to adapt the simulator for a certain model as the software should be regarded as a pure, generic neural simulation kernel. Nor does the simulator have any graphical interface nor any support for analysis of results.

3.6.8 HHSIM

Graphical **H**odgkin-**H**uxley **SIM**ulator (HHSIM) is a graphical simulation of a section of excitable neuronal membrane using the Hodgkin-Huxley equations. The simulator allows a full access to the Hodgkin-Huxley parameters, membrane parameters, stimulus parameters, and ion concentrations. It was specifically designed for teaching neurophysiology courses. HHSIM is a free software and is compatible with Windows, MAC OS, and Linux-like systems. It should be regarded as a generic neural simulation kernel with the user program adapting for a certain model.

Unlike NEURON and GENESIS, where a user explicitly controls the use of the underlying parallel system (i.e. manages communication between processors), the SPLIT simulator automatically makes the best possible use of the available hardware. In this way the user is shielded from the underlying computer system. At the moment the software does not have any graphical interface nor any built-in support for analysis of results.

3.6.9 MvaSpike

MvaSpike was designed as an event-based simulator for large, hierarchical or modular biological neural networks [135]. It consists of a core C++ library, is easily extensible and accessible from scripting languages, such as Python. MvaSpike is released under the GPL license.

A few common models of neuron are built-in, including linear and quadratic integrate-and-fire model with STDP, stochastic neurons, and phase-coded neurons. The simulator implements synaptic and axonal propagation delays, absolute refractory periods, Poisson input spike trains generation, and provides some connectivity patterns (both structured and randomly connected).

Partial parallel implementation and utilization of the XML structured data language for input and output data formatting are developed in the simulator. However, MvaSpike does not contain any graphical user interface, nor any pre- or post-processing tools.

3.6.10 SpikeNET

SpikeNET is an object-oriented neuron simulation program written in C++ [136]. It is another powerful framework for simulating large-scale networks of asynchronous spiking neurons, based on the integrate-and-fire neuron model. SpikeNET is available for public download under the GNU public license.

High simulation performance is achieved applying several techniques of computational efficiency. First of all, spikes are not propagated immediately but buffered in lists before being propagated. This greatly reduces the computations associated with large networks. This also simplifies distributed implementation as interprocessor communication can be limited to sending lists of the neurons, which just fired. Secondly, neurons are simulated with a limited number of parameters (i.e. classic properties like a membrane potential, a threshold and more novel features like dendritic sensitivity). They are organised in a two-dimensional arrays, which represent retinotopical homogenous maps and are the basic objects of the SpikeNET simulator. Thirdly, high simulation performance is achieved by weight sharing, which means that one set of weights can be used for all the neurons in an array.

Initially it was designed to investigate the biological plausibility of feed-forward processing using "at most one spike per neuron" scheme. Although the modern version of the simulator is a sophisticated simulation tool, it is still unable to process more than one spike per neuron. SpikeNET is able to simulate networks with millions of neurons and hundreds of millions of synaptic weights as long as the average spike discharge rate is low. This limitation of a low discharge rate is caused because of applying the event-driven computation approach, so that only the neurons that emit spikes are processed.

3.6.11 KInNeSS

KDE **In**tegrated **Neuron****S**imulation **S**oftware (KInNeSS) is an integrated environment for running neural simulations [137]. It uses Synchronous Artificial Neuronal Networks Distributed Runtime Algorithm (SANNDRA) as the core of all numeric simulations, which also provides distributed calculations. KInNeSS is built using object-oriented design in C++ and is an open source software for LINUX based systems.

It provides an expandable framework incorporating features such as ease of use, scalability, and an XML based schema. It is primarily aimed at modelling branched multi-compartmental neurons with biophysical properties like membrane potential, voltage-gated and ligand-gated channels, the presence of gap junctions or ionic diffusion, neuromodulation channel gating, the mechanism for habituative or depressive synapses, and axonal delays. The simulator implements STDP based only on local spatial and temporal information. This means that synaptic modification depends exclusively on the quantities present at the synapse and at the current time step. KInNeSS output data for later analysis in a variety of formats, such as MATLAB. The outputs include compartment membrane voltage, spikes, local-field potentials, and current source densities, as well as visualization of the behaviour of a simulated agent.

KInNeSS has a friendly point-and-click interface allowing the modeller to set all the necessary parameters and XML schema for both import and export of model specifications.

3.6.12 LENS

The **L**ight, **E**fficient **N**etwork **S**imulator (LENS) is a fast, flexible neural network simulation framework primarily designed for backpropagation networks [138]. Nevertheless it also supports deterministic Boltzmann machines and Kohonen networks and can be extended to other Hebbian or Bayesian models. It is written in C++ and Tcl using object-oriented design and new functions can be easily added to extend the standard functions. The simulator operates on both Unix and Windows platforms and is distributed free-of-charge for academic purposes.

The simulator supports feed-forward, simple recurrent, recurrent-backpropagation-through-time, and continuous recurrent-backpropagation-through-time (RBPTT) networks as well as deterministic Boltzmann machines and Kohonen networks. Other models of networks can be added with little effort. It implements five learning algorithms: steepest descent, momentum descent, "Doug's momentum descent", delta-bar-delta, and quick-prop. The simulator provides over 100 commands for building, training, and analyzing networks.

Lens supports batch-level parallel training on multiple machines, however the network itself is not partitioned among machines. This type of parallelism is typically used for searching for best network or training parameters in order to find the best performance.

3.6.13 JavaNNS

Java **N**eural **N**etwork **S**imulator (JavaNNS) is an efficient universal cross-platform simulator of neural networks [139]. It is written in Java and based on its predecessor Stuttgart Neural Network Simulator (SNNS) kernel (which was written in

ANSI C). JavaNNS has a new graphical user interface with a 2D and 3D graphical representation set on top of it.

JavaNNS provides event based simulation. It supports many network architectures, among them are self-organizing maps, dynamic learning vector quantiazation networks, radial basis function network, and time delay neural networks. Several learning procedures are implemented, such as the backpropagation, the counterpropagation, the QuickProp, and the resilient propagation.

In order to achieve a higher generalization performance by fewer free parameters, several pruning algorithms are able to reduce the number of weights or neurons of a network. Among them are magnitude based pruning (Mag), optimal brain damage (OBD), optimal brain surgeon (OBS), skeletonization (Skel), and non-contributing units techniques. It also has ENZO tool integrated into it, which allows optimizing the topology of neural network by means of genetic algorithms.

3.6.14 NeuroJet

NeuroJet is another neural network simulation program written in C++ using object-oriented design, which is freely available for Windows and Unix-like systems.

It was originally designed to proof a cognitive and behavioural concept via a biologically based, but still simplified, model of the hippocampal region of brain. This brain area raised an interest because of its key role in episodic learning. The current version is implemented to run on either a single computer or on a cluster. It can be easily extensible to incorporate any new models. At the current state the developers plan to design graphical user interface and make it compatible with mathematical packages, such as Matlab and Octave.

A parallel version of NeuroJet, originally known as PUNIT was designed to run on computer clusters. Later, the parallel implementation of NeuroJet was created on an NVIDIA GPU using CUDA API [140].

3.6.15 Nengo

Nengo is an open-source cross-platform software package for modelling large-scale neural systems [141]. It includes customizable models of spike generation, muscle dynamics, synaptic plasticity, and synaptic integration. Nengo is written in JAVA and uses Python script interface. It works on Mac OS, Linux and Windows systems.

Nengo provides standard and adapting leaky integrate-and-fire as well as the Hodgkin-Huxley model. The simulator regards a spike as a descrete event. Thus, it is not designed for neural models where the detailed voltage profile of a specific spike affects the post-synaptic neurons. Spike times, membrane voltages, and current can be recorded from the neurons. Variable-timestep integrator is applied, using the Dormand-Prince *4*th and *5*th order Runge-Kutta formulae.

Neural engineering framework (NEF) is used in order to encode and decode time-varying representations using spike train. It also is used to derive linearly optimal synaptic weights to transform and combine these representations. This framework represents a method for realizing a high-level description of neural models with adjustable degree of accuracy.

3.6.16 MOOSE

Multiscale **O**bject-**O**riented **S**imulation **E**nvironment (MOOSE) is a general biological neural network simulator [142]. It allows simulation from single molecules to neuronal networks.

Like the GENESIS simulator, MOOSE has a similar set of objects representing biological concepts (i.e. channels, molecules, compartments) and is backward compatible with it. MOOSE uses a Python-based scripting interface, which allows communicating among Python-aware simulators (i.e. simulating a single neuron in NEURON and an intracellular reaction in MOOSE). MOOSE is able to run in parallel on a cluster of machines using the MPI standard for communication.

3.6.17 FANN

The **F**ast **A**rtificial **N**eural **N**etwork (FANN) simulator is a free open source cross-platform library written in ANSI C [143].

The simulator implements multilayer artificial neural networks supporting fully-connected and sparsely-connected networks. It has fixed-point and floating point arithmetic for a bias neuron model with two activation functions (sigmoid and threshold). There is a choice of several types of backpropagation training, such as resilient backpropagation, quickprop, batch and incremental. It also has an option of evolving topology training, which dynamically builds and trains a neural network. For developing and analysing neural networks the simulator contains several graphic interfaces, such as FANNTool, NeuralView, FannExplorer, and sfann.

3.6.18 CX3D

Corte**x** simulation in **3D** (Cx3D) software is an open-source cross-platform software package written in Java. The simulator includes a wide range of phenomena related to growth and development of tissues and taking place in a 3-dimensional space. It is mainly applied to model large neural networks, where mechanical forces play a major role in development of nervous systems. The examples vary from simulation of a cortical folding to a neuron tube growth [144].

CX3D allows studying cell division, differentiation, migration, and extension of axonal and dendritic trees. Model components (such as spheres for somata and cylinders for neurites) are defined by creating small mechanistic modules, each having its exact space coordinate and represent all the local biological processes. The modules interact through mechanical forces of a direct contact and communicate by release of diffusible signaling molecules. The state of the components is recalculated during each time step and changes appropriately if the certain threshold is crossed.

A parallelized version of the Cx3Dp simulator scales with the number of available machines, both in speed and in the size of possible simulation.

3.6.19 Review Conclusions

Several mature simulators were presented in the previous section, which can simulate sophisticated neuron models and take advantages of distributed architectures with efficient algorithms. Each of above simulators has its own strengths and weaknesses and often is specialised in different application, having different optimisations. This allows choosing for the most appropriate one for a given modelling task. For example, BRIAN, PCSIM and NEST make use of a single compartmental models whereas NEURON, GENESIS, KInNeSS, and SPLIT also include functionality for creating multi-compartmental models. Other software focus primarily on dynamical systems analysis (i.e. XPPAUT), or model physical growth and development of tissues (i.e. CX3D).

Although many of reviewed simulation environments were initially designed for a specific purpose and domain of applicability, their set of features continually improves and expands due to persistent interest from groups of researchers and engineers.

A reasonable question to ask is whether there is any need for another neural network simulator. Usually off-the-shelf simulators are designed to be all-purpose feature-rich simulation tools, providing different simulation modes and techniques and allowing additional useful features, such as predefined templates, graphics management and data analysis. However, none of them has all the built-in facilities that could serve all existent purposes. It might be that currently available simulators do not fulfil the specific expectations of the user. Modifications can be necessary to allow different operations, such as combining reinforcement learning technique with Hebbian learning or implementing specific damage experiments and analysis procedures. The off-the-shelf software packages often are large programs with very complicated inner-working. Also the way the adapted simulation environments are structured does not allow modifying them straightforwardly. Readjusting such large pieces of code would have been a more time-consuming and error-prone process than writing a new specifically oriented tool.

Besides, the most desired features of a simulator are the ability to run a specific model (flexibility) in a reasonable amount of time (efficiency). However, efficiency is not only about the speed of simulations. The time spent on model implementation is at least as important in many situations. Many of the off-the-shelf simulators use custom scripting languages: Hoc and NMODL for NEURON, NEST's SLI, and GENESIS' SLI (the last two are different langauges with the same name). Mastering these scripts often extends the total time spent on simulation.

Therefore, it was decided at an early stage that it would be easier overall to make the necessary day-to-day changes to a smaller, custom-written program that contained only the facilities necessary for our experiments.

3.7 Summary

In this chapter we link neurophysiological measurements and theoretical studies, helping to capture some of the brain's computing capabilities. For this purpose we review popular today engineering solutions on the way to apply the current neurobiological knowledge to solve the technical problems, where conventional methods are inefficient (e.g. speech synthesis or computer vision).

In the beginning of the chapter we reviewed the basic neuron models, which are used in most neural network experiments. We outlined their advantages and disadvantages and selected the best candidates for large-scale network simulation.

Later, we overviewed the simulation strategies and techniques, used by engineers for solving technical problems, and highlighted an influence of the modelling techniques on the efficiency of network simulation. We discussed the potential contributions of a distributed architecture for simulations of neural networks. We also reviewed available architectures for distributed neural network simulation and emphasized the benefits and drawbacks of each of them.

Finally, we overviewed presently the most available simulation environments, which can simulate sophisticated neuron models and take advantages of distributed architectures with efficient algorithms.

Chapter 4

Sequential Neural Network Simulation

This chapter describes a designing procedure of a biologically-plausible network model, which is capable of searching for correct patterns, storing and reproducing them later. The biological learning processes found in the animal's brain are resembled. It is not the aim to build a realistic model of a brain, as at this time due to the lack of knowledge this task is impossible. *The first aim* is to understand the essential computations that take place in the network of interconnected neurons, resembling an actual biological neural network. *Another aim* is to construct a network designed to exploit potential of the *SpiNNaker* neuromorphic distributed computing machine gaining insights into the nature of neural computation [20].

Section 4.1 defines the network architecture used for simulation. Section 4.2 derives the employed learning mechanism, adapting the network weights based on a global binary feedback. In Section 4.3 the program is implemented, which allows a neural network to compute in ways traditionally associated with artificial neural networks. For this reason, a computationally efficient neuron model is applied, known as *Integrate-and-Fire* model. It does not include many kinds of known properties of neuron behaviour but helps to obtain an insight in the behaviour of the network at the learning stage. After it, the chapter investigates the network's performance and learning capacity dependence on the level of neuron interconnection.

As a case study, Section 4.6 presents an agent able to accumulate the vital experience from an ambient environment based on three proposed techniques. For this, we derive an algorithm that changes the weight of a neuron network by determining the exact error that the network makes on each example of a particular task in an unsupervised manner.

4.1 Network Model

This chapter defines precisely the model, its characteristics and its implementation challenges. Also this section reviews the network structure employed in the simulator.

4.1.1 Network Architecture

Various architectures have been constructed and applied to different network models. A structured network with a feed-forward signal propagation was chosen, resembling the visual cortex of a human brain (see 3.3.2.1). Input and output neurons are distinguished in the network from other neurons, belonging to the hidden layers. Each neuron operates according to the Integrate-and-Fire neuron model.

4.1.1.1 Connectivity

In biology, neurons have one of two types of synapse: excitatory, where the synapse releases neurotransmitter, which increases the membrane potential of a target cell, and inhibitory, which decreases the potential.

Similarly, every connection in the simulator has a certain strength (or *weight*), which is a real value and can be positive (for excitatory synapses) or negative (for inhibitory synapses). The weight is labelled $w_{i,j}$ for a direct connection from the axon of neuron j (pre-synaptic neuron) to the membrane or dendrites of neuron i (post-synaptic neuron). The weights can be shown as a matrix, visually representing a topology of connections.

It must be noted that synaptic connectivity of biological networks is sparse. This is caused by the fact that almost all synapses of the axon of one neuron connect to different neurons. This means that only a certain group of firing neurons can cause a fire of a post-synaptic neuron, as probably no one neuron does have a large influence on any other neuron (except certain special areas of the brain, like the cerebellar cortex). This fact is important in order to maintain low firing activity of a neuron network.

4.1.1.2 Introducing Dilution

Neurons in biological networks are connected asymetrically by *one-way* connections. It means that an axon of pre-synaptic neuron i can connect to a dendrite or

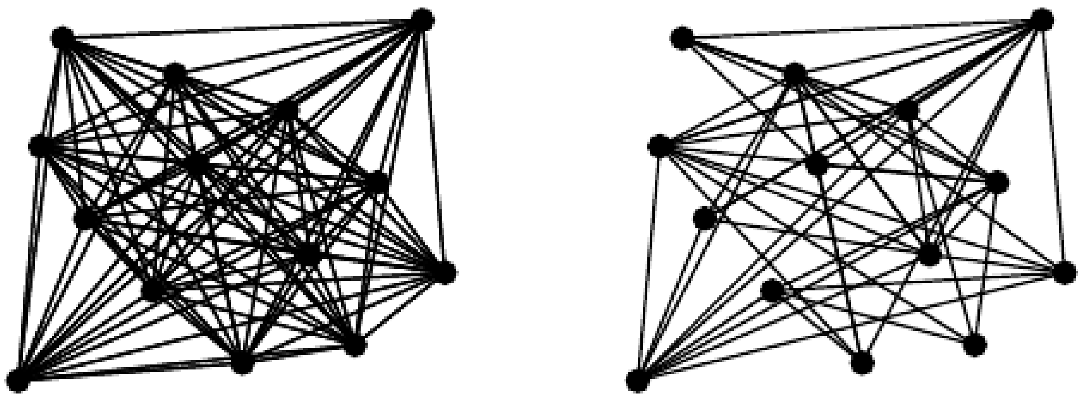


FIGURE 4.1: The effect of dilution: fully connected network (*left*) and with some missing connections (*right*).

cell body of a post-synaptic neuron j by a connection J_{ij}, while at the same time the axon of a post-synaptic neuron can connect to the cell body or a dendrite of neuron i by a connection J_{ji}, so that $J_{ij} \neq J_{ji}$. Notably, both connections have unequal weights.

It also turns out that not all neurons are connected to each other in biological neural networks. The human brain, for instance, consists of 10^{11} neurons with each neuron connecting to about 10^4 others [145]. In order to represent this phenomenon *dilution* is introduced, i.e., the random breaking of connections between neurons. In the simulation a partially connected (or *diluted*) three-layer network is used (although there is a possibility to vary the number of hidden layers). The dilution d represents whether a connectivity of the network is dense or sparse:

$$d = \frac{M}{N(N-1)} \tag{4.1}$$

where N represents the total number of neurons and M is the number of missing (or 'broken') connections.

4.1.1.3 Diluted Network Importance

There are several reasons why the investigation of diluted networks is worth attention.

Firstly, as it was mentioned, this guarantees closer resemblance to the cortical areas associated with vision [146].

Secondly, and the most importantly for our research, the reduced number of connections reduces the communication load on the network as well as the computational speed. Particularly it is beneficial for distributed hardware implementation.

Thirdly, dilution is an attractive tool to investigate the fault-tolerance and robustness of the cooperative behaviour of neural networks against malfunctioning of some of its elements. This reduces a probability of system break down if only a few elements deteriorate in their performance. This aspect is vitally important for parallel hardware architectures and is often advocate the superiority of neural network architectures over more traditional sequential computing systems.

Several types of diluted network are widely used, depending on the procedure used for breaking up neuron connections in the fully connected network [147].

In our research we interconnect neurons based on a certain probability. This diminishes the chance of involving the same neurons forming the activity paths for different input patterns. However, the level of activity depends strongly on the level of dilution and our investigation shows that the most optimal level of dilution would be 80 % of the full network interconnection (see Fig. 4.12 in 4.5).

4.1.1.4 Network Structure

A subset of the neurons of a network is chosen to be input neurons. In a biological organism they would get their input from one of the sensory systems. They can be triggered to fire in reaction to outside events, like light falling on a particular sensor cell in the eye. As the input neurons get their input only from a sensory system, they do not get input from other neurons in the neural network.

Another subset of the neurons can be distinguished as an output of the network. Their axons are designed to output the signal and therefore are not connected to other neurons in the network. In a biological organism this could represent motor neurons with long axons connected to cells of muscle fibres of the body.

The input to a neural network is provided by setting the activity of a number of neurons defined as input neurons according to a certain input pattern. An input pattern consists of a set of binary numbers (0 and 1). In the same way the output pattern is read from the subset of the output neurons. The length of input and output patterns cannot be higher than the number of the corresponding subset of neurons, but it can be lower, in which case the rest of neurons are assumed to have activity equal to zero.

All other neurons in the network are grouped to one or more hidden layers. Their output is connected either to other hidden neurons of other layers, if any, or to the output neurons.

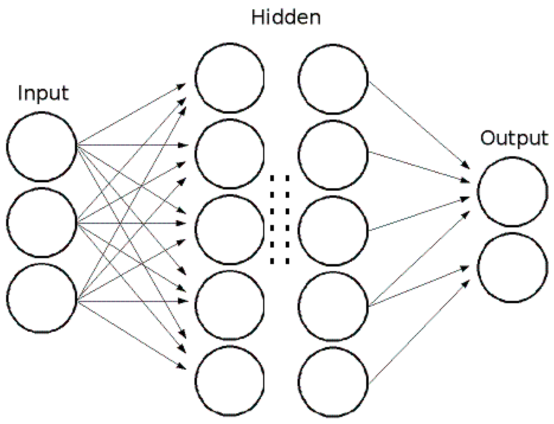


FIGURE 4.2: Example of different subsets of neurons in the network.

We deliberately avoid the direct connections between the neurons' set dedicated for the input with the neurons responsible for the output. Otherwise, the direct connections extend the learning period or corrupt the final result. It happens because these connections cannot fit all set of learning patterns and if they do not fit the current learning pattern, it takes a long time to reduce such connections' weight to zero. Besides, according to the learning mechanism, all other weights are affected by the changes.

The activity signals propagate in a feed-forward mode from an input layer to a hidden layer and from the hidden layer to another hidden layer, if any, or to an output layer. Feed-forward mode means that every neuron in layer i has only connections to neurons in layer j, if $j = i + 1$ (in broader meaning $j > i$) (refer to 3.3.2.1).

4.1.2 Neuron Model

The neuron model describes the neuron behaviour in response to input stimuli and the process of producing output spikes.

4.1.2.1 Rationale for Chosen Model

The neuron model must be optimal in terms of balance between a computational load and biological conformity. We use a simplified but computationally efficient neuron model, known as *Integrate-and-Fire*, which does not include many properties of neuron behaviour.

Using a more complicated neuron model at this stage could obstruct understanding of which mechanism causes the observed behaviour of the network under research.

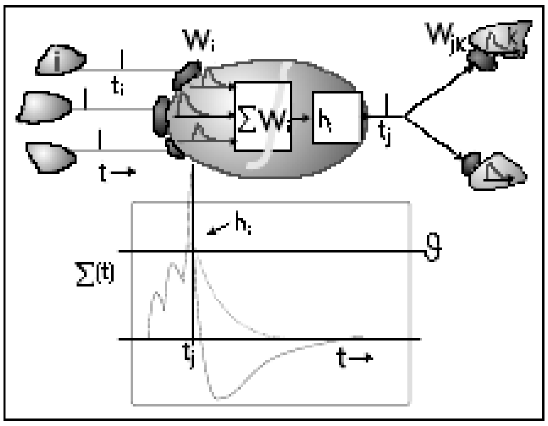


Figure 4.3: Neuron model parameters.

On the other hand, by limiting the important properties some of computational capabilities of the neuron model (STDP as an example) are ignored. However, it helps to understand what role these properties perform in the learning process and how much they influence the learning speed and capacity of the network. The simulator is able to simulate several neuron models. But for the research purposes the integrate-and-fire model is employed, which was described in section 3.2.2, because it is comparatively easy to keep track on the neuron's execution and backtrace the possible bugs.

Remarkably, networks, which are using such a simple neuron model, are able to implement a range of tasks relating input states to output states and are a convenient tool for investigating the dependence of network dynamics and activity on various network parameters.

4.1.2.2 Model Parameters

In the Integrate-and-Fire model, an internal state of any neuron i is described by the following three variables:

- a binary state variable x_i, describing whether neuron i is active and firing ($x_i = 1$) or inactive and staying at rest ($x_i = 0$).
- a variable membrane potential h_i, representing the sum of the states of the pre-synaptical neurons multiplied by the weights of their connections.
- a fixed threshold potential θ_i; a neuron becomes active when the neuron's potential h_i exceeds the threshold potential θ_i.

4.1.2.3 Model Dynamics

The state variable is calculated using the relationship between h_i, θ_i according to the equation:

$$x_i = \Theta_H(h_i - \theta_i) \tag{4.2}$$

where Θ_H is the Heaviside stepfunction, defined by $\Theta_H(x) = 0$ if $x < 0$ and $\Theta_H(x) = 1$ if $x > 0$. If the weighted sum of the states of the neurons connected to neuron i exceeds threshold θ, the state of the neuron is set to *active*, otherwise it is not.

The potential h_i of a neuron is subject to change if a neuron receives some input via its synapses from other neurons. It is determined by the sum of the states of the neurons connecting to neuron i and the weights of these connections:

$$h_i = \sum_{j \epsilon V_i} w_{ij} x_j \tag{4.3}$$

where V_i is the collection of all neurons that have an afferent synaptic connection to neuron i. Excitatory synaptic connection (EPSP) increases the membrane potential whereas inhibitory input (IPSP) decreases the membrane potential, as shown in Fig. 4.4. If enough excitatory input accumulates and reaches the threshold level, a spike is emitted. The membrane potential returns to its resting level.

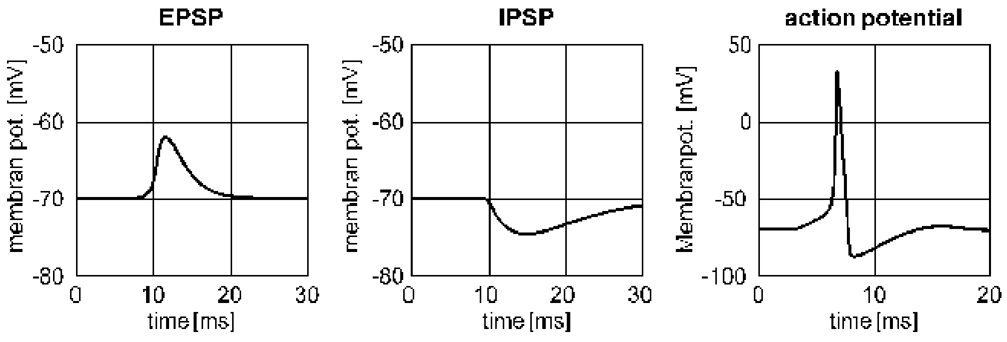


FIGURE 4.4: Influence of excitatory (EPSP) and inhibitory (IPSP) synapses on the membrane potential, resulting in a spike (an action potential) if the membrane potential crosses the threshold level [9].

Other more complicated neuron models, such as *(a)* using continuous state variables, representing the frequency of firing either *(b)* including some stochastics by introducing a probability distribution for a neuron to become active or *(c)* using other transfer-functions than Heaviside stepfunction for state variable calculation (sigmoidal, as example) or *(d)* multiplying the state variables of pre-synaptic neurons for potential h_i calculation, are also compatible with the simulator.

4.2 Learning

A lot is unknown nowadays about the behaviour of a biological neural network and the mechanisms of learning at the level of neurons. To clarify the task, this section explains what is actually ment by learning and introduces some basic principles. After that, it presents the learning paradigms and reviews some possible learning mechanisms. Finally, it discusses the way the network evolves over time by changing the weights and in this way provides a learning process.

4.2.1 What is Learning?

By learning we mean that the probability for certain behaviour to happen in reaction to a certain event is altered. Such an alteration either means that some reaction is likely to be repeated, or that the probability for some reaction to occur is lowered.

4.2.2 Biological Inspiration

For inspiration, we refer to the Morris water maze, devised by R. Morris nearly 30 years ago [148]. The maze was designed as a method to assess spatial or place learning. It has been proven that the test strongly correlates with hippocampal synaptic plasticity and glutamate receptor function [149].

4.2.2.1 Morris Water Maze

During the test a rat is immersed into a water tank with a platform located somewhere near the centre. The water contains skim milk powder to stop the rat from seeing through the water. During the training phase the platform is lifted above the water so the rat could see it and find its way out of water, as shown in Fig. 4.5. On subsequent trials with the platform submerged in the same position (although being invisible to the rat), the rat is able to locate the platform increasingly rapidly. After enough practice, a capable rat swims directly from any release point to the platform. Such an improvement occurs as a result of learning and memory for where the hidden platform is located.

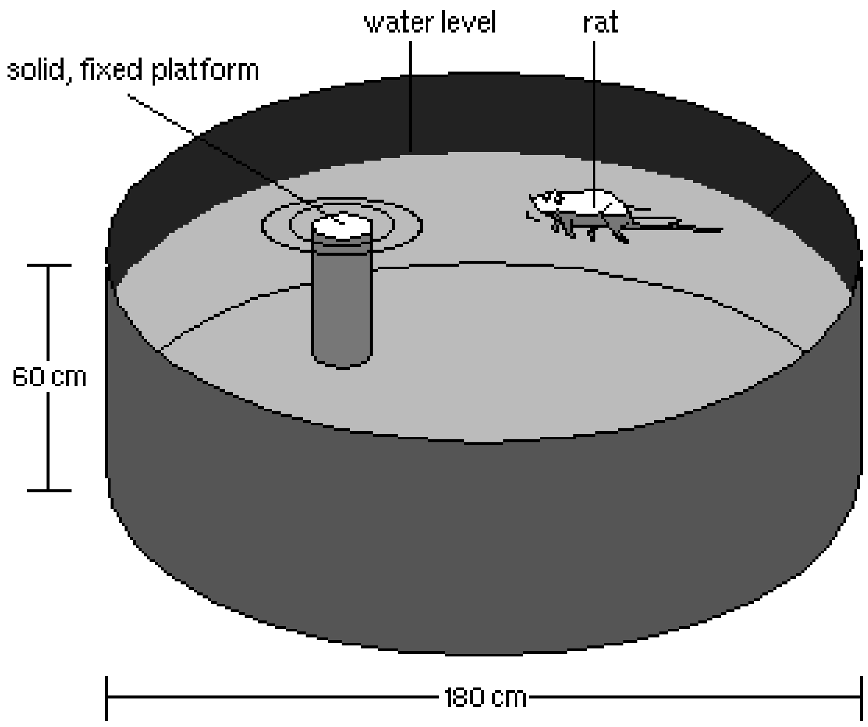


FIGURE 4.5: Morris water maze.

In the test finding the submerged platform leads to some sort of satisfaction or safety feeling, which we correspond with reinforcement of knowledge. When a learning phase of the test is repeated with the platform being in a new location, the rat heads to the direction of the new platform position, forgetting about the old position of platform (which we correspond with deinforcement of knowledge).

4.2.2.2 Representation of Biological Learning

Obviously, some kind of memory is involved in the process of learning. On the basis of biologically motivated assumptions it is found that the strength of the synaptic connections between neurons can change for some time or more permanently through two possible learning processes: LTP and LTD, discussed in sections 2.3.1.1 and 2.3.1.2. It is likely that these are the underlying mechanisms of memory and learning, if a correct (or satisfactory) output has been found in reaction to a certain input pattern.

LTP fixes and strengthens the state of the network at the moment it is applied, while LTD destabilises the state of the network, and, when applied repeatedly, changes the network output when the effect of the found output is not satisfactory. In this way the system changes its behaviour by changing its output signals following a certain input and searching for a correct output occurs. In other words, LTP should be applied if the network realises the desired output state in reaction to its input, while LTD should be applied when the output of the network is wrong and the network should search for a better output.

In the same way our neural network gets information to deinforce or reinforce the current behaviour through some kind of feedback, which represents the 'feeling' of (un)happiness. Every time when a pattern is presented to the network, a learning rule is applied in order to memorise the pattern. It is trained to recognise input-output patterns by adapting its synaptic connections, given by w_{ij}, by certain learning rule. One of this rules (LTP) memorizes input-output relations and takes place when the output is right, while another (LTD) does the opposite: it changes the input-output relations of the network and takes place when the output is wrong.

Another possibility, often used to model memory, is to instruct the network exactly what output it should generate for an input (known as *learning with a teacher*). In such a way a network is not able to solve any problem by itself. It will only imitate some desired behaviour, which was already determined in advance. Therefore we are not interested in such learning method and leave it outside of the scope of our research.

4.2.3 Learning Mechanism

This subsection describes how learning is realized by changing the weights w_{ij} of the neural network and discusses about what actually the neural network should learn.

4.2.3.1 Existing Techniques

Various techniques are widely used for association of input-output patterns. Some of the most well-known techniques are perceptron learning rule, backpropagation and Boltzman machines [150, 151].

Despite being popular, however, these techniques violate the existing biological limitation of *local* update rule, in this way abandoning the biological nature of the learning mechanism. They are not realizable just with the help of processes occurring in nature, learning does not occur through an adaptation of synaptic strength while depending purely on the activity of the involved neurons. Instead, for instance, in the backpropagation algorithm the update rule involves the back-propagation of a distant error signal, potentially computed many layers above it. Therefore we avoid using them for our model.

It is necessary to note that a learning process, based purely on *locally* specific information, is more difficult and slow comparing to non-biologically realizable algorithms.

4.2.3.2 Hebbian Postulate

Formulated as early as 1949, Hebbian rule [152] has been an important milestone for both neurophysiology and computer science. It was the first and the only plausible learning rule for artificial neuron networks. The rule was successfully used in various applications, including the model of bees foraging in an uncertain environment [153] and human decision making [154].

More than one neuron is needed to excite a postsynaptic neuron. Although this point is mentioned in the Hebbian postulate [155], often it is violated. This restriction limits the probability of any neuron to influence several activity paths and thus reduces the chance of their overlapping.

While applying the method in its present form, an autocorrelation term of the learning rule stands out. It is caused by correlation of presynaptic and postsynaptic neurons' activity: weight growth leads to a higher postsynaptic potential and therefore even more weight growth. It causes the exponential weight growth and leads to destabilisation of the network [156]. Although in an earlier paper [157] Hebb had introduced the mechanism of decreasing the synaptic weight under certain conditions, he has excluded it in its final version [155]. Consequently, it is unlikely that applying only the pure Hebbian learning will result in an entirely adaptive system capable of task-oriented learning.

4.2.3.3 Rosenblatt Principle

In 1962, Rosenblatt developed the first computer that could learn new skills by trial and error and proved [77] that a neuron network is able to associate input-output relations, if a finite number of times the weights are adopted accordingly to the following rule:

$$w_{ij}(t_{n+1}) = w_{ij}(t_n) + \Delta w_{ij}(t_n) \tag{4.4}$$

In other words, the equations tells us that with each time step the quantity of weight w_{ij} is corrected by a certain portion Δw_{ij}. Being constant, this important parameter directly influences the speed of the learning process. In a case of a very small value, the weight adaptation process drastically slows down and learning may take extremely long time. Application of large values of Δw_{ij} destabilizes the learning process because after finding the optimal weight values, large change in weight may significantly reshape the weight matrix, destroying the learned patterns. Rosenblatt suggested using the variable parameter, adapted to the level of produced error:

$$\Delta w_{ij} = \varepsilon(x_{Ti} - x_{Oi})x_j \tag{4.5}$$

where x_{Ti} is desired output or target output of neuron i, and x_{Oi} is its actual output. Furthermore, x_j is the state of the pre-synaptic input neuron j and ε is some function of the neuron states and properties of neurons i and j.

4.2.3.4 Weight Update Rule

The important limitation of the Rosenblatt rule is that the knowledge about the states and properties must be local at the involved neurons and a synapse, connecting them. In other words, the value Δw_{ij} cannot depend on variables local to neurons other than the neurons i and j. By satisfying the requirements of the local update rule, learning corresponds to the processes happening in the biologically plausible network.

To solve the described constraints without loosing the biological plausibility, i.e. keeping it consistent with learning processes of the animal's brain, Bosman et al. suggested a model of neural network in which both Hebbian and reinforcement learning occur [10].

The *reinforcement learning* happens when a network is searching for the best solution on its own by trying different possibilities and getting feedback on how well it performs instead of being instructed somehow what output to generate for a certaing input [158, 159]. On the basis of trial and error, a neural network learns to associate the provided patterns according to the principle of reinforcement learning. Natural interpretation of the principle is rewarding the satisfactory neuron network outputs (with $r = 1$), causing them to occur more frequently and punishing the incorrect outputs, which produce the feedback $r = 0$.

According to the contemporary biological studies, reinforcement mechanism was observed in biological learning processes and was successfully used to interpret the activity of dopamine neurons to mediate reward-processing and reward-dependent learning in non-human primates [160], and modulate cortico-striatal synaptic efficacy in humans [161], also solving problems, including robot control, elevator sheduling, telecommunications and chess [162].

However, in his "minibrain" model Bosman applies the extremal ("winner-takes-all") dynamics, inspired both by earlier self-organized critical models [163] and the self-organized map [164]. This considerably improves learning performance and provides a fast and highly adaptive learning system. The idea is based on artificial control of the firing neuron number by allowing firing exactly the previously set number with the highest membrane potential. The neurons, whose potential is above the threshold potential will not fire if there are enough of other neurons with higher potential, and the neurons with potentials below the threshold value could be forced to fire if their potential is the highest. A similar idea was researched by J. Bedaux and W. van Leeuwen in [165]. This external regulation of activity works well enough, however the biological property of threshold potential is neglected and for this reason we do not apply such a regulation in our model. Another drawback of extremal dynamics is a significant overload of the computational resources.

To determine Δw_{ij} we apply the learning mechanism modified by J. Bedaux and W. van Leeuwen in [165], however we ignore the extremal dynamics component:

$$\Delta w_{ij} = r\eta_i[k(2x_i - 1) - (h_i - \theta_i)]x_j - (1 - r)\rho_i(x_i - \alpha_i)x_j \tag{4.6}$$

where η, k, α and ρ are coefficients and r represents the binary feedback signal, representing 'success' or 'failure' of a provided output after each attempt to associate the correct output with the given input.

4.3 Program Implementation

This section describes the implementation of the described algorithm for neural network learning input-output patterns.

4.3.1 Initialisation Phase

The algorithm of learning input-output patterns has been developed and implemented. The structure of program's initialisation stage is given in Fig. 4.6. A choice of pre-defined and random input-output patterns has been implemented. In the case of pre-defined patterns, they are stored in an external file and read

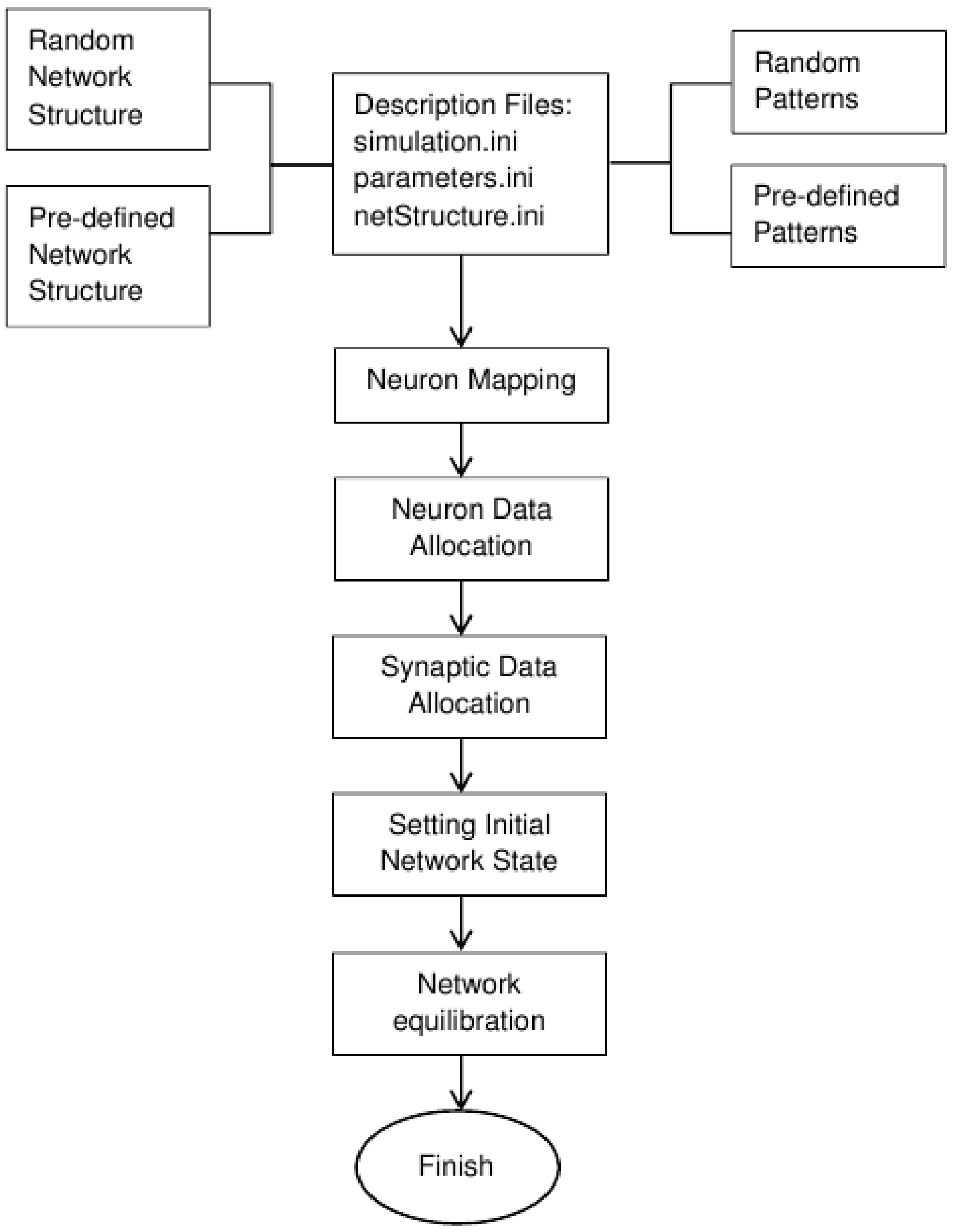


FIGURE 4.6: Simulation initialization stage.

by a program during simulation. The network can be either generated according to supplied rules or read from an external file, which may contain the state of the network from a previous simulation. The supplied rules and parameters are stored in description files, which contain initial parameters required to built the neural network and set up the learning rules. They are read by a program before simulation starts.

Some of the parameters are specified in the way to pass the range of the values. In this case the first and the last value of the vector are supplied as well as the number of values contained in this range. If the set of the numbers cannot be divided into the equal intervals, the rounding applies and intervals slightly deviate in the term of the size.

4.3.2 Algorithm

The entire design of the neural network is object-oriented. The building blocks of the system are **Network**, **Neuron** and **Synapse**, as presented in Fig. 4.7. Each of these classes stores only a minimum amount of information that is required for representing its dynamic state. An object with lots of parameters requires large memory allocation to store these parameters, which results in memory overload and in a high computation time.

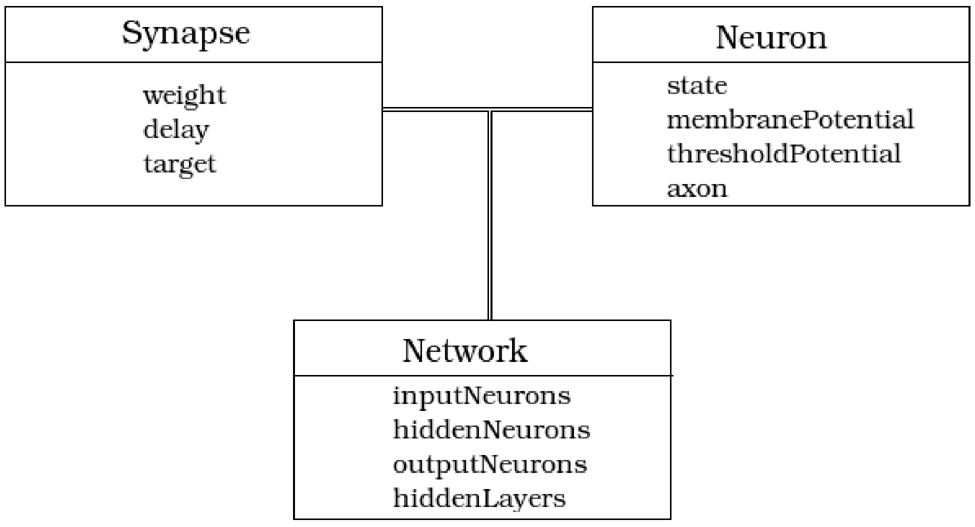


FIGURE 4.7: Simplified representation of main program classes.

The program operates as represented in Algorithm 4.1. After the certain input pattern is fed to the simulation program, the program tries possible output patterns as a reaction to an input pattern for *maxSearchAttempt* times. As long as the output pattern is not the desired one, the LTD rule is applied to the network. Once the correct output has been found, the input-output relation should be memorised by applying LTP rule. In this case it will be recalled a next time after the network learns some other relations. The learning of all input-output patterns is attempted. When all patterns are learnt once, the relations are shuffled and the learning cycle is repeated until all output patterns are recalled at the first time step after presenting the corresponding input. At this point we assume the network has successfully learned all patterns and the network state can be saved in an external file.

The program "knows" about the output correlation by the binary feedback. If the value of the feedback is 0, the output does not correspond to the input. If the value is 1, the output corresponds to the input according to the input-output relation.

```
begin
   for i = 0 to patternNumber do
      for j = 0 to maxSearchAttempt do
         reset Potentials();
         inputPattern[i];
         propagateActivity();
         if (simulationOutput != expectedOutput)
            applyLTD();
         if (simulationOutput == expectedOutput)
            applyLTP();
      shufflePatterns();
end
```

Algorithm 4.1: One cycle of simulation

In our simulation time is discretized, which means that the states of the neurons are only determined at the moments $t_n, t_{n+1}, ...$, with $t_{n+1} = t_n + \Delta t$. In each time advancement Δt the states of neurons are updated. In each update the neurons influence the post-synaptic neurons: increasing their potential if a connection is excitatory and decreasing the potential if the connection is inhibitory.

In this way some of post-synaptic neurons can become active, if their potential exceeds the threshold potential. Or the post-synaptic neuron can become inactive, if the influence of inhibitory connections is high enough to lower the value of potential below the threshold potential.

4.4 Model Weaknesses

Our approach resembles an actual biological neural network and the way it might learn. Biological resemblance embraces a neuron network's unassisted search process aimed at finding an appropriate set of synaptic weights through the potentiation and depression of the synaptic weights based on the local synaptic update rule. We created a network capable of learning new information without losing the old at the same time. However, the network's learning capacity remains the crucial issue.

The interference of new learned data with previously learned data occurs while storing the large amount of input patterns into the network. A self-destruction of a pre-learned information in an attempt to adjust the weights for storing new information is a common problem.

In the canonical example of [166], Edelman compares a mouse's behaviour with the behaviour of a robot, controlled by a neural network in a certain environment. The result shows that after the environment changes, the robot seldom retrieves pre-learned knowledge, unlike the mouse. The reason is that the field of coefficients of neural network is destroyed by the newly learned information. This example presents the importance of storing a pre-learned data while being able to perceive the new data without any overlap.

4.4.1 Active Paths Interference

When the new pattern is learned, additional paths of activity are formed. To preserve the old learned patterns, the new patterns should not influence and change the already existing paths too much. This can be accomplished by keeping the activity of the neurons low, as experiments have shown [10], [167]. The neuron activity is defined as the ratio of the firing neurons to the overall neuron number in a certain fraction of time. In this way it is possible to calculate the average activity over a certain period of time.

Modifications of the synapses is a very powerful mechanism that allows shaping and modifying the response properties of a neuron. Poorly regulated neural network's activity level can grow and shrink in an unpredictable manner. The neuron activity can present excessively high or low firing rates. Therefore unless synaptic strength changes are coordinated appropriately, such a mechanism can be very dangerous for the network stability.

4.4.2 Reasons of Active Paths Interference

The negative effect of the paths interference arises with more input patterns are applied to the neural network to be learned. The active paths overlap when the strongest connection from the different input patterns point to the same intermediary neurons. As result, the learning of something new causes forgetting of an old data.

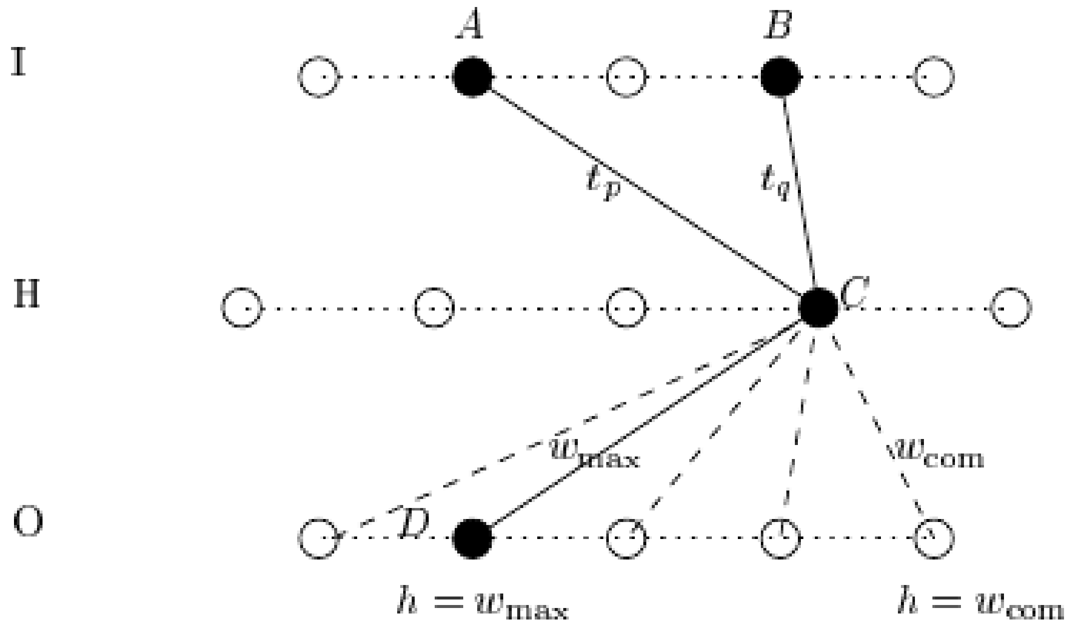


FIGURE 4.8: Path interference of A-C-D and A-B-D neuron paths [10].

There may be several reasons, which give rise to such a situation. First of all, from the active input neuron the path of activity runs along the strongest synaptic connections to the corresponding output neurons. In certain situations an established path can be completely “wiped out” by an attempt to learn new data, so that connection of the previously learned pattern is no longer the strongest. Also the competition between the activity path, formed in previous steps, and newly forming active path can happen. Such a competition often erases or partially destroys the old path and correspondingly leads to forgetting of old data by the network.

Secondly, according to the algorithm, in the case of the incorrect ouput pattern, the mechanism of decreasing the strength of recently formed synaptical connections is applied. This almost certainly removes the previously formed synaptic connection from the active level of synapses and forms another one, which produces a different output pattern. Correspondingly, a large number of incorrect output patterns cause a large change in the geometry of the active level and, hence, in the weight matrix of the network connections. This causes a so-called “avalanche” in the network landscape [168].

Bienenstock, Cooper and Munro suggested using the flexible threshold [40] as a mechanism of global neuron activity regulation. Correlated pre-synaptic and post-synaptic activity evokes LTP if the post-synaptic firing rate is above the threshold value and LTD when it is below. The threshold is not a constant value, but a function of the average post-synaptic firing rate. When the post-synaptic firing rate is highly active, the threshold increases. This mechanism reasonably regulates the activity of the neurons, stabilizing the overall activity of the network.

Similar effect can be achieved by keeping the overall network activity at low level. The formation of new network activity patterns has less influence to the already existing paths, allowing both to coexist [167].

4.5 Simulation Results and Discussions

To measure the efficiency of the learning process we selected two parameters: the number of learning steps and learning performance. The latter can be calculated as the ratio of the actual number of steps, which were needed to learn the set of applied input patterns and to associate them with the output patterns to the total number of performed learning steps.

According to the proposed approach we created a neural network of 500 neurons and studied it under various conditions. We dedicated 33 neurons to input and 10 neurons to output and taught the network for 20 different input patterns, associating them with certain non-repeating output patterns chosen by the network. Every experiment was repeated 100000 times to calculate the mean value of the learning efficiency.

Fig. 4.9 shows the performance of the neural network as a function of the number of input patterns. The performance falls with the higher number of applied patterns. The learning performance is almost equal to 1 for one or two input patterns, but drops to 0.05 for 16 input patterns.

Similar behaviour is observed by counting the number of learning steps in order to find the capacity of the network to learn. The idea is to increase the number of patterns to be learned until the system is no longer capable of learning. We used two networks of 500-neurons and 2000-neurons sizes with various connectivity patterns. Fig. 4.10 shows a comparison of the average number of learning steps performed by the neural networks as a function of the number of input patterns.

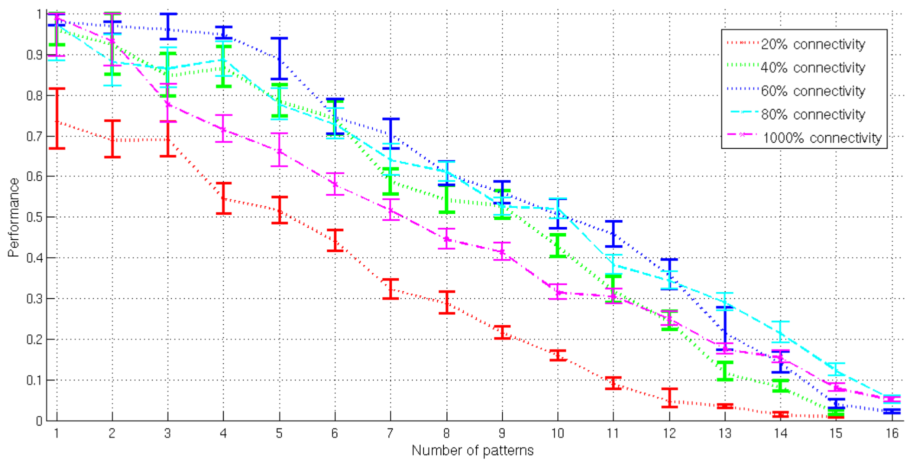


FIGURE 4.9: The performance of the neural network of 200 neurons while increasing the number of patterns to be learned.

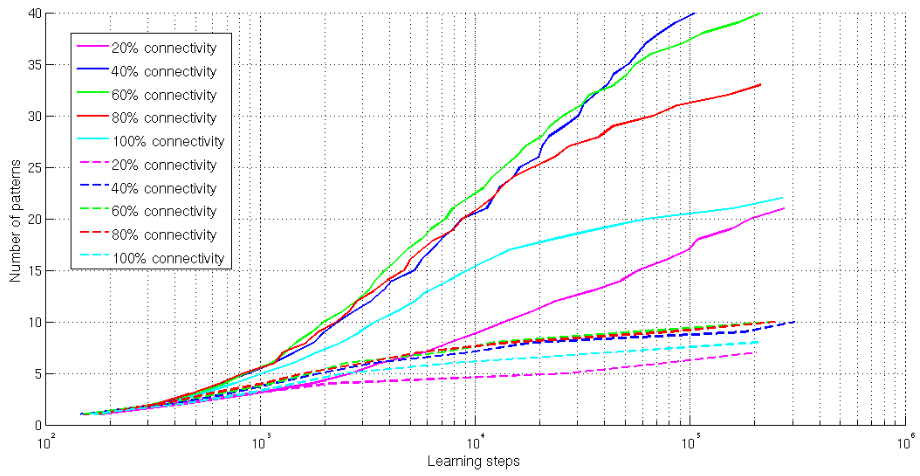


FIGURE 4.10: The capacity of the network to learn as the number of patterns increases until the system is no longer capable of learning. The solid lines represent the network of 2000 neurons and the dashed lines represent the network of 200 neurons.

The 2000-neuron network is able to learn a higher number of patterns and presents the best capacity at connectivity around 40 % and 60 %.

Fig. 4.11 shows the dependence of the number of learning steps on faults. The red line represents the dependence on the faulty nodes and the green line represents the dependence on the faulty synapses. The level of faults is represented in percents rather than by the actual number of faults as this provides a better representation of the scale of damage done to the network. The dependence is almost linear up until 0.3 % for the network with damaged nodes and up until 0.25 % for the

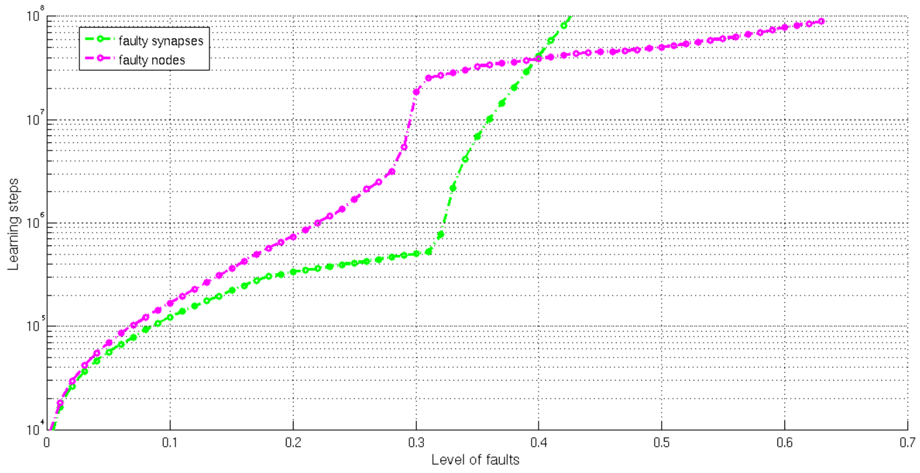


FIGURE 4.11: Dependence of the number of learning steps required to learn 20 neurons by 500-neuron size network on the faults number injected at the network level during the learning process. The level of faults is presented in percents.

network with damaged synapses. The faulty nodes reduce the learning speed more significantly comparing to the faulty synapses when the number of faults is small. However, this changes to the opposite after the level of faults reaches 0.4 %, when the level of damage caused by faulty synapses becomes larger than by the faulty nodes. The value 0.4 % represent the point when the number of faulty synapses introduced to the network is equal to the number of faulty synapses caused by the faulty nodes.

We empirically investigated how the level of connectivity influences learning. We examined its dependence on the level of connectivity by gradually reducing the connectivity of the network and measuring the required number of learning steps.

Fig. 4.12 shows that the range of the most optimal performance is located in the range of 45 % - 75 % of connectivity with the peak located at 55 %. The low interconnection level as well as nearly 100 % connectivity leads to longer adaptation time.

We took a closer look at the dynamics of the synaptic weight changes of the network. Fig. 4.13 shows the rate of weight change of a randomly taken connection. It presents how a neuron participates in the activity path formation and keeps adjusting its value during the whole process of the experiment. The fluctuations shown in Fig. 4.13 are caused by adapting the synaptic weight while opposing the effect of path interferences and are also decreasing as the network redistributes the weights and forms alternative paths to associate the input patterns with the

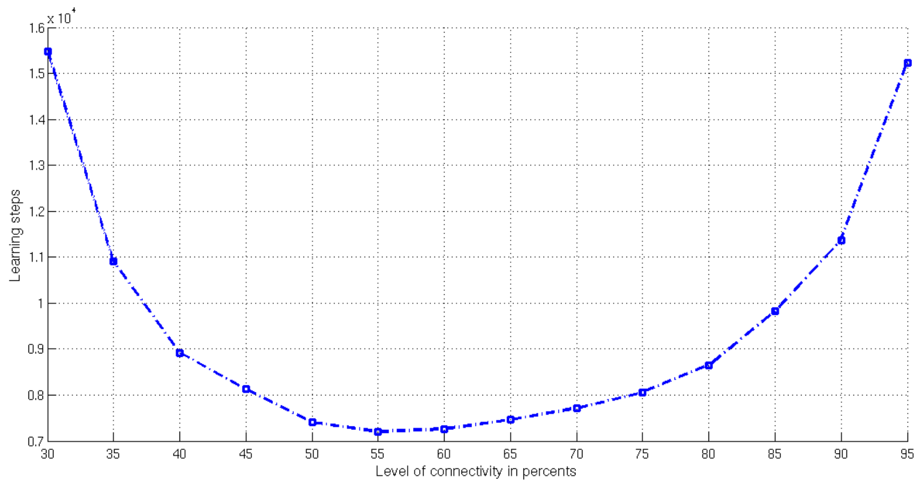


FIGURE 4.12: The number of learning steps requried to learn 20 different input patterns depending on the level of connectivity in the network of 500 neurons.

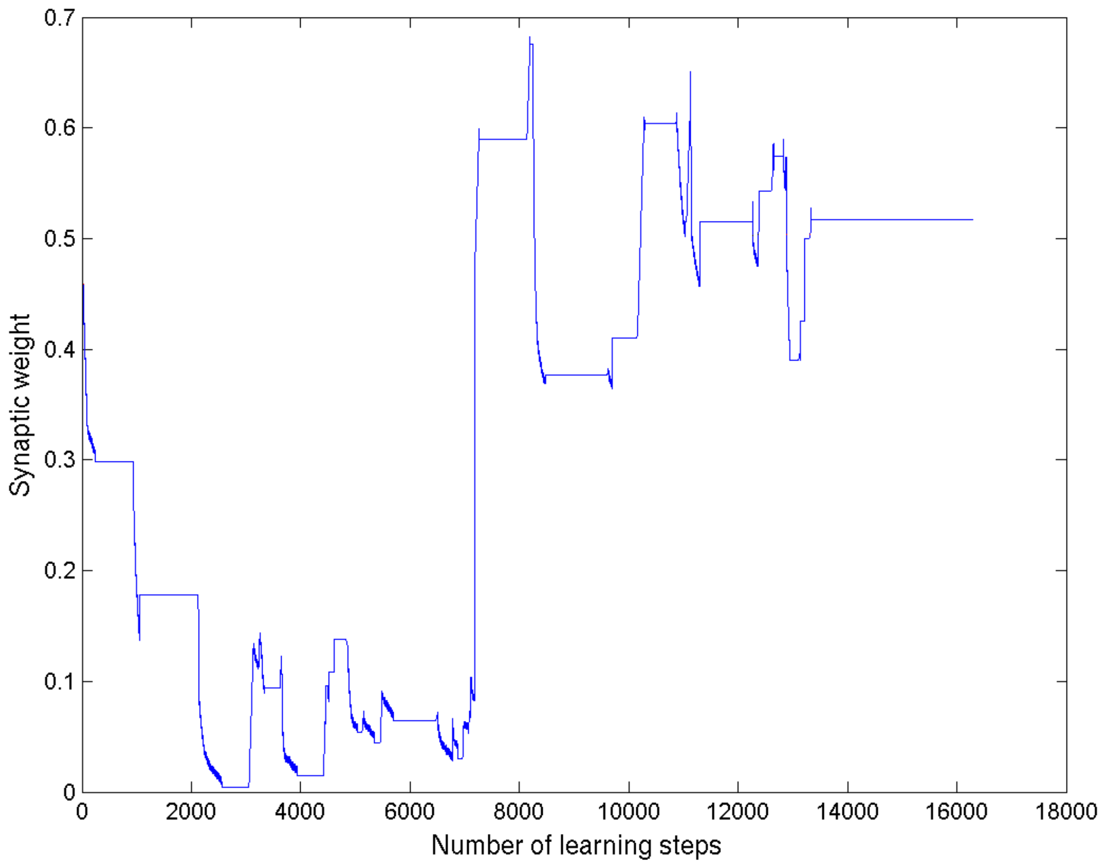


FIGURE 4.13: Dependence of the number of learning steps on faulty components (the red line for faulty nodes and the green one for faulty synapses) in the network of 500 neurons.

most optimal output patterns.

The phenomenon can be explained by taking into consideration the way the neural network stores the learned information. Intuitively we could presume that the learning efficiency of the network grows with the number of neurons. However, this is only valid to a certain extent. The applied input pattern activates the corresponding neurons and causes them to fire further to the post-synaptic neurons. Those neurons, however, activate only if their input weights are strong enough to cause the accumulated signal to exceed the firing threshold.

Due to these aforementioned activation constraints, the activity paths are formed in the network. Corresponding to each input, the most probable signal propagation will follow the associated pattern. However, when the number of input patterns or the connectivity level increases, the activity paths overlap, thereby destroying each other and corrupting the output result.

4.6 Biologically-inspired Agent

In this section we present a biologically-inspired agent, which represents a learning system. We understand a learning system as a system able to decide in a reasonable way what to do in a particular situation from previous experience and/or provided examples of appropriate behaviour even though the situation may not be experienced by the system before. This allows specifying 'what' the system should do for each case, and not 'how' the system should act for each step.

An agent is designed to study the proposed approach of simulating the neural network. The agent is able to accumulate the vital experience from an ambient environment. The neural network of the agent determines the behaviour of an agent based on its own observations. It is a dynamic system that accepts the agent's input sensor values and provides the output values, which correspond to one of the possible movements - for example, turn left or move forward. If all output values are set to zeros, the agent moves in random way.

A 3D virtual application has been created for simulating a self-learning agent hunting food. Food has a fixed position and only interacts with the agent when they are in the same geographical position (i.e. when the agent catches food), adding energy to the agent and disappearing. Another food appears at the same time in a randomly chosen position of the environment. Energy is a measure of hunting success and diminishes when the agent unsuccessfully moves in the environment for a long time, causing the agent to "die" eventually.

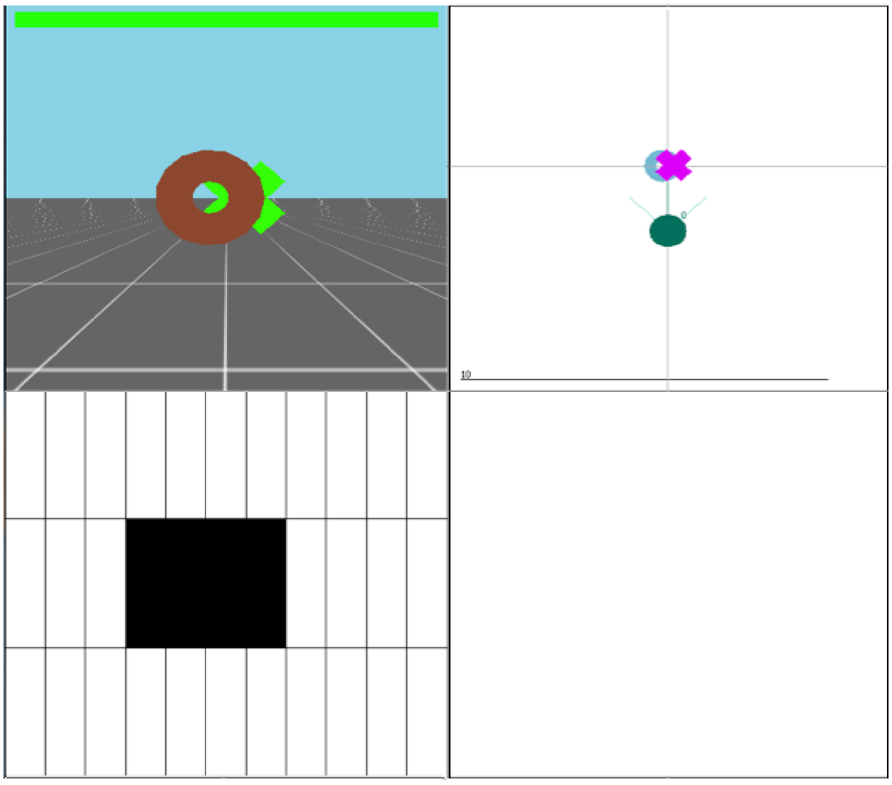


FIGURE 4.14: Environment view.

4.6.1 Environment

The screenshot of the environment's view is shown in Fig. 4.14, with a top down view in upper right corner, which displays the position of the agent and the food in the environment. Also a pair of displays are available that show what an agent can see, one giving a 3D view with an energy bar above and the other is giving the more primitive, pixilated view, which actually is used as the input to the neural network.

4.6.2 Agent and Food

The agent has five characteristics: perception, execution, decision-making, reasoning and learning mechanism. It also includes some other attributes (see Fig. 4.15).

- Perception: the internal representation of the agent's environment through its vision sensors.
- Learning: the adaptation mechanism, which saves newly acquired useful

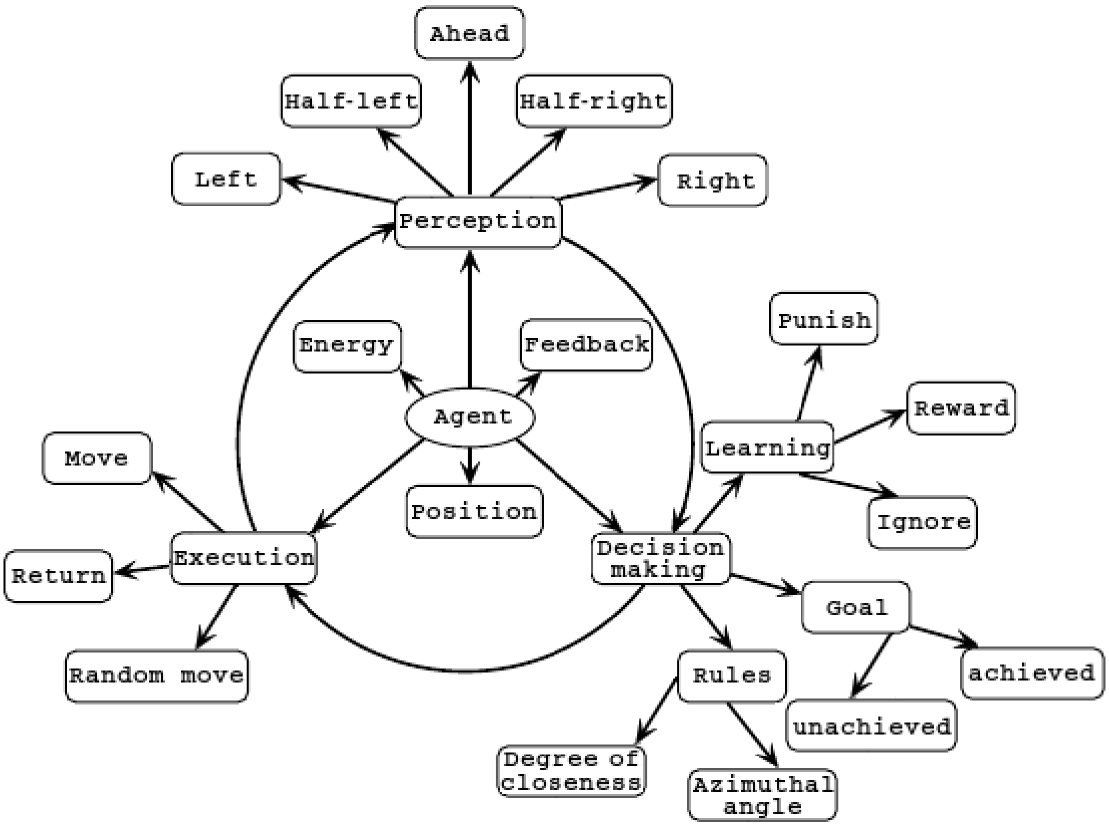


FIGURE 4.15: The agent design framework.

experience and removes redundant experience to properly utilize the capacity for efficient use of essential knowledge.

- Decision-making mechanism: the way to determine which action the agent should take under certain circumstances. The decision should be based on its objective, perception and a set of predefined rules.
- Execution: the agent executes the selected action. The agent may move forward to a new place, return to the previous position or start moving randomly if the target is not visible.
- Reasoning: the set of rules according to which the success of the previous move is calculated based on the change in the azimuthal angle and the distance to the target.
- Attribute: an agent may have attributes such as how well the agent behaves, what its physical position is and what kind of feedback it receives from the previous move.

At each simulation step, agent builds its perception of the ambient environment through the visual sensors and then makes decision based on its perception, goal and learned experience. Finally it executes the made decision. The changed status of the system will in turn lead to new perception of agent.

The neural network must in some way receive information to deinforce or reinforce the performed behaviour. Thus, it is necessary to estimate the performed action in a given state in terms of total expected reward in the long term. A commonly used approach consists of creating a table of reward values for each state – action pair. However, we consider this approach as inadequate because in a biological system the learning reward is based on the (dis)satisfaction level induced by some released chemicals. It is also not practical for a system with a large number of possible states. For example, Tesauro successfully used a feed-forward neural network to implement a backgammon player (containing about 10^{20} possible positions) [169], which would be impossible in case lookup-table were used. The success of the previous move is calculated based on the Boolean feedback signal.

The energy defines the level of success for an agent. Initially the agent starts with a maximum level of energy. When the agent acts unsuccessfully for a long time, its energy degrades. When the energy eventually reaches its minimum level, the agent dies.

4.6.3 Food Searching Algorithm

An agent and a food are located randomly in the environment. The agent's vision is limited by its view, which is defined as an 11x3 matrix of cells, as shown in Fig. 4.14. The agent perceives the environment through this view and can identify the food only within a defined distance. Only when the food is visible, the neural network is engaged; otherwise the agent moves randomly because no learning happens without the visible object. The food chasing algorithm is presented in Fig. 4.16.

Initially, while the food is not within the range of the view, the agent moves randomly in its attempts to find the food. When the agent spots it, its behaviour changes. Now with each movement the agent's view is passed into the neural network, which processes it and suggests the next motion. The agent performs a try-step and evaluates the distance to the target. If the distance to the target decreases, the positive feedback signal is sent to the neural network and the input-output pattern is “saved”. Otherwise, if the distance to the target increases or remains the same, negative feedback is sent. In this case weight redistribution

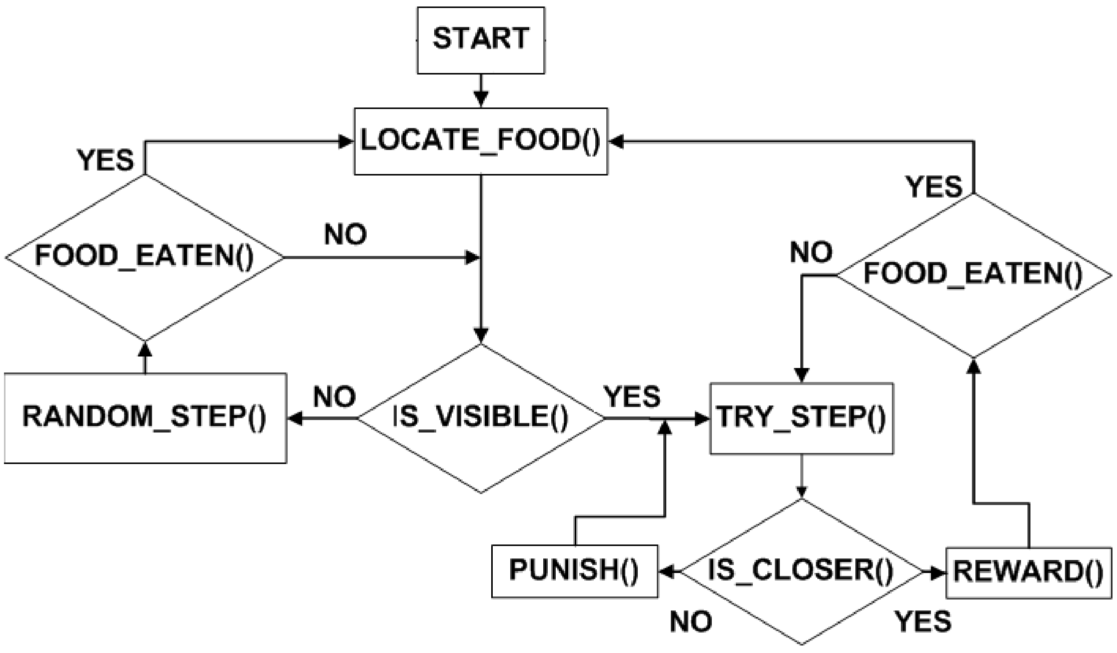


FIGURE 4.16: The food search algorithm.

takes place and an alternative try-step is suggested to the agent.

This process is repeated while the food is within the range of the view or the agent does not approach the food. In this way the feedback signal is based on the agent's own judgment of the effect of its previous motion. This is the first proposed technique, which ensures the agent-environment interaction maintains only through one single Boolean feedback signal retrieved from the agent's field of vision without the need of any extra information.

Each node in the input layer corresponds to a cell in the agent's view. In this way the distance to the target is evaluated in a biologically plausible approach by comparing the last two views only. The weight of every input node connection corresponds to a certain array, the values of which follows a Gaussian distribution. This allows the neural network to distinguish the distance to the target simply by calculating the input layer activity, which is inversely proportional to the distance between the agent and its target. The concentration of the higher coefficients in the centre of the input neuron's array presents the angle of movement. The agent has a higher chance to reach the food when it is in front of it and thus the active neurons are more centred in its view.

Using this method, the neural network measures the success of the previously suggested movement and either strengthens or weakens the connections among the nodes that have been active while processing the corresponding input pattern. This is the second proposed technique, according to which the feedback signal is based not on the network's output but on the next input provided to the neural network.

As initially the network does not contain any pre-learned patterns, its output is purely random. After each successful attempt the action is learned by the neural network block thus increasing the chance to provide similar output in similar circumstances, which might arise in the future. If, due to the suggested motion, the distance to the food increases, the anti-learning mechanism starts and continues until the proper output is found. As a result, the corresponding input-output patterns will be formed and stored in the neural network. This will increase the probability of choosing matching actions as a result of receiving the corresponding view.

Un-learning is the third proposed technique, which provides a possibility to develop more optimal strategy, when newly acquired experience is more effective comparing to known already. As example, it is more efficient to move towards the food directly instead of approaching to it by making circles around.

4.6.4 Decision Making Mechanism

At each simulation time step, agent movements are based on following simple rules:

- If alive, the agent scans its environment;
- If the target is not in the field of vision, the agent moves in random way;
- If the target is observed, in its field of vision the agent performs a try-step;
- If the try-step is successful, the reward mechanism reinforces the knowledge;
- If the try-step does not lead to success, the agent returns to its previous position and the punishment mechanism destructs the knowledge;
- If the target is reached, the energy level increases.

4.6.5 Agent Simulation Results

The agent-environment interaction is implemented by providing the simply Boolean feedback signal retrieved from the agent's field of vision without the need of any extra information. Rule-based decision-making mechanism is employed in the simulator. An algorithm is adopted to decide where the agent should move to. Every

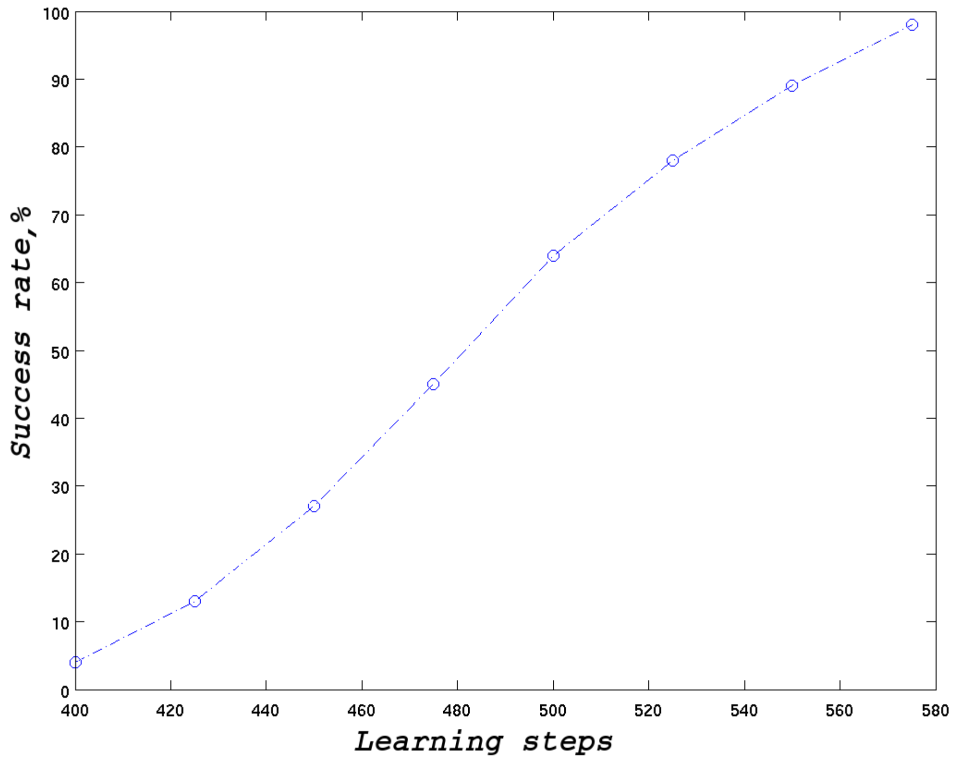


FIGURE 4.17: Food chasing success rate versus the performed learning steps.

simulation time step, agent evaluates its environment trying to approach to the target.

Our experiment proves that the proposed network architecture receiving the feedback based on the localised knowledge about the environment indeed works. Using C++, according to the proposed method we created the neural network and subjected it to the number of the experiments. During every experiment the network was associating 10 various views of environment with a proper action chosen by the network and leading to the reduction of the degree of closeness. Fig. 4.17 shows almost linear dependence of the success rate of food chasing on the number of learning steps.

Fig. 4.18 presents the dependence of the learning performance on the increasing number of patterns to be learned, while the agent learns new actions. The dependence is linear and decreases with the higher number of patterns to be learned.

We compared the dependence of the agent's success rate in searching a food with and without applying the un-learning technique (deinforcing process) in order to limit the negative effect of active paths interference. Fig. 4.19 shows the agent's success rate increases by 50 % after applying the un-learning technique.

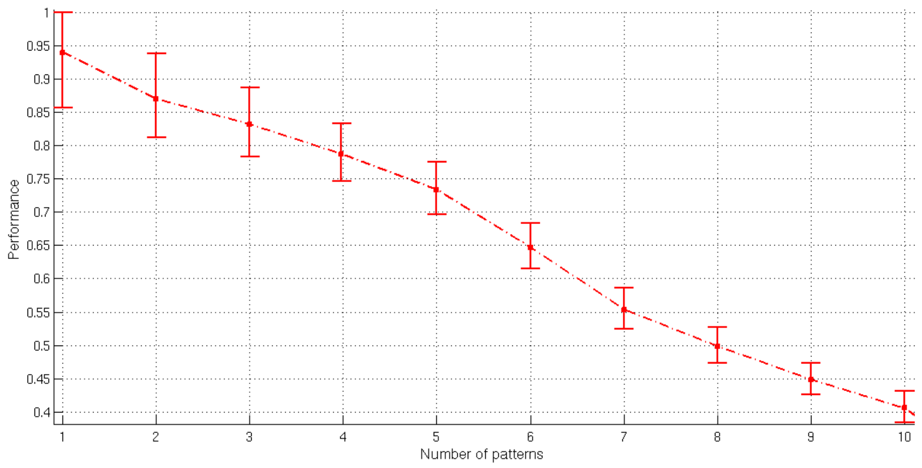


FIGURE 4.18: Dependence of the neural network performance on the learned patterns number.

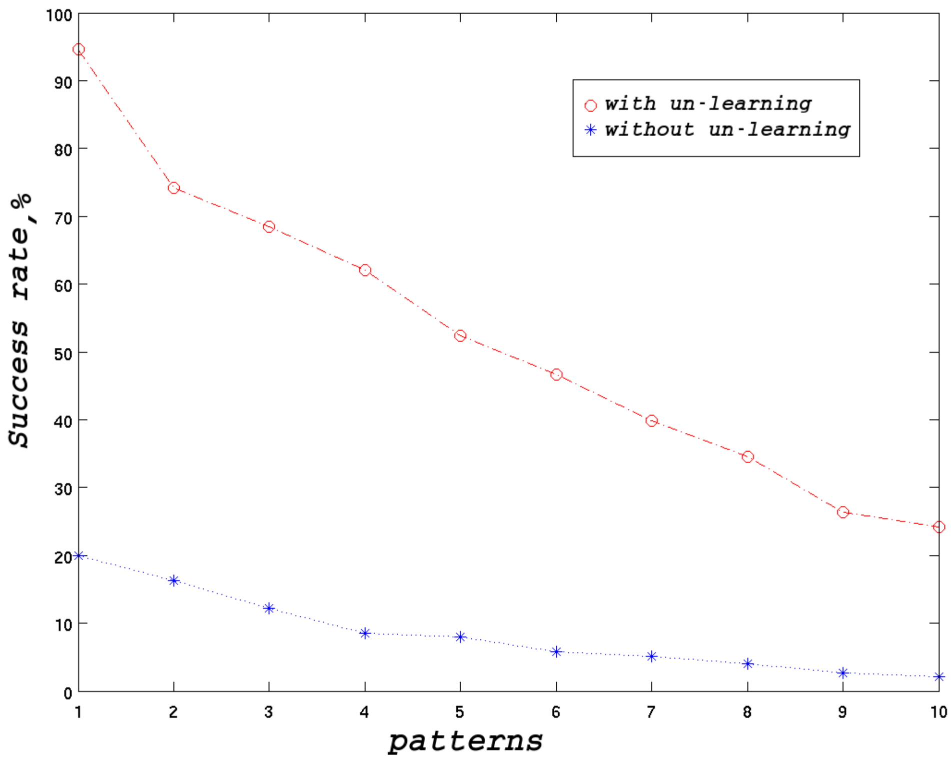


FIGURE 4.19: Comparison of the performance with and without un-learning technique as a function of the learned patterns number.

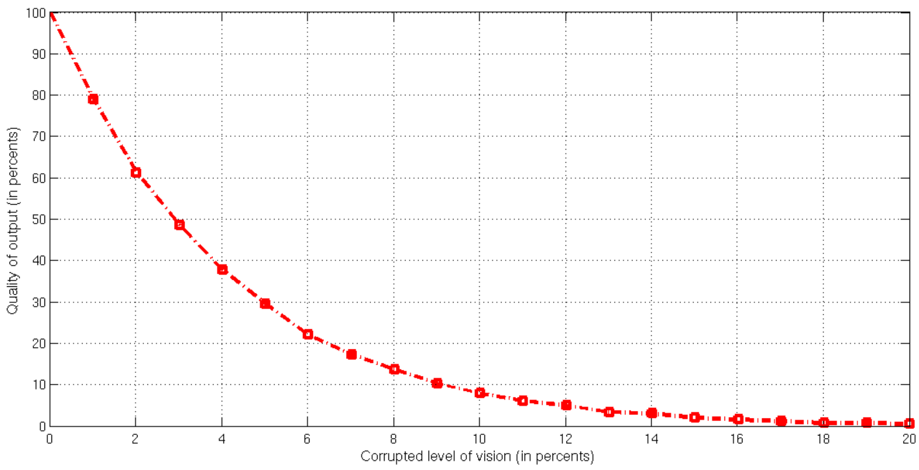


FIGURE 4.20: The influence of the corrupted level of vision on the quality of produced output.

We also paid attention to the way the previously learned but corrupted input patterns influence the quality of output. For this purpose we artificially inverted part of the input patterns' bits (each input pattern has 33 bits) before applying the corrupted inputs to the neural network. Later we compared the produced output with the expected one. Fig. 4.20 presents the quality of output when the input is corrupted. It presents how the capacity to find a food is affected with a reduced level of vision. The quality of output equals to 80 % when 1 % of set of input patterns is corrupted, and it decreases to 50 % for 3 % corrupted input.

4.7 Summary

Our goal was to simulate a neural network capable of learning in a biologically plausible way. The simulation program with a flexible, reusable, platform-independent and scalable framework has been presented.

The program employs the feed-forward neural network combining unsupervised and reinforcement learning. The plasticity mechanism was implemented employing the long term potentiation and long term depression mechanisms. An approach how to keep the activity level of the neurons and how to preserve the learned patterns has been discussed.

The software architecture and algorithm have been presented. The implementation details have been introduced and explained. The model has been functionally verified and experimental results are included.

As a case study, an agent chasing a food in an ambient environment. In the simulation agent-environment interaction is maintained only through one single Boolean feedback signal retrieved from the agent's field of vision and based not on the network's output but on the next input provided to the network. On the basis of trial and failure, the network learns certain tasks, such as move forward, rotate, step back. We showed that by applying the combination of Hebbian and reinforcement learning and incorporating the proposed techniques it is possible to teach an autonomous agent to search for a food without providing any hints except for the data retrieved from its limited field of vision.

Chapter 5

Parallel Distributed Simulation

Traditionally, software used to be oriented for sequential computation and executed on one-core computers. These days performance of sequential execution approaches the physical limitation and a new form of computation, known as *distributed computation*, becomes more popular because of higher performance, scalability and fault-tolerance. However, such an approach raises challenging issues, concerning the execution of simulation programs on distributed computing platforms.

The objectives of our study are first presented in Section 5.1. Section 5.2 overviews possible topology exploitation in order to find the most efficient mapping algorithm to optimise the communication load and provide additional performance gain. Section 5.3 discusses various algorithms of time management and synchronisation within distributed simulation.

Section 5.4 introduces a simulation environment suitable for efficient parallel distributed simulation of large-scale neural networks and representation of neurons, synaptic connections and communication between the asynchronous processors through the interchange of spike-representing messages.

Asynchronous execution raised the challenging side-effects, such as the cost of additional distribution-related communication and synchronisation of the parallel processing. Solving them, the novel protocol for local communication between processors was developed, which is immune to spikes delivered late or out of order. The algorithm and distributed hardware allows implementing a fault-tolerant mechanism, immune to processor failures and communication problems. Also, the chapter presents in details an efficient implementation, design and operation of the neural network simulator. Section 5.5 discusses the achieved results and compares them with the research objectives.

5.1 Objectives

The proposed and developed simulator has several objectives. The *first* one is to provide simulation of a large-scale neural network. For this the simulator is designed to run on a distributed hardware, resolving the issues of parallelism, communication and timing inherited in distributed simulations. The *second* objective is a possibility to apply a range of various neuron models. *Finally*, it is essential for the simulator that the definition and design of network model and learning process is biologically inspired (was described in Chapter 4).

Starting from the first objective, the chapter begins with commonly addressed related side-effects and challenges of distributed computation.

5.2 Mapping

Neuron mapping on participating processors is one of the most obvious requirements of distributed simulation. Here we define a network topology, distributing the groups of neurons and their internal connections over the processors.

Evenly distributed average activity and balanced load distribution is crucial for computational and communication load generated by simulation. It is beneficial also to take into account the dynamics of a network in addition to its topology.

5.2.1 Neuron Number Based Mapping

An intuitive and the simplest possible distribution we can apply consists in assigning an equal number of neurons to each machine. This approach is easy to implement and simple to maintain. However, various neuron groups have uneven computation cost leading to the irregular communication in the network. This produces non-uniformly distributed areas with different activity of message passing, which results in non-optimal usage of resources.

5.2.2 Delay Based Mapping

It is also possible to map a neural network according to the time delay between the groups of different neurons. However, some learning implies a variation of delays in the network. If such learning is applied to a network, mapping optimization may cause certain difficulties and quickly affect the quality of mapping. This may result in non-optimal usage of resources.

One solution might be to rebalance the distribution of the neural network during the simulation, according to the changes of delay time. However, the additional costs should be evaluated, which are generated by the transmission and data structures update.

5.2.3 Average Activity Based Mapping

Another type of dynamic load balancing is also possible depending on the activity of the neural network. The neurons most active during simulation could be redistributed to optimise the mapping. The activity of the neuron cannot be predefined because it depends on the input, applied to the network; as well as overall network activity changes dynamically during the simulation. Therefore, it is extremely cumbersome to find the ideal dynamic re-distribution of neurons between the processors depending on their activity level. An approach of dynamic re-distribution leads to further increase of the potential computational load of the network.

5.2.4 Neuron Connection Based Mapping

Taking into the consideration the utilized topology of a fully-interconnected feed-forward network containing three layers, an intuitive way of thinking suggests mapping of each layer on a separate processor. Since there are no internal connections in the same layer, the internal communication of each processor is minimal as no events are sent locally. At the same time the inter-processor communication is maximal as all events are sent over the communication network, overloading the simulation. It is obvious that for an optimal performance a trade-off between the local events and external events must be found to get the balanced network communication with optimal neuron distribution.

5.2.5 Conclusion

It is very important to find the balance taking into account the network dynamics in addition to its topology. In practice, however, it is extremely complex to find the ideal distribution for topologies and dynamics of neural network dividing it in the separated processors. Consideration of a dynamic allocation is cumbersome to implement and it leads to further increase of the potential computational load.

We choose mapping of the neurons with a maximum number of incoming and outcoming mutual connections to the same processor. It significantly reduces the amount of messages sent among the processors since the majority of events are aimed for locally-based neurons. The number of inter-processor links determines the quality of partitioning. Such an effective approach is beneficial because it is applicable for a wide range of neuron network types.

5.3 Distributed Time Management

A process of maintaining a conscious control over the event generation, emitting, storing, and processing is called *time management*. It supports a proper synchronisation of execution of distributed simulation. It ensures that events are processed in a correct order as well as guarantees exactly the same output as a result of repeated execution of a simulation with the same inputs.

5.3.1 Synchronisation

In a neural network simulation spikes are considered independently. However, events are rarely independent of each other. An important requirement of neural simulation is that if two events influence each other, they have to be executed in the correct chronological order. Events cannot be sent until earlier event processing has not been finished. The processing of an event in the foreseeable future is based on events that precede it. However, it is hard to determine, which events can be executed in parallel as the mutual influence of two events depends on the input and the internal state of the neurons, which is very unpredictable [108].

An inconsistency with a sequential simulation equivalent can arise, for example, if due to some reason a spike, coming through an inhibitory synapse, is postponed and processed later than a next in turn spike, which came through an excitatory synapse and caused a neuron to fire. So that if the postponed spike has been processed in time, a neuron would not fire in a first place.

Each processor sequentially performs events processing in accordance with the temporal order of these events, however it is necessary to adopt a method of managing global virtual time. A processor has knowledge about a subset of events processed in simulation. However, in the overall view of the simulation, the temporal events order must be consistent with a sequential simulation equivalent.

A processor has to check that the time of their next event queue is consistent with the overall time. The only synchronisation of the event list of the simulator is no longer sufficient as each processor has its version of the event list. To ensure the

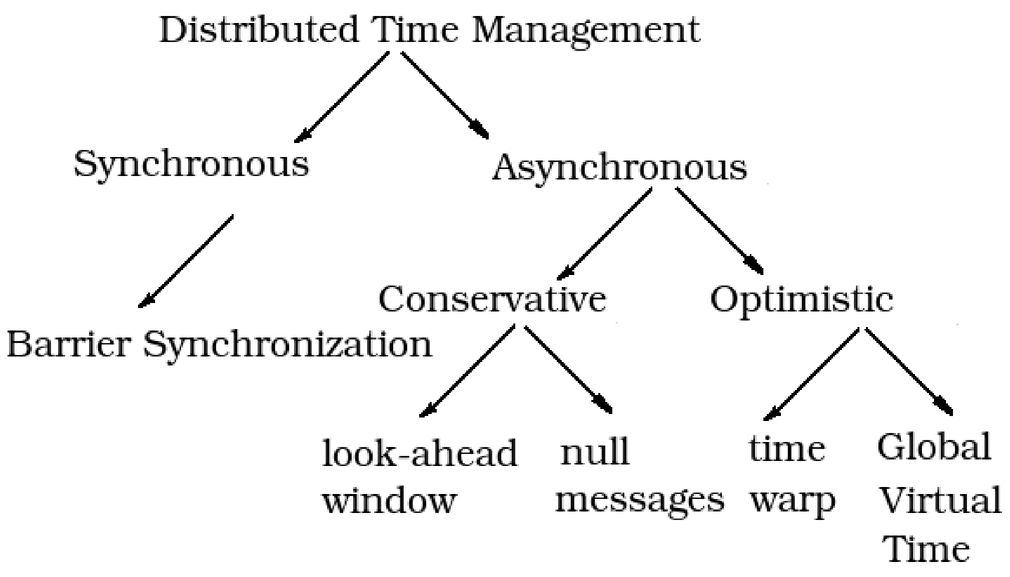


FIGURE 5.1: Distributed time management techniques.

temporal consistency, two types of global virtual time management are possible: synchronous and asynchronous (Fig. 5.1).

5.3.1.1 Synchronous Simulation

The goal of synchronisation mechanism is to ensure that each processor executes events in time-stamp order. This requirement is referred to as the *local causality constraint*. It can be shown that if each processor adheres to the local causality constraint, execution of the simulation program on a distributed environment will produce exactly the same results as an execution on a sequential computer [108]. It helps to ensure that the execution of the simulation is repeatable.

Synchronous simulation requires an explicit synchronization between each of the phases of processing. This can be done employing *barrier synchronization*. Barrier synchronization is a program's instruction for each processor (for instance, after sending a message) to wait until all other simulation processors have also reached this point in the program.

One processor can be used as a controller, requiring synchronization of other nodes at specific time intervals. It awaits the other processors' message containing information on the global virtual time allowing it to process upcoming events. In the example shown in Figure 5.2 the simulation time step Δt used in a neural network. We assume that LP_1 processor plays a role of a controller and indicates the global time. Local time of each other processor (in our case LP_2, LP_3, LP_4)

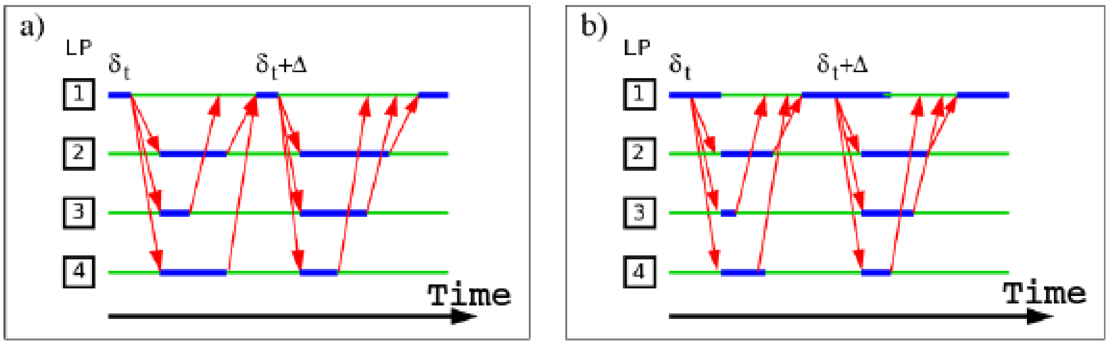


FIGURE 5.2: An example of synchronous simulation with a controlling processor not participating (a) and participating (b) in simulation. Thick line (blue) represents processing state, fine line (green) – idle, arrows (in red) – communication.

is always the same as for LP_1 processor. Time flow of activities of four processors during simulation is presented when the controller LP_1 is chosen as dedicated entirely to synchronization task, case *a*, and as well participates in event processing, case *b*. In both cases, each processor spends much of the time in idle state while waiting for the next synchronization. Each time the LP_1 reads the time stamps of upcoming event of other processors and provides a new global virtual time to all of them.

This approach is often used for time-driven simulation, where it is necessary that all processing units be synchronised to each slice. This means that the simulation proceeds to the next time slice only when all the processing units have gone through the current time slice. So that the next time slice can be compared to a limit that must be achieved by all processing units. Due to this rigidly coupled synchronization such a distributed simulation is very sensitive to changes in load distribution when most processors remain idle waiting for the most overloaded one.

5.3.1.2 Asynchronous Simulation

A processor does not receive events in any order. Each processor establishes communication channels with all incoming and outgoing of other processors. On each channel, message sent in chronological order arrives to the destination in the same order (according to so called First-Input-First-Output principle). This feature of the simulation is exploited to develop asynchronous methods of decentralized synchronization.

In asynchronous simulation, time management is decentralized, processors communicate by exchanging messages to maintain a global virtual time for synchronisation.

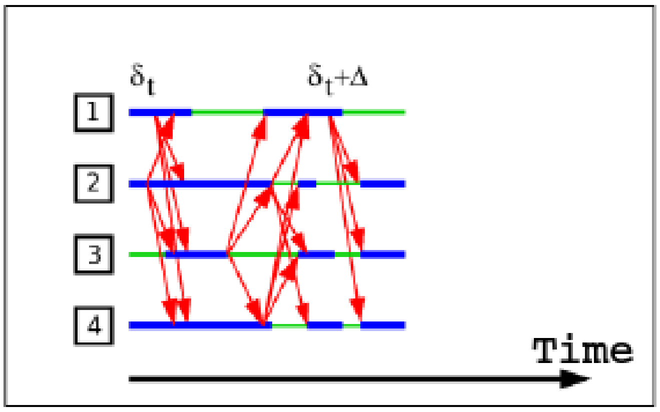


FIGURE 5.3: An example of asynchronous simulation. Thick line (blue) represents processing state, fine line (green) – idle, arrows (in red) – communication.

Fig. 5.3 illustrates an asynchronous simulation. Here waiting times are reduced because work progress of each processor is exchanged with others. The acceleration of an asynchronous simulation to synchronous simulation can be at best O(log(P)) [170]. More decentralized methods generate more message emission to perform the required synchronization.

The potential gain of an asynchronous method over a synchronous method depends directly on the dynamics of simulated networks and therefore an amount of produced synchronization messages. Depending on average activity of a network in certain cases it is possible to take advantage of this dependence limiting the number of extra messages required for asynchronous time management.

This form of synchronisation is particularly effective for event-based simulation, where processing units are not as rigidly coupled conversely to event-driven simulation. Simulation facilitates when the events are produced in larger intervals than the time range dictated by the underlying time resolution. This is the case of neural network with reduced connectivity and therefore with a low firing activity.

5.3.2 Deadlock

If a number of unprocessed event messages is relatively low comparing to the number of connections in a network, or if the unprocessed events become clustered in one part of the network, a specific condition may occur, when two or more processors are each waiting for the other in order to progress the global time. It results in one or more processors being blocked, waiting for messages from a processor, which is itself awaiting one of them. Such kind of situation is called *deadlock*. Quite often deadlock can overflow memory when the events are accumulated locally without being sent to the target processor.

For deadlock avoiding, two common approaches are used: the *conservative* and the *optimistic* synchronisation. The conservative method processes only "safe" events, preventing all types of deadlocks whereas the optimistic synchronisation processes all available events, even the events, which could falsify the previous calculations and if deadlock is identified, it fixes any generated errors.

5.3.2.1 Conservative Synchronization

The distinguishable feature of a conservative synchronisation method consists in avoiding deadlocks at any cost. This is achieved by limiting events to be processed by a so-called *secure window*. This window is set by the processor hosting the source neuron and is known as a *look-ahead* window.

Larger size look-ahead window allows the simulator to run faster. This gives an opportunity to apply sophisticated algorithms for calculation of an optimal look-ahead, where the principal task is to define when it is "safe" to process an event so that no event containing a smaller time stamp will be later received by the processor.

A processor being at simulation time T guarantees that any message to be sent in the future will have a time stamp of at least $T + L$ regardless of what messages it may later receive, as shown in Fig. 5.4. The processor is said to have a look-ahead of L.

Another way to avoid deadlock is using *null* messages. Once a processor progresses in simulation, it sends a null message to each outgoing link. Such a message does not carry any information except the local time of message sending processor and guaranteeing that no more events will be sent until this time.

Provided with this information, receiving processor safely executes the events progressing to the indicated time. Null messages are processed by each processor just like ordinary non-null messages, except no activity is simulated while processing of a null message. It only advances the simulation clock of the processor to the time stamp of the null message and none of state variables are modified and no non-null messages are sent as the result of processing a null message. Whenever a processor finishes processing a null or non-null message, it sends a new null message on each outgoing link.

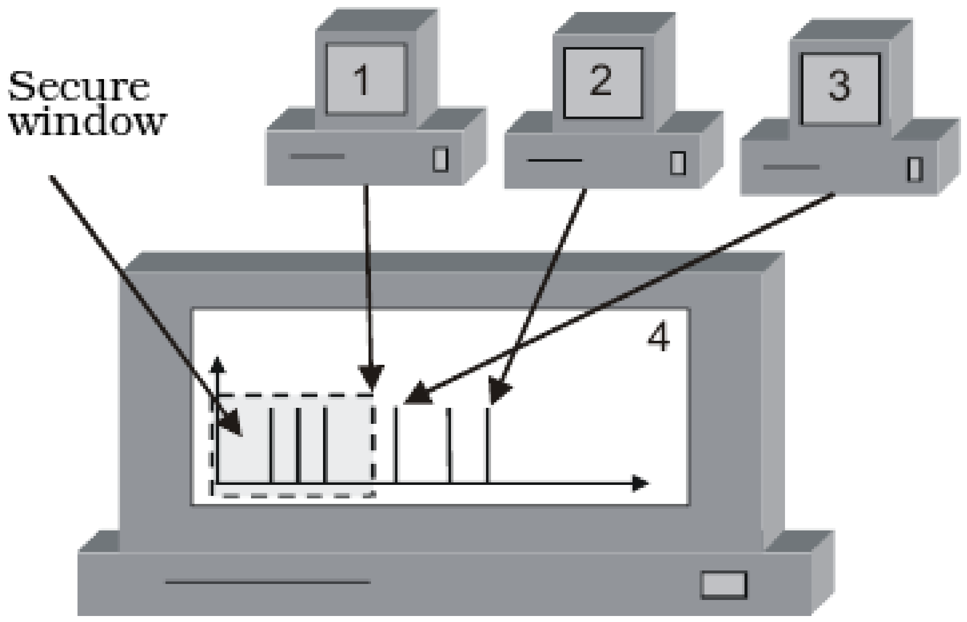


FIGURE 5.4: Secure window in conservatively synchronised simulation.

A deadlock-free nature of this approach was proven by [171]. However, the principal problem of this method is that the null messages cause the high volume of communication considerably restricting the speed of simulation.

5.3.2.2 Optimistic Synchronization

Contrary to conservative approaches that avoid violation of the local causality constraint, optimistic approaches allow violations to occur. However, they are able to detect and recover from them. This allows exploitation of greater degree of parallelism as two interdependent events can be executed simultaneously. A system recovery is performed upon the time when an error occurred if one of the events affects another. Also, optimistic approaches are more flexible and transparent and do not depend on application specific information in order to compute which event is safe to process.

The Time Warp mechanism [172] is the most widespread optimistic method. According to it, upon the receiving of an event with a time stamp smaller than one it has already processed, a system rolls back until the time the error occurred (for this checkpoints are used). After it the system reprocesses the sequence of events in timestamp order taking into account the newly received event. It is necessary to store the events processed in the past and the certain amount of recent states of the neuron to be able to restore itself to the necessary state.

The main disadvantage of this method is high memory requirements, even if no roll-backs occur. The roll-back operation requires additional calculation. This presents another disadvantage of computational overload, which is acute if the number of roll-backs is exceeding certain limit.

During the roll-back operation the process sends the corresponding anti-message and restores its previous state. An anti-messages mechanism is provided to "un-send" messages and cancels their impact. In this way it revokes the erroneous impact and improves the progress of simulation. The anti-message is a copy of a previously sent matching message. In the case when both messages are stored in the same queue, the both are deleted. However, the main limitation is that input/output operations cannot be rolled-back.

A pure Time Warp system causes excessive memory utilization and long rollbacks due to overly optimistic execution. To address the constraint of consumption a large memory amount for storing neuron states and anti-messages is partially solved by global virtual time (GVT). It represents the lower bound on the timestamp of any future messages (or anti-messages) and is equal to the smallest timestamp among unprocessed and partially processed messages. In this way the storage of messages and states older than GVT is inessential and can be reclaimed.

The optimistic methods are particularly effective for simulations not prone to deadlocks, which require little communication. In these cases, only the deadlock detection influences the processing time. Otherwise, when deadlocks are frequent, the computational costs of error recovery can significantly extend the progress of simulation.

5.4 Simulator Structure

Along with the choice of hardware resources, mapping type, and distributed time management, none the less important is the representation of simulation entities, such as neurons and the way of information exchange.

The simulator is programmed entirely in C++ using an object-oriented architecture. The simulation consists of a number of logical processes, responsible for one or a subset of neurons. The simulator works on the principle of events exchange between the neurons by means of messages circulating on the network between the processors. In this way the main task of each neuron is to receive a spike and possibly to create one in return. The following sections will help to understand the operation more precisely.

5.4.1 Communication Mode

In biological neural networks information is conveyed in a one-to-many pattern. Depending on the method of event management, communication mode can be either *sender-oriented* or *target-oriented*.

In a *target-oriented* communication mode a source neuron sends a message to all other neurons, which is checked by every neuron in the network and only the corresponding neurons accept and process the message, as shown in Fig. 5.5. The advantage of this mode consists in the considerably low number of events stored in the event list as each spike generates only one event. However, the bottleneck of this approach is that for each event the validation operation must be performed at least by every neuron in the system regardless whether it is related to this event or not.

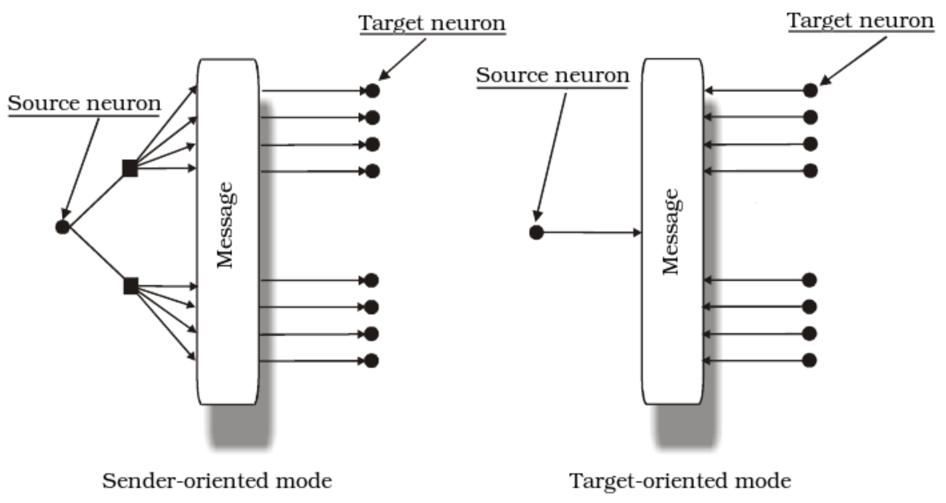


FIGURE 5.5: Sender-oriented and target-oriented communication mode.

If a reference is stored on the output of the source neuron, such a communication mode is called *sender-oriented*. The number of verification operation for sender-oriented communication mode is lower comparing to the target-oriented mode, on condition that an average number of connections per neuron is less than the number of all neurons in the network. This can be seen from the equation 5.1 for target-oriented mode and equation 5.2 for target-oriented mode:

$$k = \sum_{i}^{N} (n_i^{active} * n_i) \tag{5.1}$$

$$k = \sum_{i}^{N} \sum_{j}^{m} (n_i^{active} * n_i) \tag{5.2}$$

where k shows the number of verification operations made by all neurons, N represents the number of neurons in a network, n_i^{active} is the number of active neurons and m is the number of synapses of neuron n_i.

Let us consider a cerebral cortex. It contains the densest network known so far, with a number of neurons around 10^{10} [173], each having on average 10^4 outgoing connections [174]. In this case sender-oriented mode benefits in 10^6 times less number of verification operations. Considering another example of an artificial neural network with 3 layers each having n neurons, which gives $3n$ neurons in the network with $\frac{2n}{3}$ outgoing connections for each neuron, the ratio $\frac{9}{2}$ is still significant.

These calculations indicate that using a sender-oriented communication mode is more beneficial with conditions that:

- an outgoing AP is represented as a single event;
- considering a neural network with the low activity (similar to biological neural networks);
- the average number of connections with post-synaptic neurons is smaller than the number of neurons in the entire network (as it is in biological neural networks).

5.4.2 Spike Message

The basic element of an event-driven simulation is a *message*, containing an *event*. The event is any occurrence that changes the state of the model so various actions of a system can be considered as events.

Events cause a change in the state of the system and often result in the scheduling of other events for some future time. In order to achieve a flexible simulation framework, it is necessary to model an event so that various learning rules could be applied without modification of the simulation engine.

For simulation of neural networks, especially in the case of event simulation, messages representing action potentials (or spikes) are exchanged between neurons. A spike message is a basic element of simulation. It is emitted by a neuron towards one or more target neurons. Every spike message contains a target ID so that each target of this message can identify the activated connection. It is possible to use multi-event messages however at the moment we focus our attention on a single event message. Events must be treated in chronological order in order to preserve the causality of the simulation.

5.4.3 Message Passing Interface

To simplify the programming for distributed architecture, a language-independent Message Passing Interface (MPI) standard communication protocol has been developed by Dongarra et al. [175]. It allows processes to communicate with one another by sending and receiving messages. There are several implementations of this standard: MPICH, LAM/MPI and MVICH, which have similar performance [176]. These implementations provide access to features defined by the MPI standard in C, Fortran and C++ programming languages.

MPI provides broadcast and point-to-point communication. In broadcast communication a message is sent to all nodes within a pre-defined group of nodes. Point-to-point communication delivers messages to a single node and can be performed in “blocking” and “non-blocking” modes.

A communication is “blocking” if the node that initiated the communication (either sending or receiving procedure) is suspended until the message buffer is safe to use. A communication is “non-blocking” if the node initiated the communication can advance immediately after the call without securing the communication buffer is safe to use. In order to test whether the non-blocking communication is complete, additional calls can be used.

The blocking communication is easier to handle as it partially synchronises the nodes. But it causes a delay of large memory block messages in case of a limited connection bandwidth between communication nodes. In order to avoid the impact of message transfer delays we made the choice of using non-blocking mode of communication. In the case of our simulator, the communication initiated nodes make a request and are able to perform other calculations while awaiting for the request to be processed.

5.4.4 Master Processor

Special processor is dedicated for interacting with environment and at the same time managing the other processors work. We refer to it as a *master* processor, whereas all other processors, participating in simulation, are referred as *slave* processors. The master processor performs such operations, as network mapping onto the slave processors, input feeding to the pre-defined subset of the network and output collecting form another subset.

5.4.4.1 Interaction with Environment

Master node receives a set of input-output patterns provided by a file or an output of another program or may generate them itself (by user choice). The safest way to perform this would be to read all inputs into the queue before the simulation starts. However, this means that simulator must store possibly very long input stream, and this is an extremely cumbersome solution.

Instead a single call to a function is used, which fetches one input pattern and provides it in corresponding way to the input subset of neurons. In this way an input stream can be infinite or, conversely, the simulator repeatedly gets only one provided input pattern and gives an output result. Of course, master node should read into the first input pattern before simulation starts.

5.4.4.2 Initialization Stage

During the mapping stage master reads in data from the description files and feeds the initialization data to each of the slave processors according to Algorithm 5.1.

```
begin
    for i = 0 to netSize do
        initMessage([ID][potential][threshold][inputSize])
        for j = 0 to axonSize do
            synapseInitMessage([targetID][weight])
end
```

Algorithm 5.1: Initialization stage.

Each slave receives a message, containing the initial data: neuron's ID, membrane potential, threshold potential and a number of incoming connections. Next received message contains an array of pairs, where target neuron ID and an initial weight are specified for each synapse of the neuron. When each neuron receives its initial values and sets up the required number of connections with the corresponding weights, the initialization stage is over.

5.4.4.3 Simulation Flow

Flow of simulation consists on a number of learning cycles, during which the network attempts to alter its weights in such a way that the output neurons produce an expected pattern. On this level the neural network is synchronized in the conservative way because master starts each new cycle when the previous one is completed.

When an input pattern is fed into the network, spikes propagation is synchronized according to the optimistic approach because each slave behaves asynchronously and warp technique is applied to correct mistakenly emitted spikes.

5.4.4.4 Master-Slave Interaction

During the learning stage, master sends an input to each of the input neurons for every pattern pair to be learned, as presented in Algorithm 5.2.

```
 1 begin
 2    for i = 0 to patternNumber do
 3       for j = 0 to maxAttempts do
 4          resetPotential();
 5          setBarrier();
 6
 7  /* set the input pattern bit to each input neuron */
 8          for k = 0 to inputSet do
 9             sendActivationMessage(k);
10
11          while (!outputReceived)
12             outputCollection();
13
14          if (receivedOutput != expectedOutput)
15             reinforceActiveConnections();
16          else
17             deinforceAllConections();
18 end
```

Algorithm 5.2: A learning cycle of simulation.

At first, master sends a signal to each slave, directing him to reset their potential and later it sets a *barrier synchronization* marker (line 5 in Algorithm 5.2). This barrier is a program's instruction for each processor to wait until all other simulation processors have also reached this point in the program. As all processors run in the asynchronous mode, this security measure guarantees that all processors are receiving a proper input for corresponding cycle. Without it the delayed messages could interfere with inputs of other cycles and deteriorate the final result.

Master 'knows' which slaves are dedicated for representation of input and output neurons. After barrier synchronization, master supplies each input slave with a corresponding input value, indicating them either to spike or not based on the input pattern. Later, master enters a *while* loop in order to collect the outputs. The loop terminates when master receives a complete set of messages from the output subset of neurons.

In line 14 of Algorithm 5.2 master compares the received array with the expected one. If the arrays are equal, master sends a reinforcement message to all slaves, because it does not know, which processor was active or idle. The message instructs the slaves to reinforce the connections, which were used in the input signal propagation through the network at the current cycle. Only the active slaves perform the reinforcement, while the rest slaves ignore the message.

If the received and expected arrays are unequal, the deinforcement message is sent to all the neurons and this time all neurons deinforce their connections accordingly. The input pattern is fed into again until the correct output is found.

The learning phase repeats until all the input-output pairs are recognized by the network at once. When this happens, we assume the network has successfully learned all the patterns.

However, while learning new patterns, old patterns are partially destroyed due to interference of active paths (explained in Section 4.4.1). To overcome this negative effect, the pairs of patterns are periodically shuffled and fed into again. Firstly, this approach assists in finding the most optimal weight values valid for all patterns and secondly, it prevents possible effects due to a specific learning order.

5.4.4.5 Slave Termination

When the learning phase is over, the simulation enters its final stage, when master terminates all the slaves by sending the termination message to each processor. This message forces each processor to complete its operation and to save the current state and the weights of connections to the outside file.

5.4.5 Slave's Workload

A slave processor is idle until it receives a message from another slave or master. Slave does not 'know' whether it contains an input, hidden or output neurons. When the message is received, slave recalculates its internal state, checking whether it reached the condition of spike emission and a new message is going to be generated. If the message is generated, it is sent to all known targets, determined *a priori* while creating the network (discussed in Section 5.2). Slave saves into the *activatedBy* array the ID of the neuron, which excited it, as shown in Algorithm 5.3.

```
begin
    if (messageReceived)
        stateRecalculation(weight);
        activatedBy.push(src);

        if (firstlyActivated)
            preSynapticInform(potential);
            postSynapticUpdate(weight);
        else
            preSynapticUpdate(potential);
            if (!activeAnymore)
                postSynapticErase(weight);
end
```

Algorithm 5.3: Data exchange among slaves.

If the firing condition is fulfilled and this is a first time of neuron excitation in the current cycle, slave emits a spike to the post-synaptic neuron. In order to implement learning, it is necessary to inform pre-synaptic neurons about this event. For this reason, slave sends another message to the pre-synaptic neurons as well. This message indicates the pre-synaptic neurons that they caused a post-synaptic neuron to emit a new spike. Pre-synaptic neurons store this information into the *activated* array.

If the firing condition is met not for the first time in the current cycle, slave needs to inform the pre-synaptic neuron about the alteration of its potential value. This helps to implement the warp synchronization. In the case if it receives an inhibitory signal and is not active anymore, slave sends another message to its post-synaptic neurons. In such a case the post-synaptic neurons warp the neuron's influence onto them, changing their state correspondingly.

5.4.5.1 Applying Learning Rules

Finally, when the output neurons produce the output pattern, slaves receive a message from the master processor. This message informs either to reinforce or deinforce the weights. Two arrays (*activated* and *activatedBy*) contain all the necessary information for applying the learning rules, which were described in Section 4.2.3.4. When weights are altered, a new cycle starts.

5.5 Results and Discussions

After describing the generic neural simulation platform able to support multiple neuron network models and various ways of network mapping on distributed hardware, we discuss the achieved results.

5.5.1 Spiking Pattern

A neural network was simulated to explore the neuron spiking behaviour. Fig. 5.6 represents a spiking training obtained from the network of 40 neurons with input neurons 0, 1, 3, 4 and 9 permanently emitting spikes. At first, the number of excited neurons significantly increases until eventually it reaches the required number in order to excite the right output neurons, when the input-output pair of patterns to be learned. Fig. 5.7 shows the total neuron activity at each learning step.

5.5.2 Weight Alteration

Are all connections equally significant? To answer this question we investigated the weight alteration dynamics during the training process in order to identify the most and the least significant connections. Along with the network structure, the initial state of the weights has a large impact on the ability to learn and resilience of a neural network. Here we look into the optimal initial values of the weights.

A lower level of connectivity results in a lower fault distribution caused by potentially faulty nodes while implementing the network on the distributed hardware. Limiting the number of redundant fault-prone components potentially improves

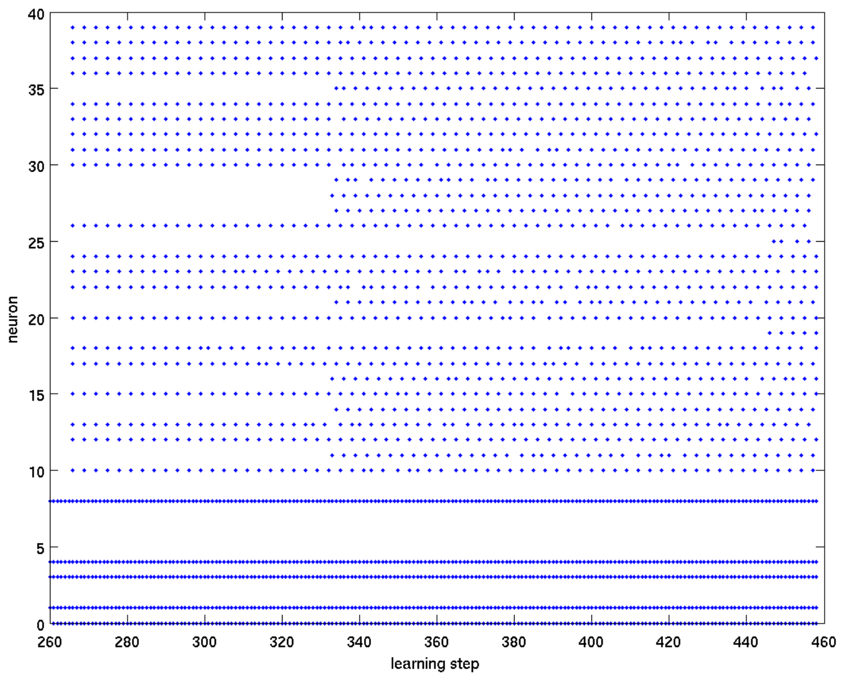


FIGURE 5.6: A sample spike train obtained simulating a network of 40 neurons.

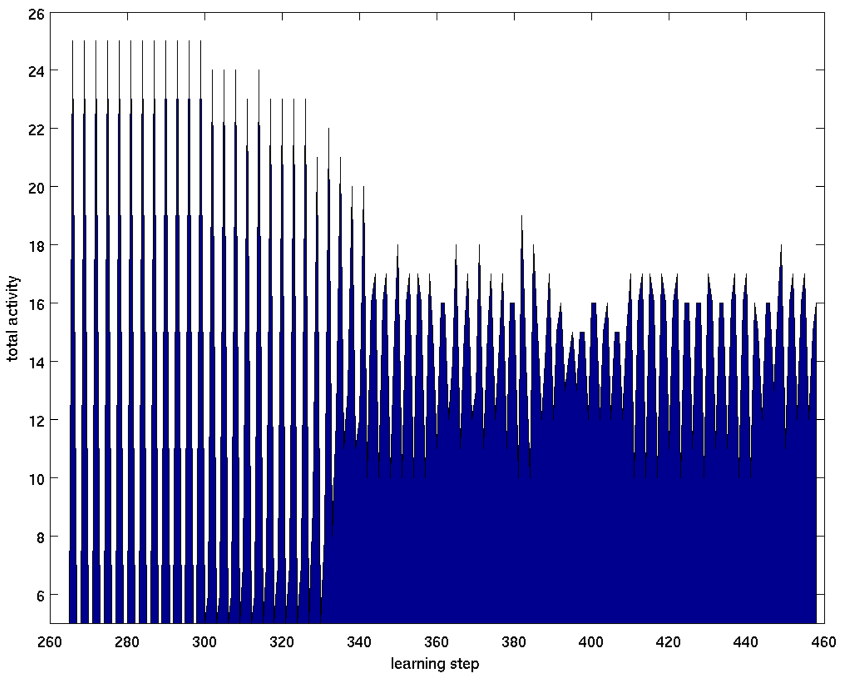


FIGURE 5.7: Total activity obtained simulating a network of 40 neurons.

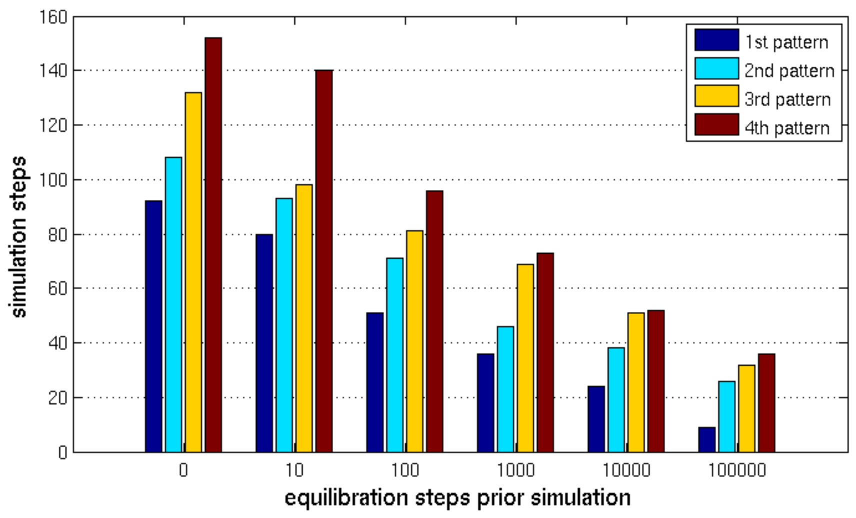


FIGURE 5.8: Training efficiency dependence on the number of equilibration steps applied prior training. During the training phase 4 different patterns were learned. The network size was 2000 neurons with 5 input and 5 output neurons and 90 % connectivity.

the fault tolerance of the entire network. As the faulty nodes have fewer outgoing connections, there are fewer post-synaptic nodes receiving the ineligible spikes. Secondly, fewer connections decreases the chance of a system failure as the mean time to failure (MTTF) of the entire system is inversely proportional to the sum of the failure rate λ of each its constituent, as shown in the Equation 5.3:

$$MTTF = \frac{1}{\sum_{i=1}^{i=N} \lambda_i}. \tag{5.3}$$

Connection links can be safely pruned without significantly affecting the performance. However, while significant connectivity reduction decreases the network sensitivity to faults, it prolongs the training time due to the smaller quantity of connections with alterable weights. Thus a trade-off must be made between the level of connectivity and the adaptation time.

We constructed a small network of 20 neurons (for better visibility) and recorded the alteration of each weight, plotting the measured values against each simulation step. We noticed that training efficiency directly depends on the initial state of the network: weight equilibration before the learning phase significantly improves the training process. This is demonstrated in Fig. 5.8. 1000 random inputs were applied to the network 10000 times before starting any measurements. For each input the "deinforcement" process was applied to the weights, modifying the overall distribution of their values in the network.

The weight alteration dynamics was investigated. We paid close attention to the weights of connections between input and hidden layers (see Fig. 5.9) and between hidden and output layers (see Fig. 5.10) as well as to the weight distribution at the three stages of simulation:

1. at the initial phase, which is reached after the mapping process;
2. after the network equilibration phase, when the LTD rule is applied to the network several times;
3. after the learning phase, when all the pattern pairs were successfully learned.

The Gaussian distribution of weights can be found on the graphs at the initial phase of simulation. After equilibration is applied, the centre of distribution slightly shifts towards the left side for the connections between input and hidden layers and remains almost unaltered for the connections outgoing from the hidden layer.

A significant difference between the weight distribution dynamics can be observed among input-hidden and hidden-output connections. In the first one, during the learning phase, distribution expands over the range from -0.1 to 1 with a centre at 0.45 and a significant concentration of weights around 0.5. However, the weights in the input-hidden connections shrinks considerably from being in the range from -4 to 4 at the initial stage to the range from -0.02 to 0.015 at the final stage with unclearly defined distribution centre around -0.002.

This phenomenon can be explained in the following way. The input neurons tend to excite only specific set of hidden neurons and concentrate their connection efficacy to the limited number of neurons. Because of this, a change of the formed active paths requires more efforts but make the connections more stable and insensible to changes. This behaviour can be seen in Fig. 5.11.

The hidden neurons broaden their influence on the wide range of neurons, although having on them comparatively low influence. These connections are more sensitive to the input changes and faster adapt the required firing pattern by exciting the right output neurons. Fig. 5.12 shows that weights are almost unalterable during the first phase of simulation, eventually changing their weights at the final simulation phase.

Concluding, there are two levels of learning in a multi-layer neural network. Whereas the weights of the input neurons are less sensible to changes, they better

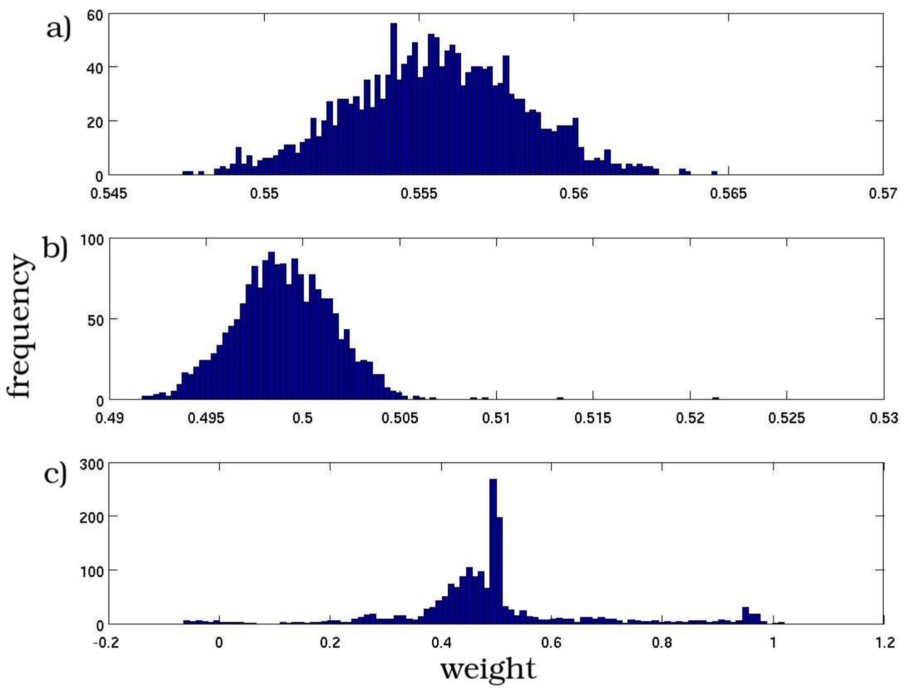


FIGURE 5.9: Weight distribution of input-hidden connections at the initial phase (a), after the equilibration phase (b), and after the learning phase (c).

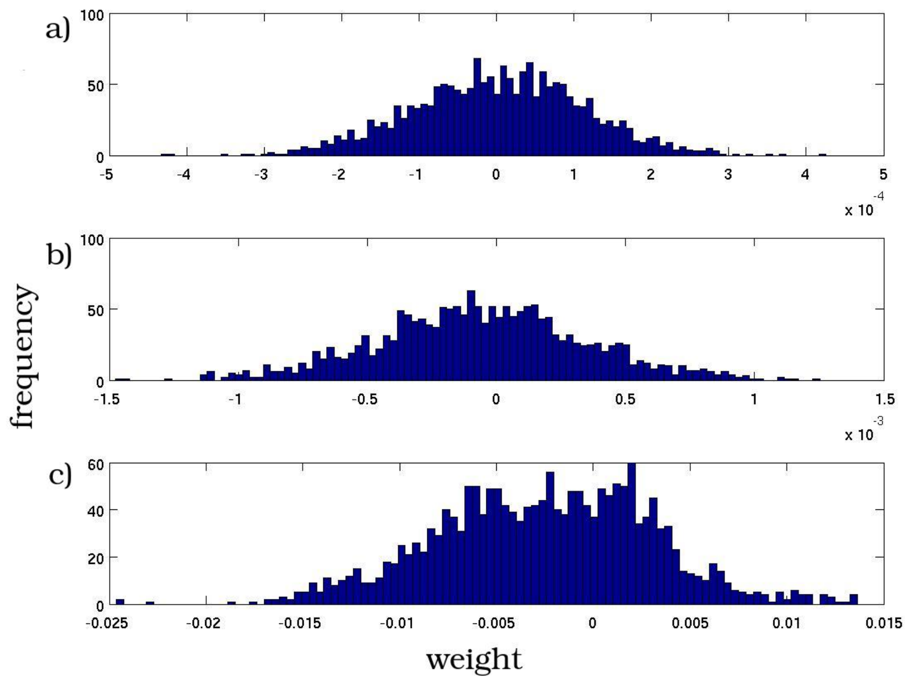


FIGURE 5.10: Weight distribution of hidden-output connections at the initial phase (a), after the equilibration phase (b), and after the learning phase (c).

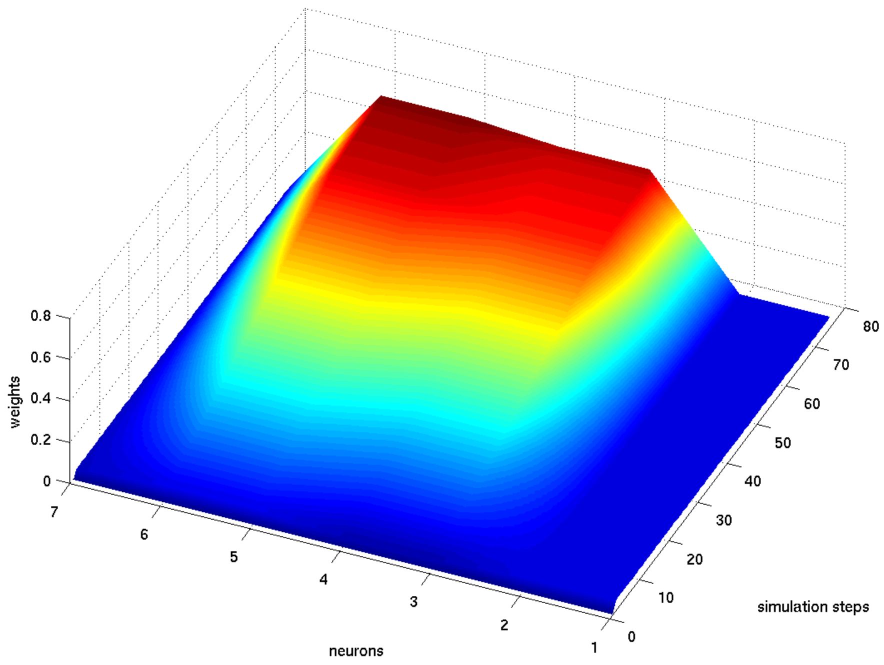


FIGURE 5.11: Weight alteration of connections between input and hidden layers during the simulation progress.

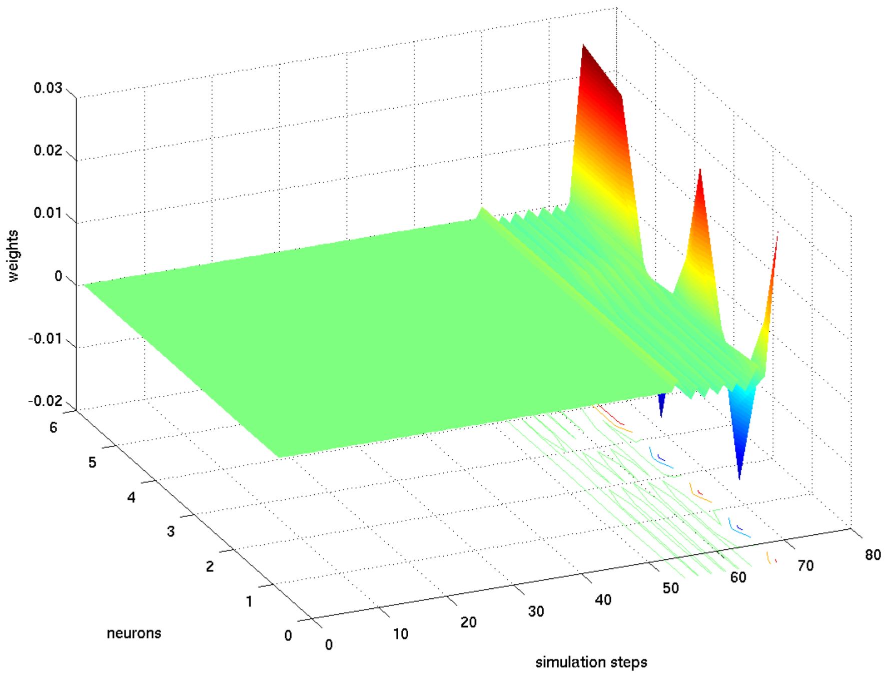


FIGURE 5.12: Weight alteration of connections between hidden and output layers during the simulation progress.

retain the learned patterns as opposed to the weights of the hidden neurons, which being highly-alterable and fastly adapting, easily 'forget' the learned patterns.

This allows us to assume that input-hidden connections are more influential on the adaption of the required output connection compared to the hidden-output connections. Taking into account that connections with large weights are highly sensitive (according to [177]), it is more advisable to prune them before starting the training phase. Pruning the connections of large weights also accelerates training process: the input-hidden connections adapt faster due to their lower number and only after their weights are settled, the hidden-output connections start actively adapting. The adaptation process is shown in Fig. 5.11 and Fig. 5.12, where it is clearly visible that the hidden-output weights start adapting. Moreover, after the training phase is complete, a possible fault in the input-hidden connections is more devastating as it automatically leads to a faulty output in the corresponding post-synaptic nodes.

5.5.3 Performance Estimation

The processing time required to complete one cycle of simulation was calculated and its dependence on the network size was estimated.

Since the real SpiNNaker chip is not available yet, the simulation was performed on the Iridis computer cluster. The cluster consists on 1008 2.27 GHz Intel Nehalem compute nodes connected to an InfiniBand network for interprocess communication. Each node has two 4-core processors with 22 GB of RAM per node. In this way 8064 processor-cores provide over 72 TFlops.

We paid attention to the time performance dependence on the number of physical processors participating in simulation. The virtual neurons were distributed among the physical processors by the principle that the neurons with the maximum number of incoming and outcoming mutual connections are mapped to the same processor. This was done in order to reduce the amount of messages sent among the processors so that the majority of events are aimed for locally-based neurons.

Fig. 5.13 presents the dependence of time required for simulation of a network with 10^4 neurons. Initially, it is very efficient to use more processors, as the computation is dominating. However, the gain in processing time decreases when using a higer number of processors. Such a slow down is caused by communication between processors, which becomes the bottleneck for the further increase in time performance.

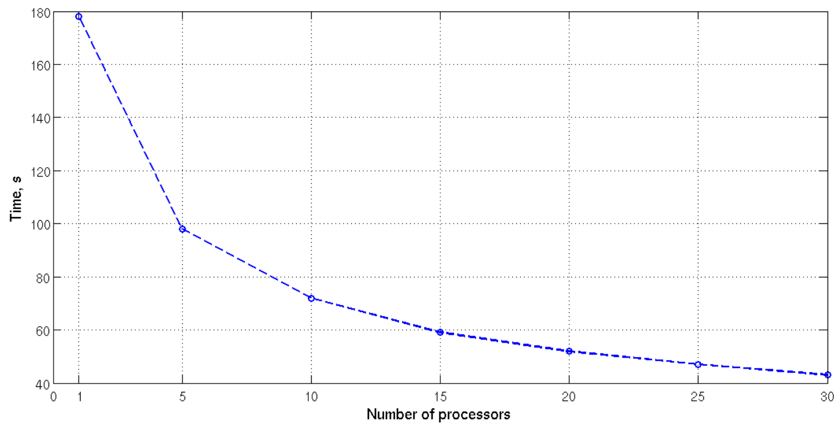


FIGURE 5.13: The time required for simulation of a network with $1 \cdot 10^4$ neurons.

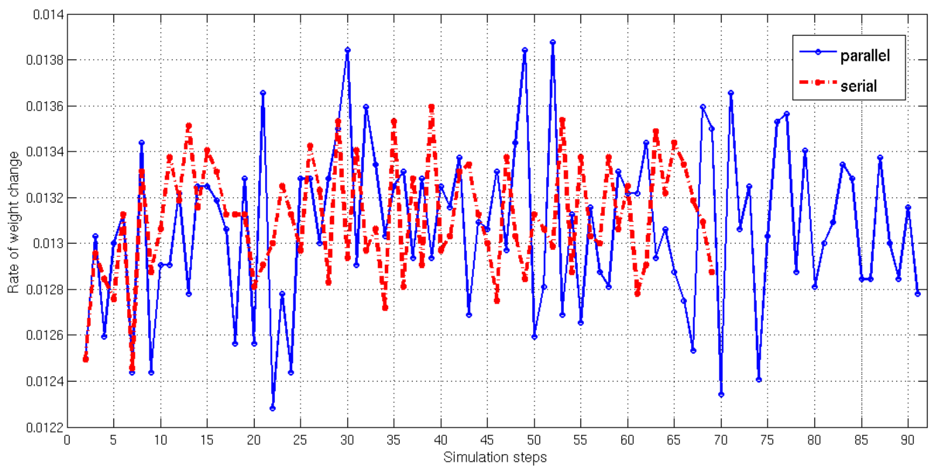


FIGURE 5.14: The mean value of rate of change in the synaptic weights of sequential (dashed red line) and parallel (solid blue line) simulation.

Fig. 5.14 compares the distributed parallel implementation with the single processor implementation in terms of how both solutions affect the learning process. It shows that during learning process with single and multicore implementations the mean values of the rate of change in the synaptic weights are not completely identical, however have a close behaviour. The learning process results in two similar but not identical matrices of synaptic weights, which are presented in Table 5.1. Nevertheless, both weight matrices represent the correct solutions of learning the set of patterns so that for any given input pattern the expected output pattern is produced.

Also, the synaptic weights tend to vary in a more or less baunded range and remain within acceptable bounds, which can be also seen from the normal weight

TABLE 5.1: The weight matrices of synaptic weights after learning process.

Input-Hidden	Hidden-Output	Input-Hidden	Hidden-Output
-0.00656572	0.075	-0.00599528	0.075
0.00586035	0.075	0.00659533	0.075
-0.00664575	0.075	-0.00569383	0.075
0.00592178	0.075	0.00662183	0.075
-0.00603716	0.552874	-0.00647891	1.04672
0.00649823	0.549234	0.00513071	1.03093
-0.00568837	0.55196	-0.00630918	1.03225
0.00664031	0.553604	0.00516618	1.02931
-0.00651443	0.594671	-0.00544413	1.05518
0.00678015	0.589824	0.00613593	1.07608
-0.00643133	0.590156	-0.00578153	1.05124
0.00661079	0.589209	0.00603277	1.06152
-0.0074707	0.0625	-0.00782281	0.05
0.00626806	0.0625	0.00637752	0.05
-0.00725277	0.0625	-0.00688018	0.05
0.00624353	0.0625	0.0061044	0.05

distribution in Fig. 5.9 and Fig. 5.10. This is caused by the fact that every time when a network is confronted with a new input pattern to be learned, generating the weight fluctuation, it should find back its balance in order to retain the previously learned patterns.

5.5.4 Fault-tolerance

Faults do and will occur in a system over time, and there always will come a time when performance is below acceptable limits. However unlike traditional computing techniques, the neural network approach does not insist on exact computation. There is the strongly non-linear nature and the distribution of information or knowledge throughout all of the network. Almost all of the units and connections participate in producing an output either directly or indirectly. Since it is difficult to determine exactly the required amount of processing units and their connections, their redundant number results in a higher degree of reliability. Thus the malfunctioning of a particular element of a system should not greatly affect the systems function if there is sufficient redundancy.

Any fault will influence the output to some degree since all components participate in any computation. This leads to graceful degradation being exhibited by most neural networks, i.e. neural networks will not suffer catastrophic failure, and also allows approaching failure to be detected by using a continuous reliability measure. The fault tolerance that results in this reliability is not inherent within neural networks: it does need to be specifically designed and built into them, and so the architectural complexity which often arises due to various fault tolerance techniques being used is absent in neural network systems. Although any faults which do occur cannot be located, they can be removed from the system since neural networks can learn.

We investigated the performance degradation of the artificial neural network under the presence of faults in a representative pattern-recognition task and analyse its insensitivity to limited errors in the computational hardware.

In order to collect statistically valid data each simulation of the network of 2000 neurons was run 1000 times. Each time a certain number of faulty nodes was placed probabilistically according to the described fault injection technique. The quality of output stands for the probability of receiving the expected output and is based on the maximum number of different bits (the Hamming distance) between the expected patterns and the produced pattern, whereby correct classification is still guaranteed. This experiment produced a plot of the neural network fault tolerance against the number of faulty nodes, by which the network reliability could then be judged.

Fig. 5.15 shows the effect of faults injection while evaluating a set of 20 patterns. We presented the damage volume in percentage to the all possible damages in order to scale the amount of damage with the size of the physical implementation. The figure shows that performance degradation is in some sense graceful. According to the plot, 5% faulty nodes guarantees 60% correct output and 10% faulty nodes reduces the probability of the correct result to 50%. A network with 2% faulty nodes produces the correct result with a probability of 90%. Fig. 5.16 presents the zoomed area of Fig. 5.15.

Although the experiments were performed simulating comparatively small networks (about 2000 neurons) with a small training set, the results are considered indicative for large networks with large training sets due to the scalable nature of calculations.

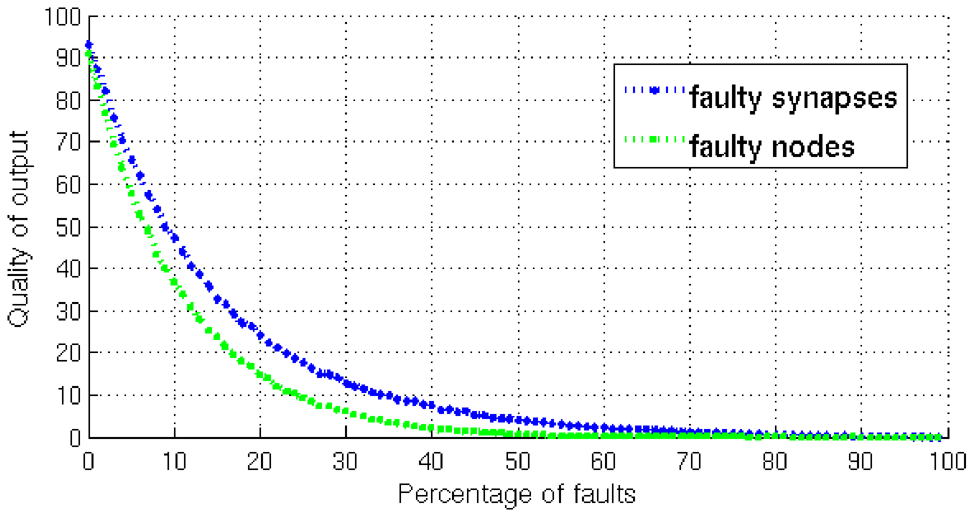


FIGURE 5.15: The quality of output against the amount of faulty neural network's nodes while recalling pre-learned 20 patterns using the network of 2000 neurons.

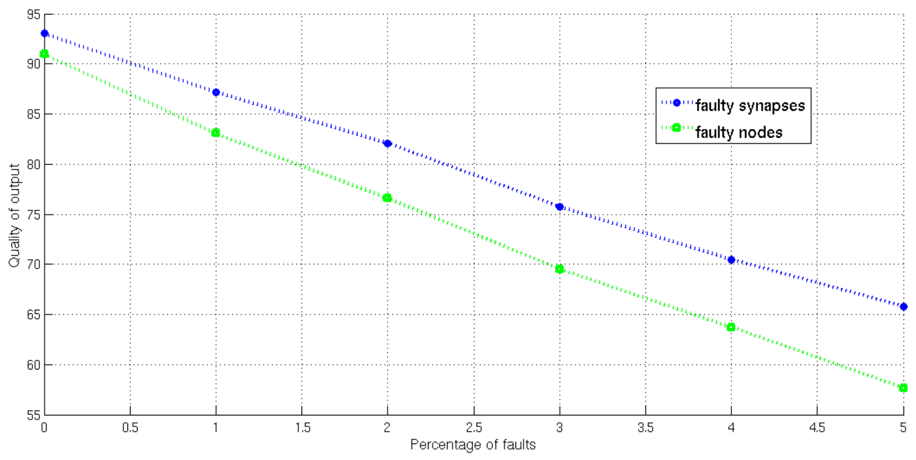


FIGURE 5.16: The quality of output against the amount of faulty neural network's nodes while recalling pre-learned 20 patterns using the network of 2000 neurons (zoomed version of Fig. 5.15).

5.6 Summary

At the beginning of this chapter we discussed various ways of neural network mapping on the distributed hardware. A mapping scheme is a very common issue in parallel distributed processing and usually is a trade-off between computation and communication and is mostly dependent on architecture of a network. We overviewed possible exploiting of a topology in order to optimise the communication load and provide additional performance gain.

The potential gain offered by an asynchronous versus synchronous time management was highlighted as well as the challenges caused by concurrent execution of events on asynchronous distributed processors were mentioned. These challenges require applying of specific synchronisation features to maintain the equal global time.

We presented in details an efficient implementation, design and operation of the neural network simulator. The structure and detailed performance analysis is also presented as well as investigation of the simulator's dependence on various neural network parameters and learning coefficients.

An efficient protocol for local communication between neurons while modelling a neural network on a parallel distributed hardware has been used. By using it, all neural data can be localised on the neuron simulating processor and only a necessary portion of data sharing occurs for the learning purposes. We also developed a fault-tolerant mechanism, immune to processor failure and communication problems, which can take place during a learning phase on a parallel distributed hardware.

Chapter 6

Conclusions and future work

6.1 Thesis Summary

This work contributes to the study of biological neural networks, in particular with respect to the learning mechanism. It explores the biologically plausible algorithm along with software implementation for efficient parallel simulation of neural networks on a distributed hardware.

The main focus is on biologically credible learning, where the learning process based on the weight adaptation depends exceptionally upon the variables, stored locally at either a synapse or the neurons, involved in the spike transmission.

Chapter 2 introduced the nature of a biological neuron, the building block of a neural network. It explained how the neural dynamics of biological neurons and their collective activity contribute to the information processing mechanism and the resultant collective intelligent behaviour. Also it overviewed the efforts to capture neural dynamics by using mathematical models, which help to reproduce the intelligent behaviour within a computer simulation. Later, it presented the importance of the synaptic plasticity mechanism to learning and memory: the ability of a synapse to change in strength in response to spike transmission activity over it.

Chapter 3 explained the different mathematical models of neurons, outlining their benefits and downsides and selected the best candidates for large-scale network simulation. It introduced the contemporary neural network modelling theories, strategies and techniques. They are used by engineers and scientists both for biologically plausible neural network simulations and for solving various technical problems, for which the conventional methods are inefficient (e.g. speech synthesis, computer vision).

Chapter 4 focused on the research work performed to devise a biologically plausible learning mechanism, capable of searching for correct patterns, storing and reproducing them later. It resembled the biological learning processes, found in the animal's brain. The chapter aimed to understand the essential computations that take place in the network of interconnected neurons, resembling actual biological neural networks. In the chapter the algorithm implementation was discussed, which is able to learn and recognise the learned patterns based only on a binary feedback signal. Investigation was performed on *activity paths* formation and their interference in the network. High level of activity paths interference directly influences learning speed and significantly limits capacity of stored patterns, causing self-destruction of a pre-learned information in an attempt to adjust the weights for storing new one.

As a case study, an agent was presented, which based on three proposed techniques is able to accumulate the vital experience from an ambient environment. For this purpose an algorithm was derived that changes the weights of a neuron network by determining the exact error, which the network makes on each example of a particular task in an unsupervised manner.

However, biological neural systems are intrinsically parallel by their nature. Therefore, in Chapter 5 the previously mentioned single processor program was extended to an asynchronous multi-processor system. Besides of increased computational reliability and scalability, such an approach raised the challenging side-effects, such as the cost of additional distribution-related communication and synchronisation of the parallel execution. Solving them, the efficient protocol for local communication between processors was used, which is immune to belatedly or out of order delivered spikes. We also developed a fault-tolerant mechanism, immune to processor failure and communication problems, which can take place during a learning phase on a parallel distributed hardware.

The simulator with the described features has been implemented during the project. The implementation details of the simulation software were illustrated along with the performance analysis and explanation of the weight changing dynamics.

6.2 Future Work

The approach proposed in this research is not perfect solution and does allow room for improvements. Different extensions or alterations are possible to enhance the performance of the learning process.

One of possibilities is to include more biologically known features into the presented model. As was mentioned in the thesis, the study on the behaviour of biological neural networks collected strong evidence that *time delays* of axonal signal and the refractory period of neuron plays an important role in the neuron dynamics. This would make possible usage of spike-timing information processing. There are believes that a network can efficiently control its activity level with the help of the delay mechanism. Another possible benefit is increased learning capacity because more advanced encoding mechanisms could be applied (e.g. population coding).

Several attempts have been made to incorporate *temporal concept* into the learning mechanism, while executing it on the asynchronous distributed hardware. When a final version of SpiNNaker will be implemented, our next step is to adapt the existing system onto it, where synchronisation mechanism is already realised on the hardware level and the issue would not cause any significant challenges.

If temporal concept were incorporated, it would be possible to include *different types of neurons* or a neuron model with various spiking patterns (such as the Izhikevich model, which can exhibit over 20 different spiking patterns of all known types of cortical neurons). It would be interesting to study the influence of different neuron dynamics on the ordered complex network and particularly the possible effect it will have on the stabilising the network activity. Another possible benefit is reduction of the active paths interference because of the different neuron dynamics. Certain types of neurons will store learned data tightly while another will be more easily adapting to the new data and thus loosely storing the learned one. It could also be a case that the LTD mechanism possibly does the job of searching for successful output patterns without strongly alternating the already learned input-output relations.

One more possible extension would be to refine the binary *feedback response*. In the current version it is a binary value, representing the correct and incorrect network output in reaction to some input. However, a more advanced relative measure of success can return more precise response, representing closeness to a desired output (such as the Hamming distance between two patterns). In such a way the learning speed can be significantly increased if a feedback signal is represented as a range of values.

References

[1] M. R. Villarreal, "Diagram of a typical myelinated vertebrate neuron." July 2007, available on: http://en.wikipedia.org/wiki/Neuron [accessed in January 25, 2013].

[2] G. Leonardo, "Schematic view of an idealized action potential," August 2006, available on: http://en.wikipedia.org/wiki/Action_potential [accessed in January 25, 2013].

[3] J. Bailey, "Towards the neurocomputer: an investigation of VHDL neuron models," Ph.D. dissertation, University of Southampton, February 2010.

[4] N. Mehrtash, D. Jung, H. H. Hellmich, T. Schönauer, V. T. Lu, and H. Klar, "Synaptic plasticity in spiking neural networks (SP(2)INN): a system approach," *IEEE Trans Neural Netw*, vol. 14, no. 5, pp. 980–992, 2003.

[5] G. Bi and M. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," *J Neurosci*, vol. 18, no. 24, pp. 10 464–10 472, December 1998.

[6] E. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE Trans Neural Netw*, vol. 15, no. 5, pp. 1063–1070, September 2004.

[7] ——, "Simple model of spiking neurons," *IEEE Trans Neural Netw*, vol. 14, no. 6, pp. 1569–1572, November 2003.

[8] M. Rudolph and A. Destexhe, "How much can we trust neural simulation strategies?" *Neurocomput*, vol. 70, no. 10-12, pp. 1966–1969, June 2007.

[9] T. Natschläger, "Networks of spiking neurons: A new generation of neural network models," in *Jenseits von Kunst*. Passagen Verlag, 1998.

[10] R. J. C. Bosman, W. A. van Leeuwen, and B. Wemmenhove, "Combining Hebbian and reinforcement learning in a minibrain model," *Neural Netw*, vol. 17, no. 1, pp. 29–36, January 2004.

[11] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve.” *J Physiol*, vol. 117, no. 4, pp. 500–544, August 1952.

[12] T. Carew and E. Kandel, “Acquisition and retention of long-term habituation in Aplysia: correlation of behavioral and cellular processes,” *Science*, vol. 182, no. 4117, pp. 1158–60, December 1973.

[13] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. Bower, M. Diesmann, A. Morrison, P. Goodman, F. Harris, M. Zirpe, T. Natschlger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. Davison, El, and A. Destexhe, “Simulation of networks of spiking neurons: A review of tools and strategies,” *J Comp Neurosci*, July 2007.

[14] D. Brettle and E. Niebur, “Detailed parallel simulation of a biological neuronal network,” *IEEE Comput Sci Eng*, vol. 1, no. 4, pp. 31–43, 1994.

[15] W. W. Lytton, A. Destexhe, and T. J. Sejnowski, “Control of slow oscillations in the thalamocortical neuron: A computer model,” *J Neurosci*, February 1996.

[16] P. C. Bush and T. J. Sejnowski, “Inhibition synchronizes sparsely connected cortical neurons within and between columns in realistic network models,” *J Comp Neurosci*, June 1996.

[17] W. Gerstner and W. M. Kistler, *Spiking Neuron Models.* Cambridge University Press, August 2002.

[18] M. Hines, J. W. Moore, and T. Carnevale, *NEURON*, NEURON Development Team, 2007.

[19] D. Beeman and J. M. Bower, *The GENESIS Simulator*, GENESIS Development Team, 2007.

[20] L. A. Plana, J. Bainbridge, S. Furber, S. Salisbury, Y. Shi, and J. Wu, “An On-Chip and Inter-Chip Communications Network for the SpiNNaker Massively-Parallel Neural Net Simulator,” in *NOCS.* IEEE Computer Society, April 2008, pp. 215–216.

[21] A. G. Guggisberg, S. S. Dalal, A. M. Findlay, and S. S. Nagarajan, “High-frequency oscillations in distributed neural networks reveal the dynamics of human decision making,” *Frontiers in Human Neuronscience*, March 2008.

[22] H. Plesser, J. Eppler, A. Morrison, M. Diesmann, and M.-O. Gewaltig, "Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers," in *Euro-Par 2007 Parallel Processing*, August 2007, pp. 672–681.

[23] G. J. Stuart and B. Sakmann, "Active propagation of somatic action potentials into neocortical pyramidal cell dendrites." *Nature*, vol. 367, no. 6458, pp. 69–72, January 1994.

[24] E. M. Izhikevich, J. A. Gally, and G. M. Edelman, "Spike-timing dynamics of neuronal groups," *Cerebral Cortex*, vol. 14, pp. 933–944, August 2004.

[25] I. Segev, M. Rapp, Y. Manor, and Y. Yarom, *Analog and digital processing in single nerve cells: dendritic integration and axonal propagation.* San Diego, CA, USA: Academic Press Professional, Inc., 1992, pp. 173–198.

[26] D. Ferster and S. Lindström, "An intracellular analysis of geniculo-cortical connectivity in area 17 of the cat." *J Physiol*, vol. 342, pp. 181–215, 1983.

[27] H. A. Swadlow, "Efferent neurons and suspected interneurons in motor cortex of the awake rabbit: axonal properties, sensory receptive fields, and subthreshold synaptic inputs." *J Neurophysiol*, vol. 71, pp. 437–53, 1994.

[28] ——, "Physiological properties of individual cerebral axons studied in vivo for as long as one year," *J Neurophysiol*, vol. 54, no. 5, pp. 1346–1362, 1985.

[29] M. Salami, C. Itami, T. Tsumoto, and F. Kimura, "Change of conduction velocity by regional myelination yields constant latency irrespective of distance between thalamus and cortex." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 100, pp. 6174–6179, 2003.

[30] I. Segev and M. London, "Untangling Dendrites with Quantitative Models," *Science*, vol. 290, no. 5492, pp. 744–750, October 2000.

[31] D. Purves, G. J. Augustine, D. Fitzpatrick, W. C. Hall, A.-S. Lamantia, J. O. McNamara, and S. M. Williams, *Neurosci*, 3rd ed. Sinauer Associates, 2004.

[32] T. Toyoizumi, J.-P. Pfister, K. Aihara, and W. Gerstner, "Generalized Bienenstock-Cooper-Munro rule for spiking neurons that maximizes information transmission," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 14, pp. 5239–5244, April 2005.

[33] M. Tsodyks, A. Uziel, and H. Markram, "Synchrony generation in recurrent networks with frequency-dependent synapses," *J Neurosci*, vol. 20, p. 50, January 2000.

[34] R. S. Zucker and W. G. Regehr, "Short-term synaptic plasticity," *Annu Rev Physiol*, vol. 64, pp. 355–405, November 2003.

[35] T. V. P. Bliss and T. Lomo, "Long-lasting potentation of synaptic transmission in the dendate area of anaesthetized rabbit following stimulation of the perforant path." *J Physiol*, vol. 232, no. 2, pp. 551–356, July 1973.

[36] I. Antonov, I. Antonova, E. R. Kandel, and R. D. Hawkins, "Activity-dependent presynaptic facilitation and Hebbian LTP are both required and interact during classical conditioning in aplysia," *Neuron*, vol. 37, no. 1, pp. 135–147, January 2003.

[37] G. Bi and M. Poo, "Synaptic modification by correlated activity: Hebb's postulate revisited." *Annu Rev Neurosci*, vol. 24, no. 1, pp. 139–166, 2001.

[38] L. F. Abbott and S. B. Nelson, "Synaptic plasticity: taming the beast." *Nature Neuroscience*, vol. 3, pp. 1178–1183, November 2000.

[39] T. J. Sejnowski, "Statistical constraints on synaptic plasticity," *J Theor Biol*, vol. 69, no. 2, pp. 385–389, November 1977.

[40] E. L. Bienenstock, L. N. Cooper, and P. W. Munro, "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex," *J Neurosci*, vol. 2, no. 1, pp. 32–48, January 1982.

[41] E. Oja, "Simplified neuron model as a principal component analyzer," *J Math Biol*, vol. 15, pp. 267–273, 1982.

[42] T. D. Sanger, "Optimal unsupervised learning in a single-layer linear feed-forward neural network," *Neural Networks*, vol. 2:6, pp. 459 – 473, 1989.

[43] S. Song and L. F. Abbott, "Cortical development and remapping through spike timing-dependent plasticity." *Neuron*, vol. 32, pp. 339–350, 2001.

[44] H. Markram, J. Lübke, M. Frotscher, and B. Sakmann, "Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs." *Science*, vol. 275, no. 5297, pp. 213–215, January 1997.

[45] C. D. Meliza and Y. Dan, "Receptive-field modification in rat visual cortex induced by paired visual stimulation and single-cell spiking," *Neuron*, vol. 49, no. 2, pp. 183–189, January 2006.

[46] R. S. Zucker, "Calcium- and activity-dependent synaptic plasticity." *Curr Opin Neurobiol*, vol. 9, no. 3, pp. 305–313, June 1999.

[47] W. Senn, M. Tsodyks, and H. Markram, "An algorithm for modifying neurotransmitter release probability based on pre- and postsynaptic spike timing," *Neural Comput*, vol. 13, no. 1, pp. 35–67, January 2001.

[48] U. R. Karmarkar and D. V. Buonomano, "A model of spike-timing dependent plasticity: One or two coincidence detectors?" *J Neurophysiol*, vol. 88, no. 1, pp. 507–513, 2002.

[49] R. Kempter, W. Gerstner, and J. Hemmen, "Hebbian learning and spiking neurons," *Physical Review E*, vol. 59, no. 4, pp. 4498–4514, April 1999.

[50] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neurosci*, vol. 3, no. 9, pp. 919–926, September 2000.

[51] M. van Rossum, G. Bi, and G. Turrigiano, "Stable Hebbian learning from Spike Timing-Dependent Plasticity," *J Neurosci*, vol. 20, no. 23, pp. 8812–8821, December 2000.

[52] T. Nowotny, V. P. Zhigulin, A. I. Selverston, H. D. I. Abarbanel, and M. I. Rabinovich, "Enhancement of synchronization in a hibrid neural circuit by Spike-Time-Dependent Plasticity." *J Neurosci*, vol. 23, no. 30, pp. 9776–9785, October 2003.

[53] G. Bi and J. Rubin, "Timing in synaptic plasticity: from detection to integration." *Trends Neurosci*, vol. 28, no. 5, pp. 222–228, May 2005.

[54] A. Turing, "On computable numbers, with an application to the entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. s2-42, pp. 230–265, 1937.

[55] W. Mcculloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biology*, vol. 5, no. 4, pp. 115–133, December 1943.

[56] M. Minsky and S. Papert, *Perceptron: an introduction to computational geometry.* MIT press, 1969.

[57] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." *J Physiol*, vol. 160, pp. 106–154, January 1962.

[58] C. Morris and H. Lecar, "Voltage oscillations in the barnacle giant muscle fiber," *Biophys J*, vol. 35, no. 1, pp. 193–213, July 1981.

[59] N. Fourcaud-Trocme, D. Hansel, C. van Vreeswijk, and N. Brunel, "How spike generation mechanisms determine the neuronal response to fluctuating inputs." *J Neurosci*, vol. 23, no. 37, pp. 11 628–11 640, December 2003.

[60] G. D. Smith, C. L. Cox, M. S. Sherman, and J. Rinzel, "Fourier analysis of sinusoidally driven thalamocortical relay neurons and a minimal integrate-and-fire-or-burst model," *J Neurophysiol*, vol. 83, no. 1, pp. 588–610, 2000.

[61] E. Izhikevich, "Resonate-and-fire neurons." *Neural Netw*, vol. 14, no. 6-7, pp. 883–894, September 2001.

[62] B. Ermentrout, "Type I membranes, phase resetting curves, and synchrony," *Neural Comput*, vol. 8, no. 5, pp. 979–1001, 1996.

[63] R. Fitzhugh, "Impulses and physiological states in theoretical models of nerve membrane," *Biophys J*, vol. 1, no. 6, pp. 445–466, July 1961.

[64] J. Nagumo, S. Arimoto, and S. Yoshizawa, "An active pulse transmission line simulating nerve axon," *Proceedings of the IRE*, vol. 50, no. 10, pp. 2061–2070, October 1962.

[65] H. A. Swadlow, A. G. Gusev, and T. Bezdudnaya, "Activation of a cortical column by a thalamocortical impulse," *J Neurosci*, vol. 22, no. 17, pp. 7766–7773, September 2002.

[66] J. C. Horton and D. L. Adams, "The cortical column: a structure without a function." *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, vol. 360, no. 1456, pp. 837–862, April 2005.

[67] V. B. Mountcastle, "An organizing principle for cerebral function: the unit model and the distributed system," in *The Mindful Brain.* Cambridge, Mass.: MIT Press, 1978.

[68] ——, "Introduction. Computatation in cortical columns." *Cerebral cortex*, vol. 13, no. 1, pp. 2–4, January 2003.

[69] D. P. Buxhoeveden and M. F. Casanova, “The minicolumn hypothesis in neuroscience,” *Brain*, vol. 125, no. 5, pp. 935–951, May 2002.

[70] V. B. Mountcastle, “The columnar organization of the neocortex.” *Brain*, vol. 120, no. 4, pp. 701–722, April 1997.

[71] S. M. Bohte, E. M. Bohte, H. L. Poutr, and J. N. Kok, “Unsupervised clustering with spiking neurons by sparse temporal coding and multi-layer RBF networks,” *IEEE Trans Neural Netw*, vol. 13, pp. 426–435, 2002.

[72] R. Vanrullen, “The power of the feed-forward sweep,” *Adv Cogn Psychol*, vol. 3, no. 1–2, pp. 167–176, July 2008.

[73] V. A. Lamme and P. R. Roelfsema, “The distinct modes of vision offered by feedforward and recurrent processing.” *Trends Neurosci*, vol. 23, no. 11, pp. 571–579, 2000.

[74] M. W. Oram and D. I. Perrett, “Time course of neural responses discriminating different views of the face and head,” *J Neurophysiol*, vol. 68, no. 1, pp. 70–84, 1992.

[75] M. J. Tovee, “Neuronal processing. How fast is the speed of thought?” *Current Biology*, vol. 4, no. 12, pp. 1125–1127, 1994.

[76] P. V. Yee and S. Haykin, *Regularized radial basis functional networks: theory and applications.* New York: John Wiley & Sons, Inc., 2001.

[77] F. Rosenblatt, *Principles of neurodynamics: perceptrons and the theory of brain mechanisms.* NewYork: Spartan, 1962.

[78] J. L. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[79] M. I. Jordan, *Chapter 25. Serial order: A parallel distributed processing approach.* Elsevier, 1997, vol. 121, pp. 471–495.

[80] T. Kohonen, *Self-organizing maps*, 3rd ed., ser. Springer series in information sciences. Springer, December 2001.

[81] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–2558, April 1982.

[82] J. L. McClelland and D. E. Rumelhart, “An interactive activation model of context effects in letter perception. Part 1: an account of basic findings,” pp. 401–436, 1988.

[83] B. Doyon, B. Cessac, M. Quoy, and M. Samuelides, “Mean-field equations, bifurcation map and chaos in discrete time, continuous state, random neural networks,” *Acta Biotheoretica*, vol. 43, pp. 169–175, June 1995.

[84] H. Jaeger, “The “echo state” approach to analysing and training recurrent neural networks,” German National Research Center for Information Technology, Tech. Rep. GMD Report 148, 2001.

[85] W. Maass, T. Natschläger, and H. Markram, “Real-time computing without stable states: A new framework for neural computation based on perturbations,” *Neural Comput*, vol. 14, no. 11, pp. 2531–2560, November 2002.

[86] J. J. Steil, “Backpropagation-decorrelation: online recurrent learning with O(N) complexity,,” in *IJCNN*, vol. 2, July 2004, pp. 843–848.

[87] H. Jaeger, “Short term memory in echo state networks,” GMD - German National Research Institute for Computer Science, GMD-Report 152, 2002.

[88] M. Mattia and P. D. Giudice, “Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses,” *Neural Comput*, vol. 12, no. 10, pp. 2305–2329, October 2000.

[89] O. Rochel and D. Martinez, “An event-driven framework for the simulation of networks of spiking neurons,” in *ESANN*, April 2003, pp. 295–300.

[90] W. R. Softky, “Simple codes versus efficient codes.” *Curr Opin Neurobiol*, vol. 5, no. 2, pp. 239–247, 1995.

[91] M. Raul and I. Iosif, “The “Neocortex” neural simulator. A modern design,” in *INES*, September 2004, pp. 99–104.

[92] G. Lee and N. H. Farhat, “The double queue method: a numerical method for integrate-and-fire neuron networks,” *Neural Netw*, pp. 921–932, 2001.

[93] M. Shelley and L. Tao, “Efficient and accurate time-stepping schemes for integrate-and-fire neuronal networks,” *J Comput Neurosci*, vol. 11, no. 2, pp. 111–119, October 2001.

[94] L. Watts, “Event-driven simulation of networks of spiking neurons,” in *NIPS*, vol. 6. MIT Press, 1993, pp. 927–934.

[95] T. Makino, “A discrete-event neural network simulator for general neuron models,” *Neural Computing and Applications*, vol. 11, no. 3, pp. 210–223, June 2003.

[96] N. Raghuvanshi, R. Narain, and M. C. Lin, “Efficient and accurate sound propagation using adaptive rectangular decomposition,” *IEEE Tran Vis Comp Graph*, vol. 15, pp. 789–801, October 2009.

[97] A. Mouraud and D. Puzenat, “Simulation of large spiking neural networks on distributed architectures, the “Damned” simulator,” *EANN*, no. 5, pp. 359–370, 2009.

[98] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain,” *Psychol Rev*, vol. 65, pp. 386–408, 1958.

[99] B. Widrow and M. Hoff, *Adaptive switching circuits.* Cambridge, MA, USA: MIT Press, 1988.

[100] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, July 1993.

[101] V. N. Vapnik and A. Y. Chervonenkis, “On the uniform convergence of relative frequencies of events to their probabilities,” *Theory of Probability and its Applications*, vol. 16, no. 2, pp. 264–280, 1971.

[102] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, vol. 8, pp. 229–256, 1992.

[103] P. Marbach and J. N. Tsitsiklis, “Simulation-based optimization of Markov reward processes,” in *IEEE Conf Decision and Control*, Tampa, FL , USA, 1998, pp. 2698–2703.

[104] D. Prokhorov, “Adaptive critic designs: A case study for neurocontrol,” *Neural Netw*, vol. 8, no. 9, pp. 1367–1372, 1995.

[105] W. Potjans, A. Morrison, and M. Diesmann, “A spiking neural network model of an actor-critic learning agent,” *Neural Comput*, vol. 21, no. 2, pp. 301–339, February 2009.

[106] A. G. Barto, R. S. Sutton, and C. W. Anderson, *Artificial neural networks.* Piscataway, NJ, USA: IEEE Press, 1990.

[107] G. A. Carpenter and S. Grossberg, "The handbook of brain theory and neural networks," M. A. Arbib, Ed. Cambridge, MA, USA: MIT Press, 1998, ch. Adaptive resonance theory (ART), pp. 79–82.

[108] R. Fujimoto, "Parallel simulation: parallel and distributed simulation systems," in *Winter Simulation Conference*, 2001, pp. 147–157.

[109] A. Mouraud and H. Paugam-Moisy, "Damned, un simulateur parallèle et événementiel, pour rèseaux de neurones impulsionnels," in *Actes NeuroComp 2006*, October 2006, pp. 120–123.

[110] U. Roth, A. Jahnke, and H. Klar, "On-line Hebbian learning for spiking neurons: Architecture of the weight-unit of NESPINN," in *Artificial Neural Networks ICANN'97*, ser. Lecture Notes in Computer Science, W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, Eds. Springer Berlin/Heidelberg, 1997, vol. 1327, pp. 1217–1222.

[111] J. Schemmel, K. Meier, and E. Mueller, "A new VLSI model of neural microcircuits including spike time dependent plasticity," *IJCNN*, pp. 1711–1716, 2004.

[112] H. Markram, "The Blue Brain project," in *ACM/IEEE Supercomputing.* New York, NY, USA: ACM, 2006.

[113] S. Furber and A. Brown, "Biologically-inspired massively-parallel architectures - computing beyond a million processors," in *ICACSD*, July 2009, pp. 3–12.

[114] M. Bumble and L. Coraor, "Implementing parallelism in random discrete event-driven simulation," in *in Lecture Notes in Computer Science 1388, Parallel and Distributed Processing.* Springer, 1998, pp. 418–427.

[115] H. H. Hellmich and H. Klar, "SEE: a concept for an FPGA based emulation engine for spiking neurons with adaptive weights," in *WSEAS Conf. on Neural Netw and Applic*, vol. 25-27, 2004, pp. 930–935.

[116] J. Bailey, P. Wilson, A. Brown, and J. Chad, "Behavioural simulation and synthesis of biological neuron systems using synthesizable VHDL," *Neurocomput*, vol. 74, no. 14-15, pp. 2393–2406, July 2011.

[117] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Fast neural networks without multipliers," *IEEE Trans Neural Netw*, vol. 4, no. 1, pp. 53–62, January 1993.

[118] B. Noory and V. Groza, "A reconfigurable approach to hardware implementation of neural networks," *Canadian Conference on Electrical and Computer Engineering*, vol. 3, pp. 1861 – 1864, May 2003.

[119] Y. Taright and M. Hubin, "FPGA implementation of a multilayer perceptron neural network using VHDL," *Signal Processing Proceedings*, vol. 2, pp. 1311 – 1314, October 1998.

[120] S. Himavathi, D. Anitha, and A. Muthuramalingam, "Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization," *IEEE Trans Neural Netw*, vol. 3, pp. 880–888, May 2007.

[121] K. Oh and K. Jung, "GPU implementation of neural networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311–1314, June 2004.

[122] M. A. Bhuiyan, V. K. Pallipuram, and M. C. Smith, "Acceleration of spiking neural networks in emerging multi-core and GPU architectures," in *IPDPS Workshops*, 2010, pp. 1–8.

[123] J. P. Tiesel and A. S. Maida, "Using parallel GPU architecture for simulation of planar I/F networks," in *IJCNN*. IEEE, June 2009, pp. 3118–3123.

[124] J. M. Nageswaran, N. D. Dutt, J. L. Krichmar, A. Nicolau, and A. V. Veidenbaum, "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors," *Neural Networks*, vol. 22, no. 5-6, pp. 791–800, August 2009.

[125] B. Han and T. Taha, "Acceleration of spiking neural network based pattern recognition on nvidia graphics processors," *Applied Optics*, vol. 49, pp. B83–B91, April 2010.

[126] A. Davison, D. Bruederle, J. Eppler, K. Jens, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "Pynn: a common interface for neuronal network simulators," *Front Neuroinformatics*, vol. 2, no. 11, January 2009.

[127] A. Hindmarsh and A. Taylor, *User Documentation for IDA, A Differential-Algebraic Equation Solver for Sequential and Parallel Computers*, Lawrence Livermore National Laboratory, 1999.

[128] M. Hines and N. Carnevale, "Expanding NEURON's repertoire of mechanisms with NMODL," *Neural Comput*, vol. 12, no. 5, pp. 995–1007, 2000.

[129] D. F. Bacon, J. Schwartz, and Y. Yemini, *MAIN Page - NEST*, NEST Development Team, 2007.

[130] ——, “Nest: A network simulation and prototyping tool,” in *Proceedings of the USENIX Winter 1988 Technical Conference.* Berkeley, CA: USENIX Association, 1988, pp. 71–78.

[131] F. Harris and P. Goodman, *NCS Documentation*, NCS Development Team, 2007.

[132] E. C. Wilson, “Parallel implementation of a large scale biologically realistic neocortical neural network simulator,” Master’s thesis, University of Nevada, November 2001.

[133] H. Markram and W. Maass, *Neural Micro circuits*, CSIM Development Team, 2007.

[134] P. Hammarlund, O. Ekeberg, T. Wilhelmsson, and A. Lansner, “Large neural network simulations on multiple hardware platforms,” in *CNS.* New York, NY, USA: Plenum Press, 1997, pp. 919–923.

[135] O. Rochel, *Mvaspike*, Mvaspike Development Team, 2007.

[136] A. Delorme and S. J. Thorpe, “Spikenet: an event-driven simulation package for modelling large networks of spiking neurons,” *Neural Networks*, vol. 14, no. 4, pp. 613–627, November 2003.

[137] M. Versace, H. Ames, J. Léveillé, B. Fortenberry, and A. Gorchetchnikov, “Kinness: A modular framework for computational neuroscience,” *Neuroinformatics*, vol. 6, no. 4, pp. 291–309, 2008.

[138] D. Rohde, “LENS: The light, efficient network simulator,” *Software package available at www. cs. cmu. edu/dr/Lens/*, 1999.

[139] I. Fischer, F. Hennecke, C. Bannes, and A. Zell, *JavaNNS User Manual, Version 1.1*, Center for Bioinformatics, University of Tübingen, 2002.

[140] S.-H. Lee and K. Skadron, “Highly parallel implementation of neurojet using GPUs,” School of Engineering and Applied Science-University of Virginia, Tech. Rep. STS 4020, 2010.

[141] T. Stewart, B. Tripp, and C. Eliasmith, “Python scripting in the Nengo simulator,” *Front Neuroinformatics*, vol. 3, no. 7, March 2009.

[142] U. Bhalla, S. Ray, N. Dudani, S. George, A. Gilra, and G. Harsharani, “Multiscale models in Moose: interoperability and standardization,” *Front Neuroinformatics*, vol. 1, 2011.

[143] S. Nissen, "Implementation of a Fast Artificial Neural Network Library (fann)," *Report, Department of Computer Science University of Copenhagen (DIKU)*, vol. 31, 2003.

[144] F. Zubler and R. Douglas, "A framework for modeling the growth and development of neurons and networks," *Front Comput Neurosci*, vol. 3, 2009.

[145] S. Furber and S. Temple, "Neural systems engineering," in *Computational Intelligence: A Compendium*, 2008, pp. 763–796.

[146] D. J. Felleman and D. C. V. Essen, "Distributed hierarchical processing in the primate cerebral cortex," *Cerebral Cortex*, pp. 1–47, February 1991.

[147] M. Bouten, A. Engel, A. Komoda, and R. Serneels, "Quenched versus annealed dilution in neural networks," *Journal of Physics A: Mathematical and General*, vol. 23, no. 20, p. 4643, 1990.

[148] R. Morris, "Developments of a water-maze procedure for studying spatial learning in the rat." *J Neurosci Methods*, vol. 11, no. 1, pp. 47–60, 1984.

[149] C. V. Vorhees and M. T. Williams, "Morris water maze: procedures for assessing spatial and related forms of learning and memory." *Nature protocols*, vol. 1, no. 2, pp. 848–858, 2006.

[150] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, J. Hertz, A. Krogh, and R. Palmer, Eds. Addison-Wesley, 1991, vol. 1.

[151] B. Mueller, J. Reinhardt, and M. Strickland, *Neural networks: an introduction*, 2nd ed. Berlin: Springer, 1995.

[152] D. O. Hebb, "The organization of behavior," pp. 43–54, 1988.

[153] P. R. Montague, P. Dayan, C. Person, and T. J. Sejnowski, "Bee foraging in uncertain environments using predictive hebbian learning." *Nature*, vol. 377, no. 6551, pp. 725–728, October 1995.

[154] P. Montague, P. Dayan, and T. Sejnowski, "A framework for mesencephalic dopamine systems based on predictive hebbian learning," *J Neurosci*, vol. 16, no. 5, pp. 1936–1947, March 1996.

[155] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, new ed. New York: Wiley, June 1949.

[156] B. Porr and F. Woergoetter, "Fast heterosynaptic learning in a robot food retrieval task inspired by the limbic system," *Biosystems*, vol. 89, no. 1-3, pp. 294 – 299, June 2007.

[157] D. O. Hebb, "Conditioned and unconditioned reflexes and inhibition," *Master's thesis, McGill University*, 1932.

[158] A. Barto, *Reinforcement learning: The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. Cambridge, MA: MIT Press, 1995.

[159] R. Sutton and A. Barto, *Reinforcement learning: an introduction.* MIT Press, 1998.

[160] W. Schultz, P. Dayan, and P. R. Montague, "A neural substrate of prediction and reward," *Science*, vol. 275, no. 5306, pp. 1593–1599, March 1997.

[161] M. Pessiglione, B. Seymour, G. Flandin, R. J. Dolan, and C. D. Frith, "Dopamine-dependent prediction errors underpin reward-seeking behaviour in humans," *Nature*, vol. 442, no. 7106, pp. 1042–1045, August 2006.

[162] R. S. Sutton, "Reinforcement learning: Past, present and future," in *SEAL*, 1998, pp. 195–197.

[163] P. Bak and K. Sneppen, "Punctuated equilibrium and criticality in a simple model of evolution," *Phys Rev Lett*, vol. 71, pp. 4083–4086, Dec 1993.

[164] J. Wakeling and P. Bak, "Intelligent systems in the context of surrounding environment," October 2001, published in Physical Review E 64, 051920.

[165] J. Bedaux and W. A. Van Leeuwen, "Biologically inspired learning in a layered neural net," *Physica A: Statistical Mechanics and its Applications*, vol. 335, no. 1–2, April 2004.

[166] J. L. Krichmar, A. K. Seth, D. A. Nitz, J. G. Fleischer, and G. M. Edelman, "Spatial navigation and causal analysis in a brain-based device modeling cortical-hippocampal interactions," *Neuroinformatics*, vol. 3, no. 3, pp. 197–222, September 2005.

[167] P. Bak and D. R. Chialvo, "Adaptive learning by extremal dynamics and negative feedback," *Phys Rev E*, vol. 63, no. 3, p. 031912, February 2001.

[168] J. Wakeling, "Order-disorder transition in the Chialvo-Bak minibrain controlled by network geometry," pp. 561–569, July 2003, in Physica A 325.

[169] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.

[170] R. Felderman and L. Kleinrock, "An upper bound on the improvement of asynchronous versus synchronous distributed processing," in *The Society for Computer Simulation, Distributed Simulation 1990*, D. Nicol, Ed., January 1990, pp. 131–136.

[171] K. M. Chandy and J. Misra, "Distributed simulation: A case study in design and verification of distributed programs," *IEEE Trans Software Engineering*, vol. SE-5, no. 5, pp. 440–452, 2006.

[172] D. R. Jefferson, "Virtual time," *ACM Trans Program Lang Syst*, vol. 7, pp. 404–425, July 1985.

[173] C. Koch, *Biophysics of Computation: Information Processing in Single Neurons (Computational Neuroscience).* Oxford University Press, 1998.

[174] G. Shepherd, *The Synaptic Organization of the Brain*, 4th ed. Oxford University Press, 1998.

[175] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI-The Complete Reference, Volume 1: The MPI Core*, 2nd ed. Cambridge, MA, USA: MIT Press, 1998.

[176] H. Ong and P. A. Farrell, "Performance comparison of LAM/MPI, MPICH, and MVICH on a Linux cluster connected by a gigabit ethernet network," in *Annual Linux Showcase and Conference*, 2000, pp. 10–14.

[177] C. Chiu, K. Mehrotra, C. K. Mohan, and S. Ranka, "Training techniques to obtain fault tolerant neural networks," in *Proc. FTCS-24*, 1994, pp. 360–369.