ABSTRACT

Title of dissertation: SELF-ORGANIZING MAP NEURAL ARCHITECTURES BASED ON LIMIT CYCLE ATTRACTORS

Di-Wei Huang, Doctor of Philosophy, 2016

Dissertation directed by: Professor James A. Reggia
Department of Computer Science

Recent efforts to develop large-scale neural architectures have paid relatively little attention to the use of self-organizing maps (SOMs). Part of the reason is that most conventional SOMs use a static encoding representation: Each input is typically represented by the fixed activation of a single node in the map layer. This not only carries information in an inefficient and unreliable way that impedes building robust multi-SOM neural architectures, but it is also inconsistent with rhythmic oscillations in biological neural networks. Here I develop and study an alternative encoding scheme that instead uses limit cycle attractors of multi-focal activity patterns to represent input patterns/sequences. Such a fundamental change in representation raises several questions: Can this be done effectively and reliably? If so, will map formation still occur? What properties would limit cycle SOMs exhibit? Could multiple such SOMs interact effectively? Could robust architectures based on such SOMs be built for practical applications?

The principal results of examining these questions are as follows. First, conditions are established for limit cycle attractors to emerge in a SOM through self-organization

when encoding both static and temporal sequence inputs. It is found that under appropriate conditions a set of learned limit cycles are stable, unique, and preserve input relationships. In spite of the continually changing activity in a limit cycle SOM, map formation continues to occur reliably. Next, associations between limit cycles in different SOMs are learned. It is shown that limit cycles in one SOM can be successfully retrieved by another SOM's limit cycle activity. Control timings can be set quite arbitrarily during both training and activation. Importantly, the learned associations generalize to new inputs that have never been seen during training. Finally, a complete neural architecture based on multiple limit cycle SOMs is presented for robotic arm control. This architecture combines open-loop and closed-loop methods to achieve high accuracy and fast movements through smooth trajectories. The architecture is robust in that disrupting or damaging the system in a variety of ways does not completely destroy the system. I conclude that limit cycle SOMs have great potentials for use in constructing robust neural architectures.

SELF-ORGANIZING MAP NEURAL ARCHITECTURES BASED ON LIMIT CYCLE ATTRACTORS

by

Di-Wei Huang

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2016

Advisory Committee:

Professor James A. Reggia, Chair
Professor J. Yiannis Aloimonos
Professor Rodolphe J. Gentili,
Professor Donald R. Perlis
Professor Gerald S. Wilkinson, Dean's Representative

Acknowledgments

This work was supported by ONR award N000141310597.

I wish to thank Professor James Reggia for the invaluable help he has offered as my advisor. I cannot recall a moment with him when he was not caring, encouraging, and inspiring. It has been a great pleasure working with him, and I cannot imagine having a better mentor to help me through this long journey*.

This dissertation could not have been completed without the help by Professor Rodolphe Gentili as a co-advisor. As a computer science student, I have learned a lot from his expertise in neurobiology and kinesiology. He has always been a great person to work with.

I would like to thank Professor Yiannis Aloimonos, Professor Donald Perlis, and Professor Gerald Wilkinson for their guidance and support as my dissertation committee members. Their insightful comments strengthened this dissertation as well as opened up many future possibilities.

I would also like to thank Garrett Katz for all the thought-provoking discussions and for listening to me complain.

Finally, I would like to thank my mother for always being loving, supportive, and understanding during my hardest time, and for constantly worrying about me "not wanting to get a *real job*". I'm getting there, Mom.

*No bribes were involved in exchange for these words.

Table of Contents

List of Tables vi

List of Figures vii

1 Introduction 1
1.1 Goals and Specific Aims 5
1.2 Overview 8

2 Background 9
2.1 Basic SOMs 9
2.2 SOMs for Temporal Sequence Processing 17
2.3 Multi-SOM Neural Architectures 24

3 Learning Limit Cycle Representations in SOMs 30
3.1 A SOM Model with Limit Cycle Dynamics 32
3.2 Representing Binary Sequences 39
3.2.1 Methods 40
3.2.2 Attractor Formation 43
3.2.3 Attractor Stability 45
3.2.4 Representation Uniqueness 46
3.2.5 Effects of Training 49
3.2.6 Oscillatory Activation: An Example 49
3.2.7 Map Formation 51
3.3 Representing Continuous 2D Space 55
3.3.1 Methods 58
3.3.2 Limit Cycle Formation 59
3.3.3 Limit Cycles Properties 60
3.3.4 Map Formation 63
3.4 Summary and Discussion 65

4 Learning Associations Between Limit Cycles in Multi-SOM Architectures 67
4.1 Retrieving Limit Cycles Using Static Activity 69
4.1.1 Methods 69
4.1.2 Results 74

4.2 Retrieving Limit Cycles Using Limit Cycles: A Dual-Route Approach 76
4.2.1 Methods . . . 77
4.2.2 Results . . . 81
4.3 Retrieving Limit Cycles Using Limit Cycles: Working with Continuous Spaces . . . 82
4.3.1 Arm Model . . . 84
4.3.2 Neural Architecture . . . 84
4.3.3 Training Methods . . . 88
4.3.3.1 Stage 1: Individual Map Training . . . 88
4.3.3.2 Stage 2: Joint Command Output . . . 90
4.3.3.3 Stage 3: Inter-Modality Associations . . . 91
4.3.3.4 Learning Associations Between Sequences with Multi-Winner Patterns . . . 93
4.3.4 Experimental Methods . . . 95
4.3.5 Map and Limit Cycle Formation . . . 97
4.3.6 Convergence of Pattern Alignment . . . 98
4.3.7 Spatial Error . . . 100
4.3.8 Sensitivity to Timing Parameters . . . 101
4.4 Summary and Discussion . . . 102

5 A Limit-Cycle SOM Architecture for Stable Arm Control 106
5.1 Overview . . . 109
5.2 A Simulated Environment . . . 110
5.2.1 Simulator for Maryland Imitation Learning Environment (SMILE) 111
5.3 Neural Architecture . . . 118
5.3.1 Open-loop Subsystem . . . 120
5.3.2 Closed-loop Subsystem . . . 122
5.3.3 Motor Output Filter . . . 124
5.4 Training Methods . . . 126
5.4.1 Stage 1: Individual map training . . . 127
5.4.2 Stage 2: Joint command output . . . 127
5.4.3 Stage 3: Inter-modality associations . . . 129
5.5 Experimental Methods . . . 131
5.6 Results . . . 135
5.6.1 Map and Limit Cycle Formation . . . 135
5.6.2 Arm Performance and Trajectory . . . 137
5.6.3 Standalone Open-Loop and Closed-Loop Subsystems . . . 140
5.6.4 Performance in the Presence of Noise . . . 147
5.6.5 Sensitivity to Timing Parameters . . . 154
5.7 Physical Robot Implementation Results . . . 155
5.8 Summary and Discussion . . . 159

6 Discussion and Conclusion 163
6.1 Summary 163
6.2 Limitations and Future Work 168
6.3 Summary of Contributions 170

Bibliography 173

List of Tables

2.1 Two traditional types of SOMs 15

3.1 Parameters for nonlinearly decreasing functions used during training . 38
3.2 Summary of limit cycle SOM parameters 39
3.3 Comparisons between pre- and post-training attractors 49
3.4 Distance correlation and average distance of limit cycles 63

5.1 Summary of performance results for robotic arm control 138

List of Figures

2.1 An example SOM neural network 11
2.2 Weight values of a SOM trained with an iris flower data set 12
2.3 Labeled map of a SOM trained with an animal data set 13
2.4 Simple recurrent SOM 20

3.1 An example limit cycle SOM network 34
3.2 Sample phoneme sequence and static image data 41
3.3 Effects of self-link strength and continuation time on attractor formation 44
3.4 Distribution of the limit cycle lengths 45
3.5 Attractor stability 47
3.6 Attractor uniqueness 48
3.7 Typical limit cycles representing phoneme sequences 50
3.8 Phoneme map formation 53
3.9 Phoneme map U-matrices 55
3.10 Image map formation and U-matrices 56
3.11 A typical limit cycle representing a 2D location 60
3.12 Comparisons of limit cycles representing different 2D locations 61
3.13 Distances between limit cycles representing different 2D locations . . 62
3.14 2D spatial map formation 64

4.1 Architecture for associating static representations with limit cycles . . 71
4.2 Experimental procedure for associating static representations with limit cycles 72
4.3 Results for associating static representations with limit cycles 76
4.4 Architecture for associating two sets of limit cycles 78
4.5 Experimental procedure for associating two sets of limit cycles 79
4.6 Results for associating two sets of limit cycles 81
4.7 Schematic diagram of the arm model 85
4.8 Architecture for open-loop arm control 87
4.9 A typical limit cycle of length 6 representing a 3D spatial location . . 90
4.10 Summary of gate timing 97
4.11 3D spatial and joint angle map formation 98
4.12 Distribution of limit cycle lengths 99
4.13 Convergence of limit cycle alignments in the course of training. 100

4.14 Distribution of spatial errors before and after training 101
4.15 Effects of timing parameter values 103

5.1 Architecture for combined open-loop and closed-loop arm control . . 109
5.2 An example view of the SMILE software simulation 113
5.3 Sample objects in SMILE 114
5.4 The demonstration interface in SMILE 115
5.5 Screenshots of a "UM" demonstration in SMILE 116
5.6 Screenshots of a robot performing the demonstrated "UM" task in SMILE 116
5.7 Snapshots of a physical robot performing the "UM" task 117
5.8 The goal states of the "UM" task 117
5.9 Weighting functions for the open-loop and closed-loop outputs 125
5.10 Schematic illustration of the three training stages 128
5.11 Schematic illustration of the execution 132
5.12 Examples of map formation 135
5.13 Distribution of limit cycle lengths 136
5.14 Distribution of spatial errors 139
5.15 Arm trajectory for a typical reaching movement 141
5.16 Comparisons among the open-loop-only, closed-loop-only, and full architectures 142
5.17 Arm trajectory of the open-loop-only architecture 143
5.18 Arm trajectory of the closed-loop-only architecture 144
5.19 Distribution of per-iteration angular error for the closed-loop subsystem 145
5.20 Effects of internal interference 148
5.21 Arm trajectories for impaired spatial and spatial difference maps . . . 151
5.22 Arm trajectories for arm joint perturbation during a reaching movement 153
5.23 Effects of timing parameter values 156
5.24 Snapshots of arm trajectories of a physical robotic arm 157
5.25 Arm trajectory of a physical robotic arm 158

— Chapter 1 —

Introduction

Substantial research in neurocomputation has been evolving towards building large-scale neural architectures. A *neural architecture* is a system of neural networks composed of multiple interacting neural components, modules, or "regions", that often loosely correspond to cortical or other brain regions. Recent development of large-scale neural architectures ranges from computationally oriented deep neural networks (Bengio and Lee, 2015; LeCun et al., 2015) to neuro-anatomically grounded simulations of all or major portions of human/mammalian brain structure and function, or at least major subsystems of the brain that span multiple cortical regions. The latter varies from extremely large networks of biologically-realistic spiking neurons to those that are more abstract, based on a higher level of components such as cortical columns, or are focused on simultaneously supporting cognitive functions (Eliasmith et al., 2012; Garis et al., 2010; Weems and Reggia, 2006; Winder and Reggia, 2012). Work in this area has been accelerating in part due to major recent research funding initiatives (the Human Brain Project in Europe, the BRAIN Initiative in the US, etc. (Abbott, 2013)). Often, a region in a neural architecture has been represented by a layer of mutually unconnected nodes, something that is an over-simplification of its counterpart in the brain. An alternative is to instead represent a region using a

self-organizing map (SOM).

A SOM is a two-layer neural network inspired by cortical maps found in biological neural systems (Kohonen, 2013; Malsburg, 1973). It learns, using unsupervised methods, to map high-dimensional inputs onto its output nodes that form a low-dimensional (usually 2D) lattice. This mapping is non-linear and topology-preserving, meaning that nearby output nodes in a trained map are typically sensitive to similar input patterns, although sudden jumps may also occur. Due to this property SOMs can effectively cluster and visualize complex high-dimensional data that may otherwise be difficult to understand, and thus they have been frequently used across several disciplines and applications, especially for data visualization (Hsu and Lin, 2012; Kohonen, 2001, 2013; Manukyan et al., 2012). SOMs have been applied in a wide range of fields, such as robotic control, weather monitoring, genome analysis, and economic modeling, to name just a few examples. In addition to their computational uses for unsupervised clustering and visualization, SOMs also account for many aspects of observed phenomena in biological cortical regions, including topographical self-organization of somatosensory, visual, and auditory cortices (Miikkulainen et al., 2005; Sutton et al., 1994), the alignment of multiple feature maps (Chen and Reggia, 1996), the formation of mirror-symmetric maps (Schulz and Reggia, 2004; Sylvester and Reggia, 2009), and related cognitive phenomena (Bednar and Miikkulainen, 2000).

However, to date SOMs have only played a limited role in large-scale neural architectures, something that is surprising given the tremendous interest in SOMs in general. In spite of their successes as a standalone computational tool, most conventional SOM models are substantially limited from the viewpoint of computer

science, cognitive science, and neuroscience. They face several significant barriers to more widespread use in large-scale neural architectures.

First, many SOMs follow Kohonen (2001) in activating a single "winner" node whose incoming weights best match an input pattern. Such single-winner activation encodes information in an inefficient way similarly to a "one-hot" coding scheme, limiting an N-node SOM to representing/differentiating only N distinct external input patterns.

A second barrier to adopting SOMs in neural architectures is that many SOMs do not address temporal sequence processing. SOM models often take a single fixed pattern as input, and when the next input is received, effectively reset the map region before processing the following input, with the order of sequential inputs thus being largely irrelevant. In contrast, biological systems receive a continuing stream of input patterns, and even if these patterns are discretized, often their order is critically significant (e.g., phonemes forming a word). To date, only a very limited number of SOMs have been developed for processing temporal sequences.

A third barrier to adopting SOMs in neural architectures, and the one most central to the work reported below, is that most past SOMs have used a *static representation* of information. This means that each input pattern or sequence of patterns is typically represented by a *single fixed* activation pattern over the map layer, without considering how that activation pattern evolves with time. On the contrary, brain activity is continuously changing and highly oscillatory, something that is difficult to reconcile with static representation. In conventional SOMs, activation patterns are driven directly by fixed input stimuli, and thus remain static until

external control mechanisms alter or reset the map layer. This remains true even in past SOMs for sequence processing: Even though the map regions have changing activation patterns over time, the representation that encodes a whole input sequence is still a single spatial activity pattern that is chosen from the series of changing activation patterns (e.g., the activation pattern that occurs in response to the last element of an input sequence (Schulz and Reggia, 2004)). A sequence's representation is therefore still static. Static representations as used in past SOMs are not stable, since an isolated activity state generally does not have the mechanisms to recover from activity perturbations. Importantly, a static representation is usually transient — it usually occurs only once for a very short time, typically in-between updates of a SOM's activation (unless the SOM's activity is artificially frozen). This makes it difficult to control the timing of accessing each SOM's transient representation in a neural architecture, since such control not only needs to be highly temporally accurate, but also requires the knowledge about *when* to access each SOM.

Finally, conventional SOMs are ill-suited for building biologically grounded simulations of brain structure and function. One reason is that, in contrast to widespread and distributed neural activity in biological cortical regions, many conventional SOMs use a biologically implausible *global* selection process of a single winning node. Another critical reason is that biological neural systems generally do not exist in static activation states, and they are unlikely to operate solely based on fixed spatial patterns. On the contrary, biological systems are clearly characterized by prominent ongoing rhythmic oscillatory activity (Buzsaki, 2006; Niedermeyer and Silva, 2005), and there is substantial evidence that cognitive functions such as memory are strongly related to

this rhythmic oscillation of neural activities (Fell and Axmacher, 2011). The oscillatory nature of this ongoing activity has rarely been examined in past work on SOMs.

To address the above issues concerning conventional SOMs, a non-static distributed representation could be developed. However, it is currently unknown how such a representation based on changing activity can be learned in a SOM to encode external stimuli, as well as what its properties would be. It is also important to understand its relative advantages over conventional static representations, and whether map formation like that in conventional SOMs would still even occur when this non-static representation is in use. Further, given non-static activity in individual SOMs, it is currently unknown whether such activity can be harnessed to drive an entire neural architecture containing multiple SOMs. It seems even more uncertain whether non-static representations in a multi-SOM architecture can be used at a more abstract level for general computations, especially those whose outputs are required to be static (e.g., holding a robot's arm in a fixed position), and whether such an architecture generalizes to new situations.

1.1 Goals and Specific Aims

Motivated by the above issues, and inspired by the oscillatory nature of brain activity, the overall goal of this research is to explore building neural architectures containing multiple SOMs using new *dynamic representations* that are limit cycle attractors. More specifically, static and temporal sequence inputs alike are to be encoded using a temporal sequence of multi-winner activity patterns in a SOM that collectively form

a limit cycle attractor, in contrast to using a fixed single-winner activity pattern. The idea is to study how sustained neural responses to transient stimuli that form multi-focal dynamic activity can possibly be used in computation. To my knowledge, this is the first attempt to study multi-SOM architectures based on limit cycle activity. My hypothesis is that such neural architectures based on limit cycle attractors can serve as useful computational tools that are robust, generalize to new situations, simplify control, form topographic maps, and will also generate cortex-like oscillatory activity.

In this context, there are three specific aims of this research, as follows.

1. Explore the use of limit cycle attractor dynamics to encode inputs in individual SOMs. The goal is to create a class of SOMs, called *limit cycle SOMs*, that adopts limit cycle attractors consisting of multi-winner activity patterns to represent both static and temporal sequence inputs through self-organization (i.e., in an unsupervised fashion). The viability of using limit cycle attractors as an internal representation needs to be carefully evaluated, including their stability, uniqueness, and correlations to input space patterns. Appropriate comparisons are also needed to better understand and characterize limit cycle representations in relation to other alternatives, including other types of self-organized attractors as well as the baseline static representation method.

2. Determine the viability of learning to associate limit cycle representations between different SOMs, so that one limit cycle in a SOM can be used to retrieve a corresponding limit cycle in another SOM. The intent here is to establish an

effective mechanism for communication between two SOMs, something that is a critical step towards building a neural architecture with multiple interconnected limit cycle SOMs. Learning such associations for limit cycle activity is a much more challenging task than for maps with static single-winner activity. This is because each limit cycle contains multiple activity patterns and each of these activity patterns contains multiple winning nodes that have to be associated with another limit cycle. To make it worse, each winning node is likely to participate in many different limit cycles. It is currently unknown whether such associations can be learned at all. Even if learning such associations is possible, it is still unclear how general and how robust it would be.

3. Build a multi-region neural architecture using limit cycle SOMs for a practical application, stable arm control, to establish the effectiveness of this approach. The goal is for the architecture to learn to move a robotic arm to reach a target 3D location and then hold it fixed there. The key question being asked here is whether and how this architecture can be created to accomplish holding a *fixed* non-oscillatory position, much as a person can do with an arm reaching movement, in spite of its internal activity being continually changing, and to simultaneously generalize to new situations. The goal is to eventually be able to control a physical robotic arm using this architecture. If successful, this architecture would be a first step towards using dynamic/oscillatory activity for general computation. Further, it is important to examine the contribution of each neural component in a limit cycle architecture to overall performance, by

disabling or perturbing different parts and by varying internal control timing. Such results will also help reveal how robust a limit cycle SOM-based architecture can be in general.

1.2 Overview

The rest of this dissertation is organized as follows. Chapter 2 presents background information about basic SOMs, SOMs for temporal sequence processing based on changing or oscillatory activity, and past neural architectures containing multiple SOMs. Chapter 3 examines the fundamental issues involved in creating limit cycle SOMs to encode both static and temporal sequence inputs, as well as the properties of limit cycle representations relative to other alternatives. Chapter 4 explores methods for learning associations between multiple limit cycle SOMs. It does so by introducing different network configurations and learning rules in two different tasks and evaluating their effectiveness. Chapter 5 presents a full neural architecture containing multiple limit cycle SOMs to control a robotic arm for stable and precise reaching movements. Experiments with a simulated and a physical robotic arm are reported. Chapter 6 concludes this dissertation with a summary and discussion of the limitations and possible future directions for addressing them.

— Chapter 2 —

Background

There is an enormous literature concerning SOMs. The number of known related publications reached 7717 in 2005, and this number does not include those appearing during the last decade (Kaski et al., 1998; Oja et al., 2003; Pöllä et al., 2009). However, relatively little of this literature has focused on changing activity and multi-SOM architectures. Further, to my knowledge, none of this past work has discussed the possibility of using limit cycle attractors as representations in a way that has any similarity to this work. This chapter first briefly introduces basic SOMs, followed by a review of extended SOMs that allow changing activity patterns for temporal sequence processing. Lastly, the current state of integrating multiple SOMs to build neural architectures is summarized.

2.1 Basic SOMs

While details may vary, the basic SOM can generally be described as a group of discrete artificial neurons, sometimes referred to as simply "nodes", organized in a low dimensional (usually 2-D) lattice, which, through self-organization, learns a typically nonlinear, topographic projection of the distribution of input feature vectors in high

dimensional space. Figure 2.1 shows the structure of an example 2-D SOM network. While many studies assume each map node to be connected to all input features, this is not a requirement for obtaining self-organization behaviors. Node activity of a SOM is determined by a competitive activation process, where each map node competes to become active based on to how similar its weights are to inputs. The most activated node, i.e., the one with the most similar weights, is typically referred to as a "winner". Following random weight initialization, competitive Hebbian-type weight adaptation occurs at each winner and its neighbor nodes, where the rate of change typically decreases as the location of a node is further away from the winner. The training process is unsupervised in that the input data are simply being presented to the SOM repeatedly, and the result is a mostly smooth topographic projection, or a "map", of the input space patterns. The map is mostly smooth in the sense that neighboring nodes have similar weight vectors. This self-organization behavior is based on both the statistical properties of input distributions and interactions among the nodes in the SOM. Although the resulting map is overall smooth, it may also be subdivided into regions that are internally smooth and are separated by relatively less smooth boundaries, with each region representing a cluster in the input space.

To illustrate map formation, Figure 2.2 shows a post-training map example for the popular iris flower dataset (Fisher, 1936)*. The map is smooth since the weights tend to be similar between neighboring nodes, and when visiting nodes across the map in any direction, one can observe that the weight changes are gradual and forming a

*Examples are produced using the SOM Toolbox available at `http://www.cis.hut.fi/somtoolbox/`

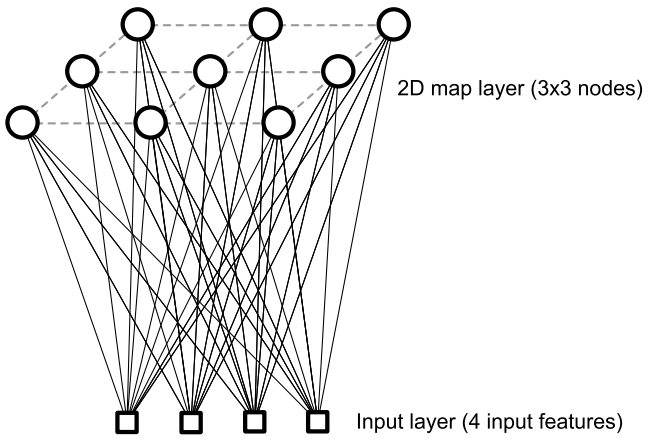

Figure 2.1: An example 2-D SOM with 3×3 nodes receiving input feature vectors containing 4 components. Although each node in the example SOM is connected to all input features, this all-to-all connectivity is not required and some SOM networks use more restricted connectivity. In general the map and input layers are much larger than what is shown here for illustrative purposes.

gradient. For example, the left-most map in the top row of Figure 2.2 indicates that weights corresponding to septal length gradually increases from the top-left corner to near the bottom-right corner. The three maps at the bottom row of Figure 2.2 appear to be less smooth (i.e., greater weight changes between neighboring nodes around cluster boundaries). This is because these three features are binary-valued (i.e., true/false), indicating which type of iris (i.e., Setosa, Versicolour, or Virginica) each input pattern represents. Figure 2.3 shows another example of map formation for animals. The training set is taken from Kohonen (2001): Each animal's appearances and behaviors are described by 13 binary input features such as "is big", "has feathers", and "likes to swim". After training, each animal is labeled at the map node whose weight vector is the most similar to the animal's feature vector. While animal grouping is not explicitly embedded in the input features, the learned map clearly groups similar animals in nearby regions. For example, one interpretation is that large mammals are

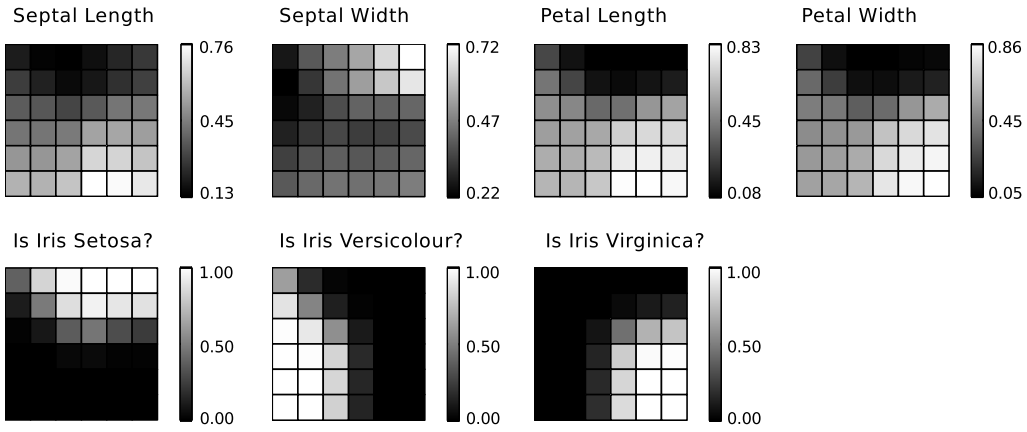

Figure 2.2: An example 6×6 SOM trained with the popular iris flower data set (Fisher, 1936). Each subplot depicts the weight value for each of the 7 input feature, including septal length, septal width, petal length, petal width, and the three classes: Is Iris Setosa, Is Iris Versicolour, and Is Iris Virginica. The last three input features are binary indicators (either 0 or 1), and exactly one of them can be 1. Greater connection weights are represented as lighter shades in the grids. Weight value range is shown next to each map. As an example, the upper-right node represents Iris Setosa that has medium septal length, high septal width, and low petal width and length.

grouped at the top left, felines at the top center, canines at the top right, and birds at the bottom.

According to how competitive activation is realized in the map layer, existing SOMs can be classified into two broad types (Table 2.1). The first type, named *one-step SOM* here, does not involve lateral inhibitory connections for activity competition. Instead, it makes a one-step decision locating the single node whose weights best match input features (Kohonen, 1982, 2013). This node is declared as the winner, and a peak of neural activity centered at the winner is then artificially imposed. The connectivity between input features and map nodes is assumed to be full, and generally only one winner is allowed at a time. Although this approach substantially sacrifices biological fidelity, the computational efficiency is greatly improved because this winner-takes-all

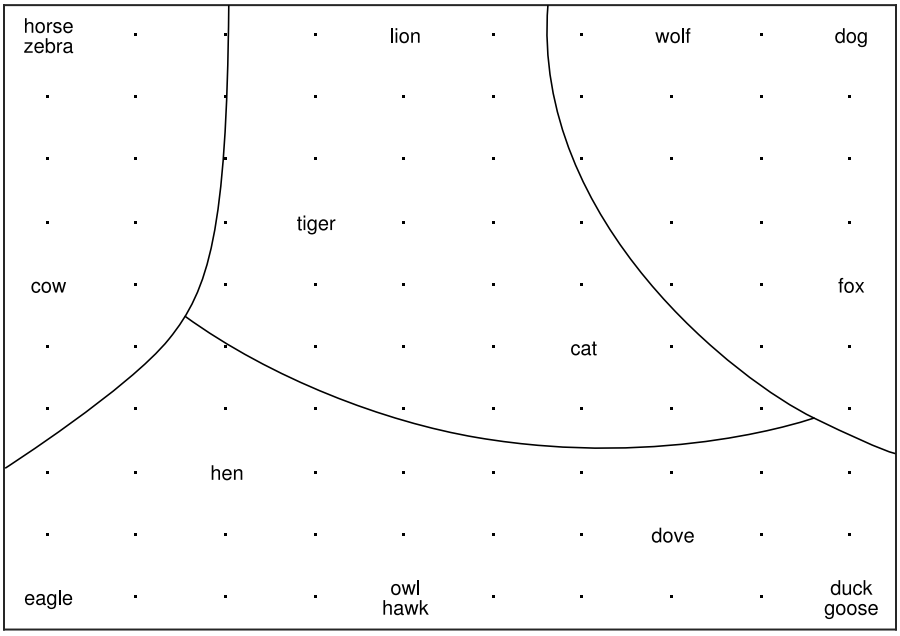

Figure 2.3: An example 10×10 labeled map. The training set consists of 16 animals, each described by 13 binary features, such as "is big", "has feathers", and "likes to swim" (Kohonen, 2001). Each animal is labeled at the map layer node whose weights are the most similar to its features. Curved lines are manually added to illustrate self-organized clusters of large mammals (top left), felines (top center), canines (top right), and birds (bottom). Note that these clusters are not embedded in the input features, but they are discovered through self-organization.

process only needs a one-step computation, rather than iterative computation for solving differential equations over all map nodes. Due to computational efficiency, one-step SOMs' salient ability to cluster and reduce dimensionality can be investigated in more detail, making them a popular computational module and visualization tool in various application disciplines (Kaski et al., 1998; Oja et al., 2003; Pöllä et al., 2009). For example, in addition to their obvious relevance to computer science-related areas, SOMs have been widely adopted in general engineering (Kohonen, 2013; Kohonen et al., 1996), bioinformatics (Bouvier et al., 2015; Delgado et al., 2015; Fankhauser and Mäser, 2005), economics and finance (Chen et al., 2013; Deboeck and Kohonen, 2013; Louis et al., 2013; Shanmuganathan et al., 2006), geoscience (Bação et al., 2005; Gorricha and Lobo, 2012; Kalteh et al., 2008), meteorology (Chang et al., 2010; Liu and Weisberg, 2011), and so on. Additionally, variants of SOMs have recently been used for processing complex structures in input spaces for computational applications such as handwriting recognition (Mohebi and Bagirov, 2014), texture/image synthesis (Hua, 2016), and tensorial data analysis (Iwasaki and Furukawa, 2016). However, the vast majority of one-step SOMs is significantly limited in selecting only a single winner node. This global selection of a winning node is not only biologically implausible, but it also limits the expressiveness of a SOM. This means that a SOM can only generate N different activity patterns, and it can only differentiate N distinct input stimuli, where N is the number of map nodes. From an information encoding standpoint, this is unacceptably inefficient.

The other type of SOM, named *iterative SOM* here (Table 2.1), stems from the seminal work of Malsburg (1973), which attempts to simulate the self-organization of

Table 2.1: Two traditional types of SOMs.

Type	One-Step SOMs	Iterative SOMs
Seminal work	Kohonen (1982)	Malsburg (1973)
Primary application	Computations	Biological modeling
Lateral connections	Implicit	Explicit
Activation dynamics	One-step selection of the best matching node	Iterative solution of nonlinear differential equations involving lateral connections
Typical activity pattern	Single global winner[1]	Multiple distributed winners[2]
Computational cost	Low	High

[1]Except some work that allows multiple winners, e.g., Schulz and Reggia (2004)
[2]Except some work that identifies a single winner, e.g., Rumbell et al. (2014)

orientation-sensitive cortical columns in the primary visual cortex of cats and monkeys. This type of SOM model contains explicit lateral connections between map-layer nodes, where each node excites nearby nodes while inhibiting farther away ones, so as to approximate the biologically observed Mexican-hat pattern of neural activities. The dynamics are described by nonlinear differential equations that, through iterative calculations, implement competitive activation dynamics where multiple peaks of neural activities (i.e., the winners) are separated from each other. Similar approaches also adopting nonlinear differential equations (Bednar and Miikkulainen, 2000; Chen and Reggia, 1996; Grajski and Merzenich, 1990; Huang and Hagiwara, 1997; Ménard and Frezza-Buet, 2005; Rumbell et al., 2014; Sutton et al., 1994) are primarily concerned with modeling and explaining biological phenomena, such as modeling the somatosensory cortex (Grajski and Merzenich, 1990; Sutton et al., 1994), explaining the alignment of multiple feature maps in the cerebral cortex (Chen and Reggia, 1996), and relating to psychological phenomena (Bednar and Miikkulainen, 2000). This

iterative type of SOMs generally has divergent but localized connectivity between input features and SOM nodes, and allows multiple simultaneous winners, which are also considered biologically grounded. However, the iterative calculations of nonlinear differential equations are extremely computationally expensive.

To overcome the shortcomings of both types of SOMs described above, a third type of SOM is the one-step SOM described in Schulz and Reggia (2005), which allows multi-winner activation. Instead of global competition, this third type of SOM uses local competition that allows multiple winners to be declared simultaneously if their similarity to input stimuli is the greatest in their local neighborhood. As a result, this one-step multi-winner SOM is not only computationally efficient, but it also encodes input stimuli in a more efficient way (multiple simultaneous winners) than classic Kohonen-style single-winner SOMs. Additionally, by avoiding global competition, one-step multi-winner SOMs facilitate implementations using parallel computations. This essentially enables constructing large-scale SOMs. Also, allowing multiple simultaneously-activated nodes is more biologically plausible, which is supported by biological observations and supported by successful SOM models of the mirror-symmetric maps formed in the neocortex (Schulz and Reggia, 2005; Sylvester and Reggia, 2009).

Although the basic SOM has had significant success in biological modeling and computational applications, it is significantly limited by disregarding temporal aspects of its activity and the inputs it receives. That is, its activity or input patterns are generally assumed to be independent of one another, and the basic SOM has no explicit memory about the inputs received in the past or previously occurring activity

patterns. In other words, it treats each input and activity pattern in isolation, rather than considering them to be sequential (a time series). This makes the basic SOM ill-suited for processing temporal data. It is also directly contrary to biological neural systems, which, by utilizing some form of short-term memory, routinely process each input stimulus in the context of a longer time-varying sequence containing other stimuli. An example can be found in the human ability to understand spoken language by processing series of time-varying auditory stimuli. Moreover, there is substantial evidence that brain activity must be understood as a spatiotemporal process, instead of regarding each spatial activity pattern in isolation (Fell and Axmacher, 2011). Therefore, the basic SOM needs to be extended to account for changing activity in order to process temporal sequence data, as well as to achieve more biological-like intelligence.

2.2 SOMs for Temporal Sequence Processing

The basic SOM has been extended in different ways to handle temporal sequences (Guimarães et al., 2003; Hammer et al., 2005; Salhi et al., 2009). A temporal sequence input can be defined as lists of feature vectors ordered in time. In order to process temporal relations of an input sequence, some forms of short-term memory must be in place to retain events from the past. In many past studies, such short-term memory is realized in the form of changing activity, i.e., different activity patterns at different times are not considered in isolation as in basic SOMs, but in relation to one another. Since limit cycle SOMs introduced in this research aim to support temporal sequences, and

since activity dynamics is of central importance to limit cycle SOMs, this section presents past strategies for SOMs to handle temporal sequences.

Earlier studies tended to process temporal sequences using unmodified basic SOMs. A naive method is to concatenate a fixed number of successive input patterns to serve as input to a SOM (Kangas, 1990; Wan and Fraser, 1999). In this case, short-term memory is directly embedded in the concatenated input patterns. This method is significantly limited in being unable to capture the statistical dependency between successive patterns because SOMs are generally insensitive to the ordering of input features. Additionally, it is usually impossible to decide a fixed optimal number of patterns to be concatenated. Another method is to explicitly record winner locations in the map layer, as each input pattern is presented in succession to a basic SOM (Kohonen, 1988; Leinonen et al., 1993; Saxon and Mukerjee, 1990). The ordered list of winner locations can thus form a path in the map layer representing the input sequence. For example in Saxon and Mukerjee (1990), a trajectory of a robotic arm (i.e., a sequence of arm joints) can be mapped to a path in the map layer, and vice versa. A similar method tracks temporal changes of input distributions by concatenating maps that process inputs at different times, and thereby forming winner paths for input sequences (Sarlin, 2013). However, these methods require a separate non-neural mechanism to serve as short-term memory to store the paths in the map layer. While this form of memory is easy for humans to interpret, it is difficult for other neural components to process.

Another approach is to represent short-term memory using leaky integrators (Carpinteiro, 1999; Chappell and Taylor, 1993; Koskela et al., 1998). A leaky integrator is

a loose model of a neuron that acts like a capacitor. Once charged, it gradually discharges over time until being charged again. Suppose $a(t)$ and $h(t)$ are the activity and the input of a leaky integrator at time t, respectively. The activity dynamics follow the form: $a(t) = h(t) + \eta a(t-1)$, where $0 < \eta < 1$ is the decay ratio. Therefore, leaky integrators carry over the activity in the past through time, serving as short-term memory for temporal context. They can be used in the input layer to combine current and past input vectors (Carpinteiro, 1999), or used to combine current and past net inputs before competitive activation (Varstal et al., 1997). It is also possible to replace nodes in the basic SOM with them (Chappell and Taylor, 1993; Koskela et al., 1998), such that winners triggered by input vectors in the past may remain partially activated at a current time and thus take part in encoding sequences. This approach eliminates the need for a separate memory mechanism. However, although each of the leaky integrators can serve as a memory unit, their acting independently (i.e., the activity of one leaky integrator does not depend on that of the others) makes the overall memory unstable, since a small amount of noise in one unit cannot be corrected by others through mutual interactions.

A well-recognized method for temporal sequence processing is to add a context layer and recurrent connections to a basic SOM as shown in Figure 2.4a. Similar to simple recurrent error backpropagation networks that are widely used for sequence processing (Elman, 1990), the neural activities of the SOM (called a simple recurrent SOM here) at the previous time step are copied and temporarily stored in the context layer, and then transferred via the recurrent connections back to the SOM as a part of its input (the other part being the feature vector via the afferent connections) at

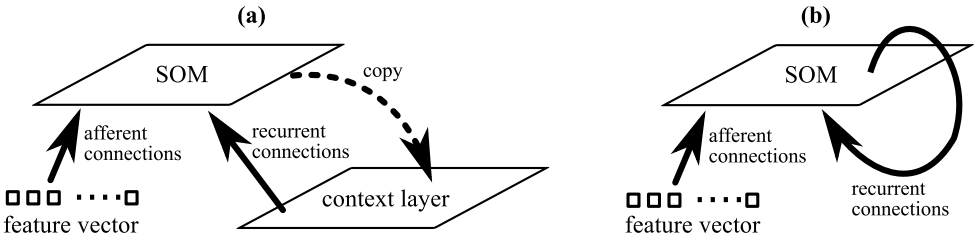

Figure 2.4: Simple recurrent SOM. (a) Similarly to simple recurrent networks using error backpropagation, a simple recurrent SOM is constructed by adding a context layer and recurrent connections. The dashed arrow denotes direct copying rather than adaptable connections. (b) An equivalent model without the context layer, assuming synchronous updating of neural activities.

the current time step (Schulz and Reggia, 2004; Voegtlin, 2002). In other words, the output activities of the SOM are taken as a part of its input with a delay of one time step, and thus the activity of each node depends on that of many other nodes at the previous time step, forming a more reliable short-term memory than leaky integrators. Notice that since the context layer is only used as a temporary storage to perform one-time-step delay, the simple recurrent SOM can be simplified by removing the context layer and assuming all neural activities are updated synchronously (Figure 2.4b).

However, since most past work uses single-winner SOMs, the activity patterns copied to the context layer are expected to contain only one activated node each. This one-hot encoding scheme is limited in the amount of information if can convey (e.g., the number of distinct patterns it can express is limited), and therefore storing single-winner activity in the context layer is inefficient. The work-arounds summarized below store other information in the context layer in place of single-winner activity. In Horio and Yamakawa (2001), leaky integrated activity is stored in the context layer. This effectively keeps track of past winners with a time discount. In Voegtlin

(2002), net input without a winner-takes-all process is stored in the context layer. Other approaches have their context layers store information that is artificially derived from single-winner activity. Examples include Strickert and Hammer (2005), where weight values of the winner node is used, Chow and Rahman (2009); Hagenbuchner et al. (2003); McQueen et al. (2004), where the location of the winner node is used, and Kaipainen and Ilmonen (2003), where the phases of map nodes that act like neural oscillators are used. However, such derivation from single-winner activity requires additional explicit computation, and may sometimes need to extract parameter values (e.g., weights) from map nodes, which lacks biological grounding. Another approach uses leaky integrators in the context layer to combine one-hot neural activities at different time steps (Wakuya and Terada, 2009), where the context layer is no longer just a time-delay temporary storage (and therefore unable to be simplified to Figure 2.4b), but an explicit memory that requires additional neurons to implement. This SOM suffers from the same instability problem as general leaky integrators.

Although some past SOMs can demonstrate oscillatory activity, their oscillations are usually a built-in behavior, as opposed to being a result of the map layer's self-organization as occurs in my research. An example is oscillator-based SOMs, which use oscillators as independent map nodes with explicit phase and frequency parameters. These models involve synchronizing the potential firing frequencies of the map nodes with external sequences that are assumed to be intrinsically periodic (Kaipainen and Ilmonen, 2003). The result is that, after training, the weight vector of the (single) winner node at each time step roughly fits the periodic training sequences. In other words, the oscillator-based SOM is mainly concerned with reproducing

the frequencies of periodic input sequences rather than representing general inputs. Another example is SOMs based on spiking neurons that fire (i.e., neuronal discharge) periodically (Rumbell et al., 2014). Note that the temporal scale here is at the level of individual neuron firing (i.e., "spikes") instead of mean firing rates that most other work reviewed here discusses. Oscillations in this case are typically a built-in behavior of individual leaky integrate-and-fire neurons, rather than a result of the map layer's activation dynamics. In both examples, the oscillatory activation in the map layer is a direct result of the underlying stimuli being periodic. It is unclear if they would exhibit any oscillatory activation at all should the input stimuli become aperiodic or unavailable, and unlike with my work in the following, the issue of whether oscillations would continue or what they would represent if they did continue is not considered at all.

Other approaches modify the fundamental working principles of the basic SOM while keeping its competitive nature, such as modifying the winner-takes-all process or how node activities are assigned. In Kohonen (1991), the winner-takes-all process is divided into multiple stages. To decide the winner for a given input pattern, the preceding and succeeding patterns of the input is first concatenated and used to select multiple potential winners, from which the final winner is then decided by the input pattern itself. Some studies allow the neural activities in a SOM to directly spread to neighboring nodes (Euliano and Principe, 1999; Wiemer, 2003), creating traveling waves of activities when triggered by the activation of winners. The diffusion of activities then influences the following winner-takes-all processes by increasing or decreasing each node's potentiality to be selected as the next winner. Such direct

interaction between neighboring nodes provides a short-term memory for temporal context. Another method forcibly prohibits the winners in the past from becoming a winner again (James and Miikkulainen, 1995), and by exponentially decaying the activities of the past winners, temporal sequences can be expressed by spatial patterns. However, since these approaches have to modify the well-understood and time-tested working principles of the basic SOM, the modifications and the additional assumptions they make need further investigation on both theoretical and biological grounds. Unfortunately, the modifications usually introduce complexities into SOMs, which makes them harder to analyze.

A common limitation of all of the above approaches is that the changing activity in SOMs is directly driven by input sequences. As a result, an activity sequence in a past SOM is usually irregular. This means that a specific activity pattern occurs only once in an activity sequence, and the duration in which it occurs is typically very short (e.g., one time step).[†] Great challenges are thus imposed for a downstream neural component to read a specific pattern from the sequence, since the timing control needs to be very accurate, which is likely to result in high control overhead. A second result of having activity directly driven by an input sequence is that, once an input sequence is removed, the activity typically becomes silent or its meaning becomes unclear. Past work tends to represent a temporal sequence using the SOM's activity patterns at one time step or during a few time steps (i.e., static representation), when part of the sequence is being provided as input. However, the brain is clearly characterized by

[†]It is unclear if Kaipainen and Ilmonen (2003) is an exception. Assuming it is (i.e., the same activation patterns periodically occur), the fact that it relies on the input sequences being periodic makes it ill-suited for processing general temporal sequences (see discussions above).

maintaining sustained neural responses to transient stimuli. The ongoing activities in biological systems after input sequences end implies that sequence processing is likely to continue even after a complete sequence has been received. This research addresses these two issues by driving the trajectory of a SOM's activity states into a limit cycle attractor, such that post-sequence activity forms periodically repeating patterns. This not only lowers the timing requirements for the control mechanism because the same patterns appear periodically, but also utilizes ongoing activities for computation after input stimuli end.

2.3 Multi-SOM Neural Architectures

Only relatively limited efforts have been made in the past to combine multiple SOMs into a single neural architecture that accomplishes more complicated computations. The most straightforward method is chaining a few SOMs in a linear structure, where the first SOM (lower level SOM) directly receives raw input vectors and passes its output to the next SOM, and the next SOM in turn passes its output to the third SOM (higher level SOM), and so on. Generally, the purpose of such an architecture is to provide views of different abstraction levels of the input data, where a higher level SOM located farther away from the raw input provides a more abstract perspective on the data.

For example, Kayacik et al. (2003) and Lichodzijewski et al. (2002) detect computer intrusion attempts with a low level SOM taking in various fields in TCP packets, while a high level SOM decides whether or not a certain event is an attack based

on the information provided by the low level SOM. Another example is Kohonen (1997), which clusters a large body of Internet documents using a low level SOM to compute word histograms, and a high level SOM to perform document-level clustering based on the histograms. A "deep" structure can be found in Liu et al. (2015), where alternating SOM layers and sampling layers are stacked and trained to recognize progressively more abstract input patterns. Similar deep structures can also serve to progressively filter background/foreground pixels in image processing tasks (Zhao et al., 2015). The idea of abstraction level can also be extended to the time domain, such that higher level SOMs account for data in longer time intervals (Carpinteiro, 1999; Guimarães, 2000; Monner and Reggia, 2009). In Monner and Reggia (2009), which processes temporal sequences of phonemes, a low level SOM combines multiple phonemes into English words, and a high level SOM combines multiple words into sentences. However, a common limitation of these methods is that they typically deal with only one stimulus modality. That is, unlike biological beings that simultaneously sense multiple aspects of external objects or events, such as sound, temperature, taste, smell, etc, these methods, despite some of them being able to process temporal sequences, only take one aspect/modality at a time and ignore the interdependencies that can occur among multiple modalities.

In contrast, there are a few studies focusing on integrating heterogeneous input from multiple modalities using SOMs. One approach is to have a SOM that serves as a multimodal integrator, which takes its input simultaneously from the output of multiple SOMs that process unimodal data, or directly from multiple sources of input with different modalities (Johnsson and Balkenius, 2008; Mohan et al., 2013;

Morse et al., 2010). A prominent example is the integration of multiple modalities of a robot, such as visual, language, and motor modalities (Lallee and Dominey, 2013; Morse et al., 2010). Another example is integrating individual English phonemes and written letters (Gustafsson and Paplinski, 2006), where a one-to-one mapping between phonemes and letters is assumed. In Johnsson et al. (2011), a special SOM takes input both directly from a source of unimodal data and from the activities of other SOMs. Using customized activation and learning rules, it is able to associate its unimodal input with the activities of other SOMs. A similar approach is to have multiple smaller multimodal SOMs, where their input is vectors built by concatenating data from multiple modalities (Wan and Fraser, 1999). These multimodal SOMs compete with each other for learning, and therefore each of them covers a cluster in the input space. Unlike with most other methods, the SOMs do not interact with each other by directly transferring neural activities, but rather by mutual inhibition. Another approach, instead of having an explicit multimodal SOM, establishes a heterogeneous associative memory between two or three unimodal SOMs using Hebbian learning (Li et al., 2007; Yoshikawa et al., 2003). As with Gustafsson and Paplinski (2006), the primary limitation is that a one-to-one mapping between winners in the SOMs is assumed. A more biologically-plausible model in Khouzam and Frezza-Buet (2013) uses a distributed competitive process in SOMs as well as topographic connectivity between SOMs, to model the cognitive ability to disambiguate input patterns based on temporal context. The connections between SOMs and thus the propagation of activity in this case can be bidirectional. Similarly to the work summarized here, this research constructs architectures that process multiple modalities. However, since

activity in each limit cycle SOM is dynamic and has multiple winners, cross-modality association is a much harder problem.

Another class of work on multi-SOM architectures puts emphasis on building anatomically or physiologically realistic models of brain regions or neural pathways. An example is the model based on Wernicke–Lichtheim–Geschwind (WLG) theory of neurobiological language processing that incorporates simultaneous verbal and visual stimuli (Weems and Reggia, 2006), although SOMs are mainly used to process unimodal data separately. In Petreska and Billard (2006), an architecture containing three interconnected SOMs, each processing visual, tactile, and proprioceptive inputs, is built to study a visuo-motor imitation task, and how brain damages might affect the ability to imitate. Another multi-SOM architecture models grid cells in medial entorhinal cortex (MEC) and place cells in hippocampus, both related to the ability to understand current position in space and to navigate (Pilly and Grossberg, 2012). In this model, three lower-level SOMs representing grid cells and a higher-level SOM representing place cells form a two-layer hierarchy. Explicit lateral connections are used to implement competitive activation and learning instead of a one-step winner-takes-all process. A spiking-neuron version of the same model is also constructed, which entails more biological resemblance (Pilly and Grossberg, 2013).

Other SOM-based architectures are arranged in tree-like structures. Work in this category focuses more on computation and less on biological relevance. One example is detection of plagiarism at different document levels (e.g., paragraph, page, etc.) that are arranged in a tree structure (Chow and Rahman, 2009). A multi-layer SOM architecture is used to progressively integrate document features bottom-up, where

each level of the tree is processed by a different SOM. In addition to processing hierarchical data, tree-like SOM architectures can also effectively serve as a search tree to find the best matching unit faster than a flat map layer, especially when the number of nodes in the map layer is large. Architectures of this kind are referred to as hierarchical SOMs (HSOMs) (Henriques et al., 2012; Koikkalainen and Horppu, 2007), and there is also a variant based on spiking neurons (Tarek et al., 2014). To overcome the problem of the basic SOM that a map node may represent inputs that are too diverse, growing hierarchical SOMs (GHSOMs) allow each map node to spawn a descendent map layer dynamically, forming a tree-like architecture (Chattopadhyay et al., 2014; Rauber et al., 2002). The benefit of this method is having adaptive resolution for input space. A similar approach arranges a SOM of SOMs, where each node of a SOM (higher-level) is itself a SOM (lower-level) (Furukawa, 2009). The lower-level SOMs each represent a small region in input space, while the higher-level SOM manages relative relations among the lower-level SOMs. While the above tree-like architectures of SOMs tend to be effective in computational aspects, tree-like anatomical arrangement of cortical regions is not probable.

In summary, most past architectures are limited in using single-winner SOMs with static representation, and therefore face a problem similar to what simple recurrent SOMs encounter (Section 2.2). That is, the one-hot coded neural activity lacks sufficient coding capacity to be passed between SOMs in an architecture. While some work-arounds have been used, including passing the similarity measure between the nodes' weights and the input (Gustafsson and Paplinski, 2006; Kayacik et al., 2003; Lichodzijewski et al., 2002), passing winner node indexes (Liu et al.,

2015), passing winner's weights (Walter and Ritter, 1996), relying on leaky integrators (Carpinteiro, 1999), and extracting information regarding the clustered formed in the SOM (Guimarães, 2000), a more straightforward method would be to use multi-winner SOMs instead (Monner and Reggia, 2009; Weems and Reggia, 2006). This research is based on the latter method. Further, past architectures are mostly based on static representation or fixed-point dynamics, and thus do not account for continuously changing activity. While there are existing SOMs able to demonstrate oscillatory activity, how an architecture can be constructed using them is unclear.

— Chapter 3 —

Learning Limit Cycle Representations in SOMs

In the context of building brain-inspired neural architectures based on SOMs, a first step is to explore the use of dynamically oscillatory activity in a single SOM to represent external stimuli. Such activity is inspired by biological neural systems' prominently rhythmic oscillations and the strong relations of these oscillations to cognitive functions such as memory. However, as explained in the previous chapter, past SOMs have generally used a static representation for each input pattern, which is typically a single fixed spatial pattern in the map layer, despite that biological systems are highly unlikely to operate solely based on fixed spatial patterns. Although there are some exceptions, as was summarized in Chapter 2, their oscillatory activity is typically a behavior purposely built into individual map nodes, such as using phase oscillators or periodic oscillating spiking neurons. There is no surprise that SOMs made of such oscillating nodes can display oscillatory activity. On the other hand, it is less clear if oscillatory activity in a SOM can emerge from map nodes that are intrinsically non-oscillating, based on neural circuitry and self-organization.

A second limitation of past SOMs that was considered earlier is that their activity is typically tied to input stimuli directly. This means that an activity pattern in the map layer exists only for as long as its corresponding inputs are present (unless the activity

is artificially frozen, i.e., to stop updating SOM activity). An architecture composed of this type of SOMs would have strict timing constraints, since all computation must be done within the duration of inputs. This can potentially incur high control overhead if the inputs are transient. In contrast to this behavior, biological neural systems can generally maintain sustained neural responses to transient stimuli, allowing ongoing processing of the stimuli.

Motivated by these issues, here a new dynamic representation is investigated for use in SOMs for the first time. This representation involves not only spatial patterns but also temporal extensions in the form of map limit cycle attractors. Each input pattern/sequence becomes encoded by a short limit cycle attractor in the state space of a multi-winner SOM, regardless of whether input stimuli form temporal sequences or are static spatial patterns. The limit cycles are self-organized, rather than manually specified. They are self-sustained and persist after the input that triggered them terminates, and they can therefore be viewed as a candidate representation for working memory. The length of a learned limit cycle representation is found to generally not be the same as that of the corresponding input sequence that it represents.

Computational experiments are used to address several questions concerning the proposed use of limit cycle representations in SOMs. First, what are the conditions and training methods necessary for acquiring limit cycle attractors in a SOM with recurrent feedback connections? Second, since it is unclear a priori, does map formation still occur over a lattice like it does with more traditional SOMs, and if so, does it occur robustly in the context of persistent limit cycle activity? Third, do self-organized limit cycle representations demonstrate beneficial characteristics for encoding information? For

example, does this way of representing information provide resistance to perturbations, increased uniqueness, and the ability to generalize to new inputs?

In the following, first a SOM model forming a dynamical system is introduced. This SOM is then used to encode binary inputs that represent phoneme sequences of spoken words and their corresponding visual images. Conditions for obtaining limit cycle attractors are determined, and the properties of the resulting limit cycles are studied. Next, the SOM is used to represent continuous 2D space, where there are infinitely many possible inputs. How the SOM encodes inputs in this case, as well as its ability to generalize to new inputs, are investigated. Finally, a brief summary and discussion conclude this chapter.

3.1 A SOM Model with Limit Cycle Dynamics

One-step multi-winner SOMs with locally recurrent feedback connections, modified from Schulz and Reggia (2004) so that they now support continuous post-stimulus activation dynamics, are used in this study. A key difference that separates this study from other work is that SOMs with feedback connections allow activation dynamics to continue running and adapting after input sequences are over. The modified SOMs, referred to as *limit cycle SOMs* hereafter, generate sparsely-coded activation patterns with multiple simultaneous node activations, and retain time-varying activity indefinitely after external input ends.

The artificial neurons (nodes) in the limit cycle SOMs considered here are organized in a 2D rectangular grid. The neighborhood of a node i is the set of nodes that are

located within a radius r of the node i. Box distances are measured between nodes on the grid. Formally, let $\mathbb{N}_i$ denote the neighborhood of a node i within radius r:

$$\mathbb{N}_i = \{k \mid k \neq i \text{ and } d(i,k) \leq r\}, \tag{3.1}$$

$$d(i,k) = \max\left(|row_i - row_k|, |column_i - column_k|\right). \tag{3.2}$$

In the computational experiments that follow, if more than one SOM is present in a multi-SOM architecture, each SOM receives streams of input via synaptic connections from one or more upstream components, such as external inputs, other SOMs, and/or itself (i.e., the latter via recurrent feedback connections). The set of connections coming from the same upstream component is called a *channel* (or *pathway* at times). The set of connections for a channel can be full or topographic. *Full connections* are often used for connecting a non-SOM component, e.g., external inputs, a hidden layer, etc., to a SOM, where a link exists between every pair of upstream and downstream neurons. *Topographic connections* are often used for connecting two SOMs (including direct feedback connections in a SOM), where a node i in the downstream SOM receives connections only from the neighborhood of its corresponding node i' in the upstream SOM ($row_i = row_{i'}$, $column_i = column_{i'}$). Figure 3.1 shows a simple example of a SOM whose structure resembles those in the experiments in this study. It has two channels. The fully-connected channel provides afferent input, and the topographic recurrent channel provides time-delayed feedback. The recurrently connected SOM shown here can be interconnected with other SOMs via additional channels, as occurs in some of the experiments described later.

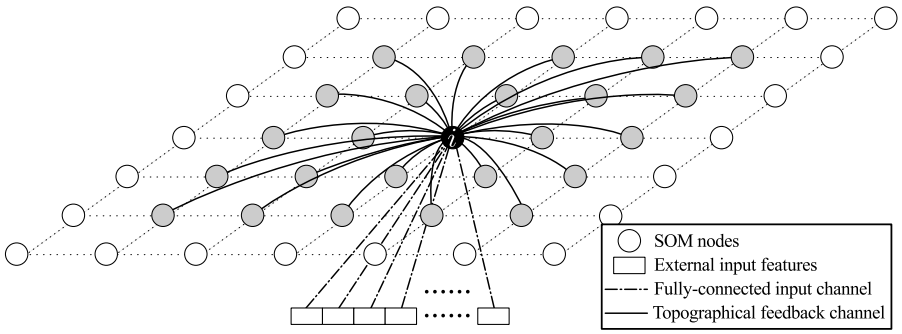

Figure 3.1: A small example 7×7 SOM for illustrative purposes. The SOM has two channels: a fully-connected afferent channel and a topographic recurrent channel (radius 2). For readability, only a small number of nodes are illustrated and incoming links for only map node i are drawn. SOMs having structures similar to this one but much larger serve as a basic neural component throughout this dissertation.

Let $\boldsymbol{x}_j(t)$ denote the input vector that a SOM receives from channel j at time t. The value of $\boldsymbol{x}_j(t)$ is often different at different times t, and therefore $\boldsymbol{x}_j(t)$ is a temporal sequence. This temporal sequence can represent an external input sequence or a sequence of changing activity of an upstream neural component, depending on where channel j takes inputs from. The net input for each node i receiving one or more channels j at time t can be generally expressed as

$$h_i(t) = \sum_j \alpha_j \left(\boldsymbol{w}_{ij} \cdot \boldsymbol{x}_j(t) \right), \tag{3.3}$$

where $\boldsymbol{w}_{ij}$ denotes the weight vector at node i for channel j, and α_j is a constant parameter specifying relative weighting among channels. For recurrent connections, the input is delayed by one time step, i.e., $\boldsymbol{x}_j(t) = \boldsymbol{a}(t-1)$, where $\boldsymbol{a}(t)$ is the activation vector (output vector) of the SOM. As a concrete example, the net input of SOM

nodes shown in Figure 3.1 can be written as:

$$h_i(t) = \alpha_1 \left(\boldsymbol{w}_{i1} \cdot \boldsymbol{I}(t) \right) + \alpha_2 \left(\boldsymbol{w}_{i2} \cdot \boldsymbol{a}(t-1) \right), \tag{3.4}$$

where channel 1 represents afferent inputs, channel 2 represents recurrent feedback inputs, $\boldsymbol{I}$ represents an external input sequence. Since the recurrent feedback connections are topographical, $\boldsymbol{w}_{i2}$ has non-zero values only for elements corresponding to nodes that reside in i's radius 2 neighborhood. It is assumed the map's activation pattern is initially zero, i.e., $\boldsymbol{a}(t=-1) = \boldsymbol{0}$.

A one-step multi-winner activation rule is used to determine $\boldsymbol{a}(t)$ for $t \geq 0$ based on $\boldsymbol{h}(t)$, where each $a_i(t) \in [0,1]$ is calculated as:

$$\mathbb{W}(t) = \{ k \mid h_k > h_l, \forall l \in \mathbb{N}_k \}, \tag{3.5}$$

$$a_i(t) = \min \left(1, \sum_{k \in (\{i\} \cup \mathbb{N}_i) \cap \mathbb{W}(t)} \gamma^{d(i,k)} \right). \tag{3.6}$$

Winners $\mathbb{W}$ are nodes that have the highest net input in their local competition neighborhood $\mathbb{N}$ (of radius r; see Equations 3.1, 3.2). In extremely rare situations where $h_k = h_l$, ties are broken arbitrarily (e.g., the one having lower node index wins). The activation level for each node is determined by how close it is to the winner nodes. The winners themselves are maximally activated, while their surrounding nodes have lower activation levels based on their distance from the winners. Together each winner node and its neighbors form a peak of activation centered at the winner. The shape of the peaks depends on the decay parameter γ ($0 \leq \gamma < 1$). A smaller γ makes steeper

peaks.

The weight adaptation for each afferent channel j follows competitive Hebbian learning:

$$\hat{\boldsymbol{w}}_{ij}(t+1) = \boldsymbol{w}_{ij}(t) + \mu_j a_i(t)\boldsymbol{x}_j(t), \tag{3.7}$$

$$\boldsymbol{w}_{ij}(t+1) = \hat{\boldsymbol{w}}_{ij}(t+1)/\|\hat{\boldsymbol{w}}_{ij}(t+1)\|_2, \tag{3.8}$$

where μ_j is the learning rate, and $\|\cdot\|_2$ represents the l_2-norm (i.e., Euclidean norm).

Each direct feedback channel, such as the recurrent connections in Figure 3.1, is adapted using temporally asymmetric Hebbian learning (Schulz and Reggia, 2004), which is based on biological evidence that the efficacy of a synapse is strengthened if presynaptic firing precedes the postsynaptic firing in a 20 to 50 ms time window (Bi and Poo, 1998; Feldman, 2012; Finnerty et al., 2015; Markram et al., 1997; Zhang et al., 1998). The weight value $w_{ijk} \in \boldsymbol{w}_{ij}$ for each connection from node k to node i, $i \neq k$, via channel j is updated as:

$$\hat{w_{ijk}}(t+1) = w_{ijk}(t) + \mu_j a_k(t-1)\max\left(0, a_i(t) - a_i(t-1)\right), \tag{3.9}$$

$$w_{ijk}(t+1) = \hat{w_{ijk}}(t+1)/\textstyle\sum_l \hat{w_{ijl}}(t+1). \tag{3.10}$$

The term $\max\left(0, a_i(t) - a_i(t-1)\right)$ specifies that the increase, rather than the value, of the activation level is taken into consideration. Note that if each w_{iji} ($k = i$; a self-link that is originated and terminated at the same node i) was not treated differently from non-self-links, it would be strengthened during learning whenever a_i increases and

soon dominate $\boldsymbol{w}_{ij}$. For this reason, each self-link w_{iji} is fixed to a constant parameter β:

$$\forall t, w_{iji}(t) = \beta \tag{3.11}$$

A SOM can be trained with both temporal sequence inputs and static inputs. Suppose training data contain a set of temporal sequences $\{I_k \mid k = 0, 1, \cdots\}$. Each sequence I_k is an ordered list of vectors $\boldsymbol{I}_k^{(0)}, \boldsymbol{I}_k^{(1)}, \cdots, \boldsymbol{I}_k^{(|I_k|-1)}$, where $|I_k|$ represents the length of sequence I_k (the length can be different for different k). Static input can be viewed as a special-case temporal sequence, where each vector $\boldsymbol{I}_k^{(l)}$ stays the same for $0 \leq l < |I_k|$. The value of $|I_k|$ in this case is assumed to be short, e.g., 2, and fixed for all k. In both cases, given I_k as an afferent input sequence via channel j of a SOM, the value of the SOM's input $\boldsymbol{x}_j$ can be generally expressed as:

$$\boldsymbol{x}_j(t) = \begin{cases} \boldsymbol{I}_k^{(t)} & \text{if } 0 \leq t < |I_k|; \\ \\ \mathbf{0} & \text{if } t \geq |I_k|. \end{cases} \tag{3.12}$$

This means that the input activity of channel j becomes $\mathbf{0}$ after the input sequence I_k runs out.

Training of a SOM is done for a number of epochs. In each epoch, all training sequences are presented to the SOM once in a random order. Time t is reset to 0 before presenting each sequence. For each input sequence, learning occurs both while the temporal sequence I_k is being presented *and afterwards*, during which the activity of the SOM continues to evolve without afferent input. Weight adaptation is done according to Equations 3.7–3.11. The learning rates μ_j and the peak parameter γ

Table 3.1: Parameters for nonlinearly decreasing functions used during training.

z	z_{init}	z_{fin}	z_{infl}	z_σ
μ_j for afferent channels	0.44	0	0.4	0.0001
μ_j for direct feedback channels	0.62	0	0.8	0.04
γ	0.37	0	0.2	0.16

decrease nonlinearly as the epoch number progresses, according to the function

$$z = z_{\text{fin}} + \frac{z_{\text{init}} - z_{\text{fin}}}{1 + \exp(\frac{\phi - z_{\text{infl}}}{z_\sigma})}, \tag{3.13}$$

where z is a parameter (μ_j or γ) and ϕ is the fraction of epochs completed. Relevant parameter values used in the work described below are listed in Table 3.1. This function simulates the widely-adopted two phase training often used with SOMs, namely a rough organization phase with large learning rates and neighborhood sizes, followed by a fine-tuning phase with small learning rates and neighborhood sizes.

A critical element in my approach is that a SOM is allowed to continue running and learning after an input sequence is over, for another τ time steps (a fixed parameter). This period is referred to as the *continuation time*. During this time, although afferent inputs are no longer provided, the SOM has non-zero activity thanks to recurrent feedback connections. The map layer forms a dynamical system with changing activity. Specifically, when I_k is presented for training, the SOM is run and its weights adapted for all time steps $t = 0, 1, \cdots, (|I_k| - 1 + \tau)$. During the last τ time steps, activation and adaptation are performed as usual, although no external input is being provided. This results in the afferent weights being unchanged in this time period because $\boldsymbol{x}_j(t) = \mathbf{0}$ (see Equation 3.7), while recurrent weights adaptation is continued. Notice

Table 3.2: Summary of limit cycle SOM parameters.

Parameter	Meaning
r	Radius of local competitive neighborhoods (Equation 3.5)
α_j	Relative weighting for channel j when computing net input (Equation 3.3)
β	Fixed weight for same-node feedback connections
μ_j	Learning rate for channel j (Equations 3.7, 3.9)
γ	Peak parameter determining the steepness of activation peaks (Equation 3.6)
τ	Post-stimulus continuation time

that except at the beginning of each input sequence, the activation levels of a SOM are never reset to $\mathbf{0}$, so that the recurrent connections can learn the temporal relations between activation patterns in consecutive time steps during the continuation time. This method allows ongoing activation dynamics of SOMs to self-organize.

To summarize, the SOM model used in this study contains locally recurrent connections that result in the SOM being a dynamical system in terms of its activation states. It uses one-step multi-winner activation and learns in an unsupervised fashion based on Hebb's rule, both in terms of basic Hebbian learning (afferent connections) and temporally asymmetric Hebbian learning (recurrent connections). Activation and learning are continued after input sequences end, allowing activity dynamics to self-organize without afferent inputs. Parameters are summarized in Table 3.2.

3.2 Representing Binary Sequences

Given the SOM model described above, it is unknown what activity dynamics will be generated, or if smooth maps like those in conventional SOMs will still form

while employing continually changing activity patterns over the map layer. It is also unclear what the relative advantages are for different types of attractors to serve as representations, and how dynamic and static representations compare. Therefore, in this first basic experiment, a relatively small set of binary inputs is considered to understand these issues.

3.2.1 Methods

This experiment uses the same data set as in Weems and Reggia (2006), which describes 50 objects using two modalities: English phoneme sequences (auditory words) and bitmap images (visual) of corresponding objects. Let P_k be a temporal sequence of phonetic stimuli that is the name of an object in the dataset, e.g., $P_{\text{apple}} =$"/ae/, /p/, /l/" for apple (see Figure 3.2 left). Each phoneme is encoded by a binary vector of 34 distinctive features. The phoneme sequences each contain 2 to 9 phoneme feature vectors. Similarly, let V_k be a pictorial stimuli that is a single bitmap image (50×50) of the object that P_k names, e.g., the image of an apple (Figure 3.2 right). Therefore, each object k in this dataset can be described by a tuple $\langle P_k, V_k \rangle$.

A SOM is trained with either the set of phoneme sequences or that of visual images. After training, the activation dynamics naturally occurring in each map during continuation time are designated to be the dynamic representation that encodes the corresponding input stimuli. The peak parameter is set to $\gamma = 0$, i.e., all winners have activation level 1 and non-winners 0, and all learning rates $\mu_j = 0$. For long enough post-training continuation time (e.g., 200 time steps), the dynamics can be

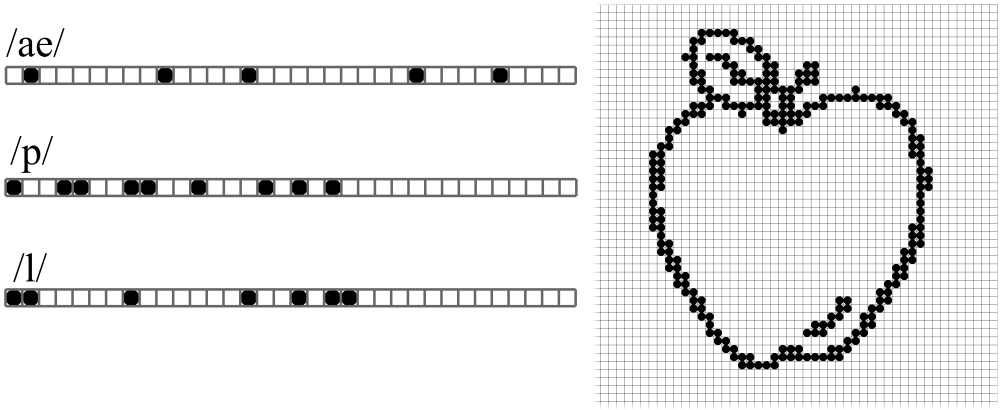

Figure 3.2: Sample data. Left: an input phoneme sequence for the word "apple". The three 34-bit input patterns corresponding to each phoneme are shown (black = 1, white = 0). Right: the corresponding bitmap image of an apple.

qualitatively classified as one of three types of attractors: fixed point, limit cycle, and "complex". Let R_k be the dynamic representation encoding an input stimuli/sequence (P_k or V_k). Each R_k is a time-ordered list of map activity spatial patterns defined as follows for different types of attractors. With a fixed point attractor, the dynamics eventually reach a state where further updating of the activation levels results in the same state. In this case, R_k contains only this particular fixed-point state. For limit cycles, R_k consists of an ordered list of periodically repeating distinct states. The number of states in R_k is the length of the limit cycle. Fixed points and limit cycles are both referred to as *simple attractors* in the following. Dynamics that do not show apparent regularity during the continuation time are classified as being a *complex attractor* (this includes chaotic attractors and potentially very long limit cycles where no state has yet repeated). In this case, R_k contains the whole list of states occurring during the continuation time.

For the sake of comparison, control experiments are also conducted with a contin-

uation time $\tau = 0$ during training. In the control experiments, static representations are simulated by taking the single activation pattern occurring at the end of each input sequence to be that sequence's representation, as has been done in some other past studies (Schulz and Reggia, 2004). Setting $\tau = 0$ means that during training, there is no additional adaptation time after each input sequence. During testing, no attractor dynamics are observed beyond the end of each input sequence. Instead, the static representation during testing is a single pattern, denoted as $R_k^{\text{static}} = \boldsymbol{a}(|P_k| - 1)$, if the SOM takes phonemes as input.

The following results are obtained using a 30×20 SOM with two channels. The first channel receives input phoneme sequences via fully-connected connections, and the second forms direct feedback via topographic connections, providing much the same structure as in Figure 3.1 but with a larger map region. For clarity, let the length of continuation time during training be an adjustable parameter τ, and that after training be a constant of 200 time steps. The SOM is trained for 1000 epochs with different τ and β (the fixed weight of same-node feedback links) values, while the other parameters are fixed as: $r = 2, \alpha_1 = 0.64, \alpha_2 = 0.36$ (see Table 3.2 for meaning). After training, the SOM is run one last time without adaptation, using the same data set and 200 continuation time steps. Then both the static and the dynamic representations are recorded for analysis. The results reported below are based on 20 independent simulations per data point, each simulation having different initial weights (random). The error bars below indicate one standard deviation. A separate SOM of the same structure is also trained with image inputs for observing map formation (discussed in Section 3.2.7).

3.2.2 Attractor Formation

The map region activity state attractors formed are largely dependent on two parameters: β, the constant same-node recurrent weight for each node, and τ, the value of post-input continuation time during training.

Large $|\beta|$ values would dominate all other recurrent connections in the same channel, and marginalize their contributions to the attractor dynamics. As a result, the SOM would operate based on blind activation (for $\beta \gg 0$) or deactivation (for $\beta \ll 0$) of the current winners, rather than on self-organization of the adaptable recurrent weights. Small $|\beta|$ values, e.g., $\beta \in [-1, 1]$, on the other hand, make the contributions of the same-node recurrent weights comparable to the other adaptable recurrent weights. Figures 3.3(a)-(b) show the types of attractors, (c) the number of time steps preceding simple attractors, and (d) the lengths of simple attractors, that are formed using different β values. For most of the negative and small positive β, i.e., $\beta < 0.15$, nearly all are small limit cycles. In the range $0.15 < \beta < 0.5$, sizes of the limit cycles increase and a peak occurs at $\beta = 0.35$, at which point the dynamics also take longer to settle in the limit cycles. In the range $0.2 < \beta < 0.5$, complex attractors are also formed in addition to large cycles. As β increases beyond 0.5, fixed points soon take over.

Figures 3.3(e)-(f) show how continuation time τ affects attractor formation. At $\tau = 0$, which is what conventional training methods imply, most of the resulting attractors are complex. This is because the recurrent connections are under-trained for the situation where the SOM is run without external input. At $\tau = 5$, some complex attractors are reduced to large limit cycles. For $\tau > 15$, the SOM stably produces

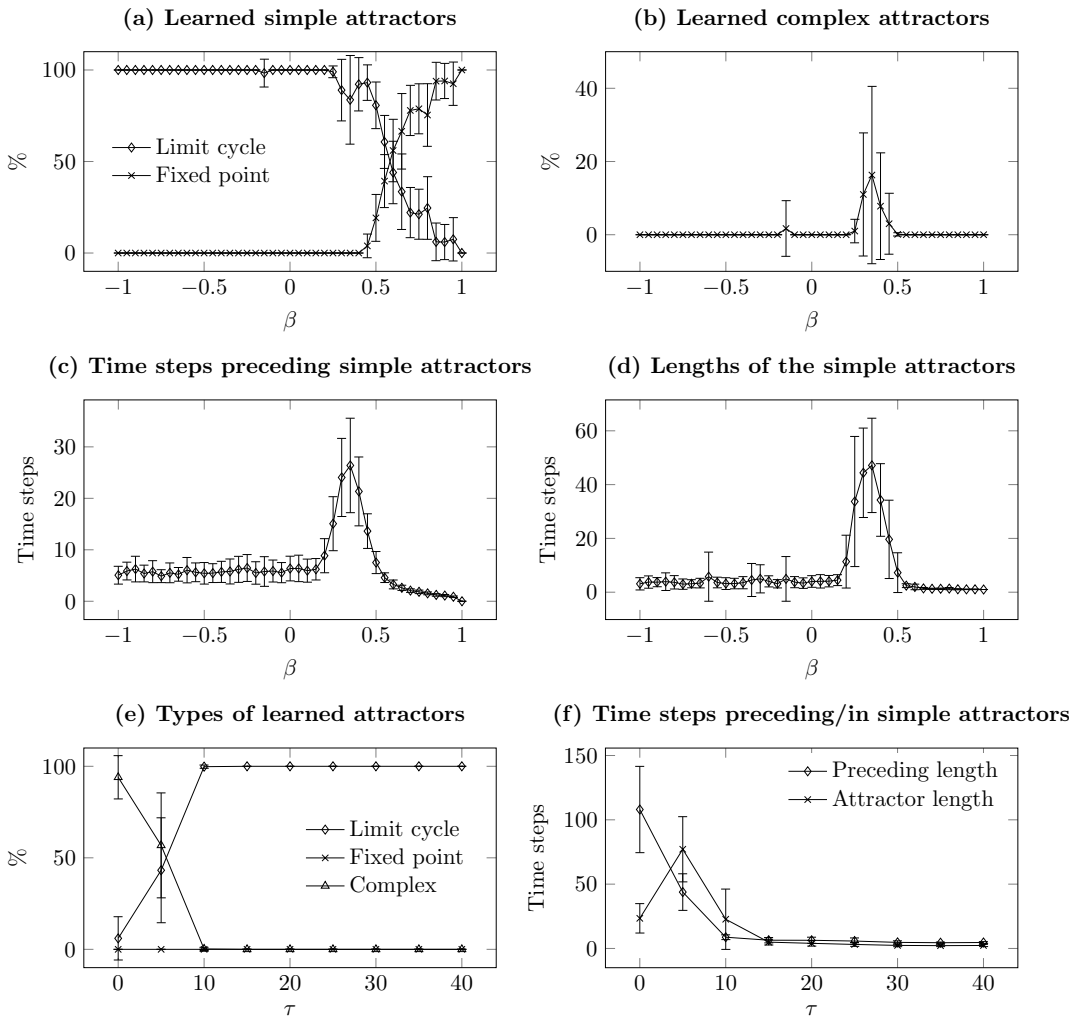

Figure 3.3: (a)-(d) show the effects of β (while τ is fixed at 20), and (e)-(f) the effects of τ (while β is fixed at 0), on the formation of attractors. Among all attractors formed after training, (a), (b), and (e) show the percentages that are fixed points, limit cycles, and complex attractors. For simple attractors (i.e., fixed points and limit cycles), (c) and (f) show the numbers of time steps it takes for the dynamics to settle in the attractors. (d) and (f) show the lengths of the simple attractors.

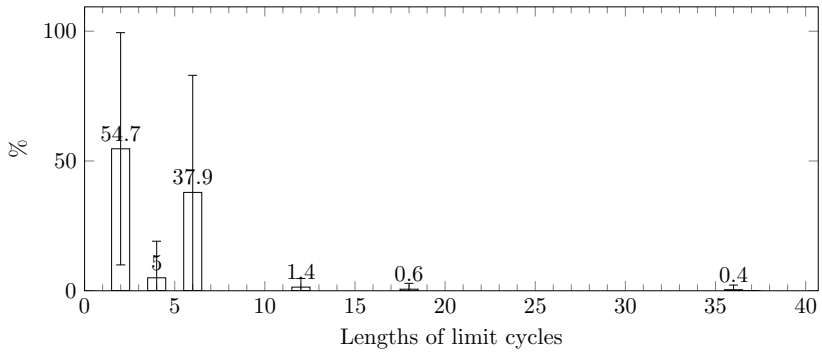

Figure 3.4: The length distribution of the limit cycles that are formed using $\beta = 0$ and $\tau = 20$.

small limit cycles. A histogram of the cycle lengths are depicted in Figure 3.4. The cycle lengths tend to be multiples of 2 or 3.

To be useful for subsequent processing, learned representations need to be quickly reproducible, or retrievable, in SOMs. Retrieving a dynamic representations means restoring all constituent activation states. Therefore, complex attractors need to be avoided because their irregular activation makes them nearly irreproducible. Large cycles are undesirable as well because restoring them takes a longer time than small cycles or fixed points. Limit cycles that occur late are not preferred for the same reason. Therefore, according to the results shown in Figure 3.3, a feasible range of parameters lies in $\tau \geq 15$ and $\beta \in [-1, 0.15] \cup [0.5, 1]$. In the following subsections, this range will be narrowed down when stability and uniqueness are taken into consideration.

3.2.3 Attractor Stability

The robustness of dynamic representations directly depends on the stability of their attractors. To assess this stability, an activation state $\boldsymbol{b}$ in each attractor representation

R_k is perturbed for all i as

$$b_i' = \begin{cases} b_i - \nu Z & \text{if } b_i = 1; \\ \\ b_i + \nu Z & \text{if } b_i = 0, \end{cases} \tag{3.14}$$

where ν is the amplitude of perturbation and Z a uniform random number in $[0, 1)$. Then the SOM's initial activation is set to this perturbed pattern, $\boldsymbol{a}(t = -1) = \boldsymbol{b}'$, and run for 200 time steps without external input. If the resulting dynamics successfully restore the attractor R_k, R_k is marked stable. The stability metric in Figure 3.5 is measured as the percentage in all attractors that are marked stable.

In Figure 3.5a, stability reaches the highest value around $\beta = 0$. Larger β causes stability to drop dramatically, which coincides with the β range where large cycles, complex, and fixed point attractors are formed (see Figure 3.3). As β decreases from 0, although the attractors remain small cycles, stability drops gradually. In Figure 3.5b, stability increases quickly as τ increases from 0 to 10, and gradually decreases as τ increases further. This indicates that prolonged post-input sequence training does not improve stability. Therefore, a preferred range of parameters is $\beta \in [-0.1, 0.1]$ and $\tau \in [15, 25]$.

3.2.4 Representation Uniqueness

Generally, uniqueness of representations is regarded as a desirable property for encoding a set of items (words, images, etc.). The more unique a representation is, the less likely its corresponding item is to be confused with other items in subsequent processing, such

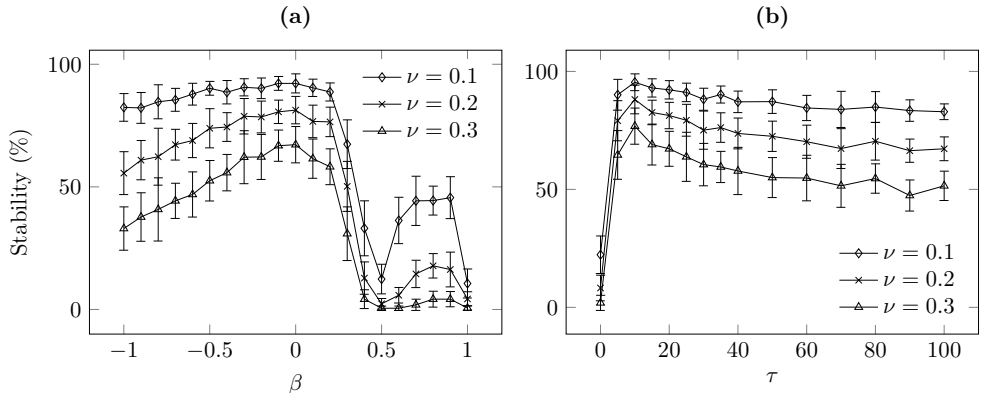

Figure 3.5: Stability of the attractors as measured by the percentages of attractors that are restored from random perturbation. ν denotes the amplitude of noise added to a state in each attractor. In (a), τ is fixed at 20; in (b), β is fixed at 0.

as in a classification task. The distinction between two conventional representations can be calculated as the distance between the two static states in a straightforward way. However, since the proposed encoding method allows each representation to contain multiple states, the distance function must be generalized to accommodate two sequences of states:

$$\text{dist}(R_k, R_l) = \min_{\boldsymbol{p}\in R_k, \boldsymbol{q}\in R_l} \|\boldsymbol{p} - \boldsymbol{q}\|_1, \tag{3.15}$$

where $\boldsymbol{p}, \boldsymbol{q}$ are any constituent activation states in R_k, R_l, respectively. This distance reflects the minimum number of nodes whose activation must be inverted (i.e., changing 0 to 1 or vice versa) to cause one representation's attractor (e.g., a limit cycle attractor) to be converted into another's. It is a "conservative" lower bound on distances between two sequences: all the individual activity patterns in R_k and R_l are at least as different as this distance. In other words, if this distance between two sequences is larger than the distance between two static representations, all the individual patterns in the two

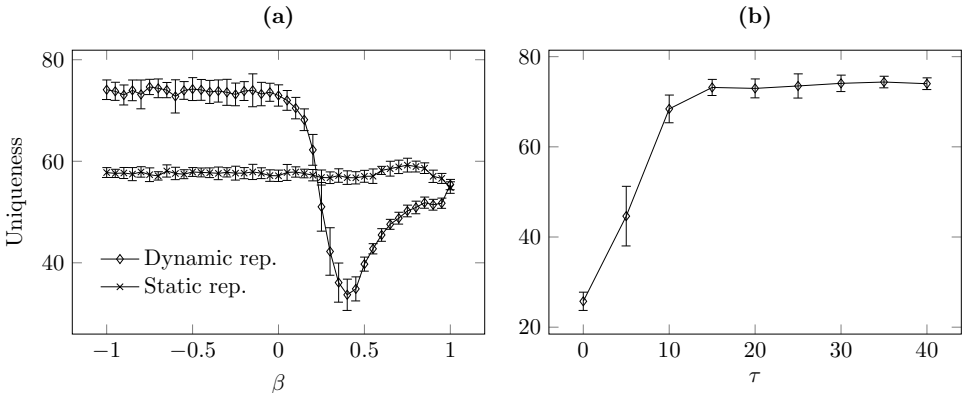

Figure 3.6: Uniqueness of the dynamic representations (attractors) as measured by the averages of the nearest distance among all combinations of the patterns in each pair of attractors. For comparison, the average distances of static representations are also shown in (a). Higher degrees of uniqueness reduce the chances that two representations are confused in subsequent processing. In (a), τ is fixed at 20; in (b), β is fixed at 0.

sequences are more distinct than the two static patterns, and therefore one can be confident that the dynamic representations are more distinct in general. Note that this distance function also works for representations that contain one state, such as static representations or fixed point attractors. Using this distance function, the uniqueness metric can be defined as the average distance over all pairs of representations.

Figure 3.6 shows the uniqueness metric for different β and τ values. In Figure 3.6a, the range of β that produces small limit cycles ($\beta < 0.15$) yields about 30% better uniqueness than static representations. For $\beta > 0.15$, uniqueness drops significantly for large cycle, complex, and fixed point attractors. This means that small limit cycles ($\beta < 0.15$) are generally more unique/distinct from one another than fixed-point and complex attractors ($\beta > 0.15$). In Figure 3.6b, uniqueness reaches its highest when $\tau \geq 15$.

Table 3.3: Comparisons between pre- and post-training attractors.

Properties	Pre-training		Post-training ($\tau = 20$)	
Types of attractors (%)				
Fixed points	0	(SD=0)	0	(SD=0)
Limit cycles	61.9	(SD=20.1)	100	(SD=0)
Complex	38.1	(SD=20.0)	0	(SD=0)
Time preceding simple attractors	99.9	(SD=11.2)	6.4	(SD=2.4)
Simple attractor length	15.6	(SD=5.3)	4.0	(SD=2.2)
Stability metric with $\nu = 0.1$ (%)	9.0	(SD=7.4)	92.2	(SD=3.9)
Uniqueness metric	35.4	(SD=3.3)	73.0	(SD=2.1)

3.2.5 Effects of Training

To show how training affects the resulting limit cycles, Table 3.3 summarizes the properties of the limit cycles obtained in a SOM before and after training. The values are obtained based on 20 independent simulations initialized using different random weights and $\beta = 0$. Before training, the attractors are either complex or limit cycles that occur long after inputs end. After training, all attractors become limit cycles that are both shorter and start much earlier. In both cases, there are no fixed point attractors. Further, the post-training limit cycles are much more stable and unique compared with the pre-training attractors.

3.2.6 Oscillatory Activation: An Example

As an example, two limit cycle representations in a typical simulation, using $\beta = 0, \tau = 20$, are visualized in Figure 3.7. The learned representation for the phoneme sequence of "butterfly" is a limit cycle of length 2 (Figure 3.7(a)). The two alternating activation patterns of the SOM are shown in the top of Figure 3.7(a). Each cell

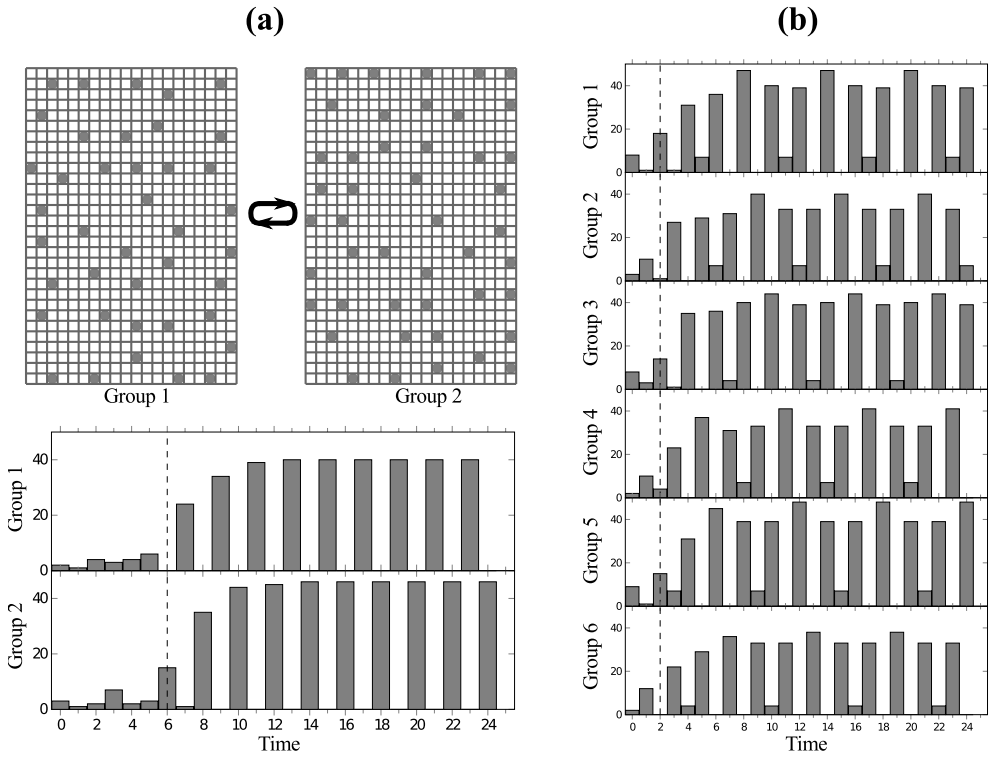

Figure 3.7: Illustration of two examples of limit cycles representing the phoneme sequences (a) "butterfly" and (b) "apple". The limit cycles are obtained using $\beta = 0$ and $\tau = 20$. In (a), a limit cycle of length 2 is shown. The top part shows the alternating activation patterns in the limit cycle, where activated SOM nodes are marked using dark cells. All activated nodes in one pattern are arbitrarily designated to be group 1, and those in the other pattern group 2. The bottom part of (a) shows the number of nodes in each group that are turned on at each time step. The dashed line indicates the end of the input phoneme sequence. In (b), another limit cycle of length 6 is shown. Nodes may be assigned to multiple groups if they are activated more than once in one pass through the limit cycle.

corresponds to a node in the SOM, and dark cells indicate nodes that are activated. Designate all activated nodes in one pattern (left) to be group 1, and all activated nodes in the other (right) to be group 2. The bottom of Figure 3.7(a) shows the number of nodes in group 1 and 2 that are turned on at each time step. The dashed line at $t = 6$ indicates when the phoneme input sequence ends. Neither group is significantly activated until the input sequence is finished, shortly after which both

groups become fully activated and demonstrate oscillatory patterns.

Another limit cycle of length 6 that encodes the phoneme sequence for "apple" is illustrated in Figure 3.7(b). The nodes that are activated in each of the six distinct activation patterns in the limit cycle are designated to be groups 1–6. Nodes may be assigned to more than one group if they are activated more than once in a cycle. The times where a group is fully activated is indicated by the tallest bars in each chart. Although a majority of the nodes in each group is activated every other time step, at least some of them are activated every three time steps as indicated by the shorter bars. Closer inspection reveals that other than 2 and 3, some nodes have a period of 6 time steps.

Finally, note that these examples also illustrate a general observation of these experiments that the length of a representing limit cycle is not generally proportional to the length of the corresponding input sequence. The phoneme sequence for "butterfly" is significantly longer than that of "apple", yet their limit cycle representations are 2 and 6, respectively.

3.2.7 Map Formation

With multi-winner activation, limit cycle SOM dynamics, and post-input continuing adaptation, it is unclear in advance whether map formation similar to that in conventional SOMs will still occur. The SOM for phoneme sequence processing is used to examine this issue, based on the parameters $\beta = 0, \tau = 20$ that have now been demonstrated to consistently generate small limit cycles. In each of 20 independent

simulations, it was found that self-organizing maps of input patterns formed reliably in the presence of limit cycle representations.

Figure 3.8 shows a representative example of the afferent weights of a SOM from a typical simulation. Map formation is clearly observed. By comparing pre-(a) and post-training(b) weight values, the overall lighter shades after training indicate that the weight vectors are more tuned to the phoneme data. Similar phonemes are grouped in close proximity. For instance, the post-training vowels have become grouped in "islands" of nodes (highlighted in the figure by thick borders) in a "sea" of consonants. That there exist multiple vowel islands instead of a single island is caused by the multi-winner activation, where local winners separate from each other coexist. Also notice that most of the vowels are often surrounded by /l/ and /r/, the consonants having the most similar features to vowels.

Limit cycle representations formed in a SOM have no obvious correlations with the SOM's afferent weights. For example, limit cycles that activate a node labeled /f/ in Figure 3.8(b) do not necessarily encode phoneme sequences containing /f/. Limit cycle representations are a result of post-stimuli self-organization of feedback weights, which is only indirectly related to afferent inputs. They can be viewed as abstractions of the input sequences.

A U-matrix (Ultsch, 1993) analysis of the phoneme map is shown in Figure 3.9. The U-matrix values, calculated as the similarity (inner product) of the adjacent weight vectors, are shown as greyscale in each cell. The lighter the shade, the smoother the weight changes are at the location. Before training (Figure 3.9(a)), the U-matrix values are mostly distributed in the mid-range (0.5–0.7) without prominent pattern,

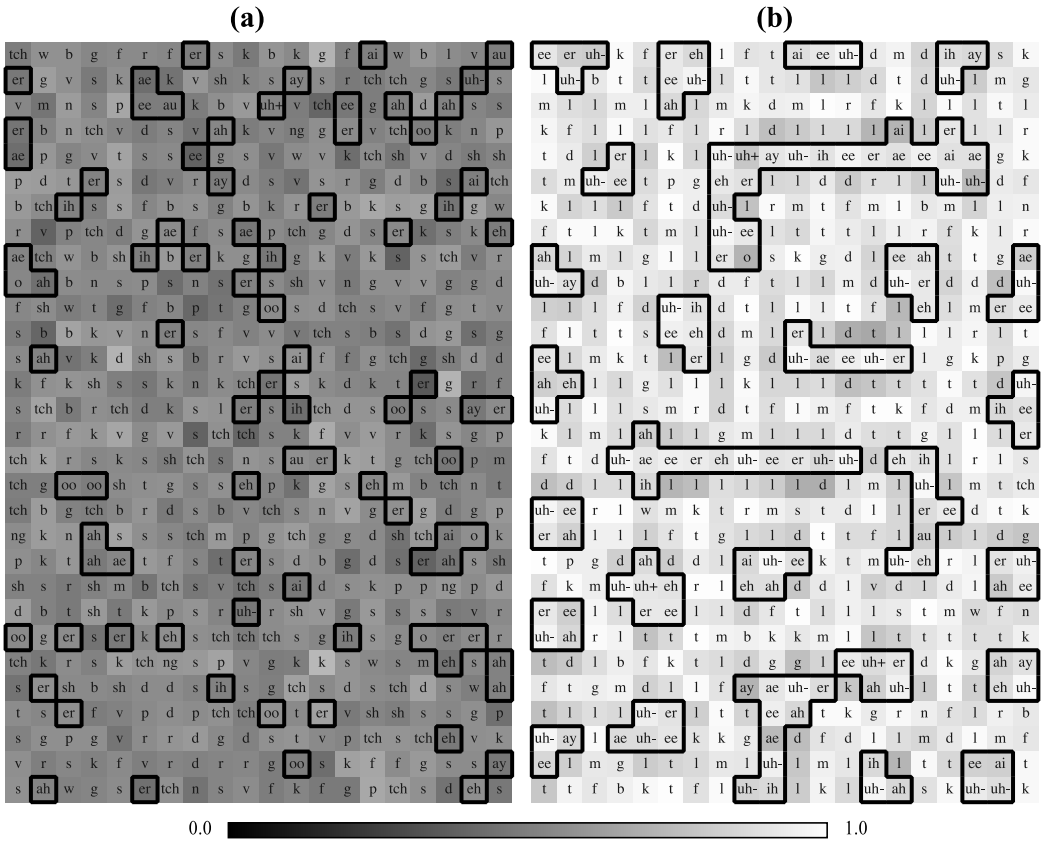

Figure 3.8: A labeled SOM in a typical simulation (a) before and (b) after training using phoneme sequence data. The phoneme label in each node is the one that best matches (using an inner product metric) the node's weight vector in the afferent channel. Degrees of matching are shown by shading in the cells. Lighter color indicates a better match. Cells labeled with vowels are highlighted by thicker borders and are seen to be clustered in (b). ($\beta = 0$, $\tau = 20$)

whereas the post-training map (Figure 3.9(b)) shows improved contrast: clusters of lighter-shaded regions ("valleys") emerged and are separated by narrow and darker-shaded regions ("walls"). The shades of most cells (74.5% of the cells) are lighter after training, indicating that the map becomes smoother in most regions after training. The darker cells indicate that greater sudden jumps of weights occur after training, although cells of this type are relatively few. Although the overall smoothness may not be as good as in a conventional SOM, due to the use of a multi-winner SOM, the

U-matrix visualization clearly suggests the formation of a piecewise smooth map. To further improves map smoothness, a separate experiment was performed where the hard binary inputs 0 and 1 were replaced by 0.1 and 0.9, respectively. The resulting U-matrix is shown in Figure 3.9(c). While the shading patterns are not qualitatively different, the overall shades become lighter than in Figure 3.9(b), including those "walls", which indicates an overall smoother map (95.6% of the U-matrix cells became lighter after training). Unfortunately, this change causes the number of unique limit cycles to decrease too, and therefore I continue using hard binary inputs for the rest of the experiment. Overall, the above visualizations indicate that map formation indeed emerges in the SOM variant and the modified training process that this study uses.

A separate SOM was trained using fixed image inputs. The SOM model and the parameters are identical to that used in the above experiments, except that the number of afferent weights is increased to 2500 to account for 50×50 pixels of the images. Each image is presented for a fixed number of time steps, while the SOM is trained using $\beta = 0$ and $\tau = 20$. In all of the 20 independent simulations, limit cycles form, even though there is a fixed input pattern rather than a temporal sequence of inputs. Figure 3.10 (top) illustrates a typical example of the afferent weights for each map node after training, where the 2500 weight values for each map node are visualized as a 50×50 image. Darker pixel shades indicate higher weight values. An overview of the entire map (left in Figure 3.10 (top)) shows map formation where images occupying similar spatial locations tend to occur in clusters having similar overall shapes. For instance, horizontally flat images are grouped together in the top two rows of the magnified view of the encircled region, and this is pictured on the

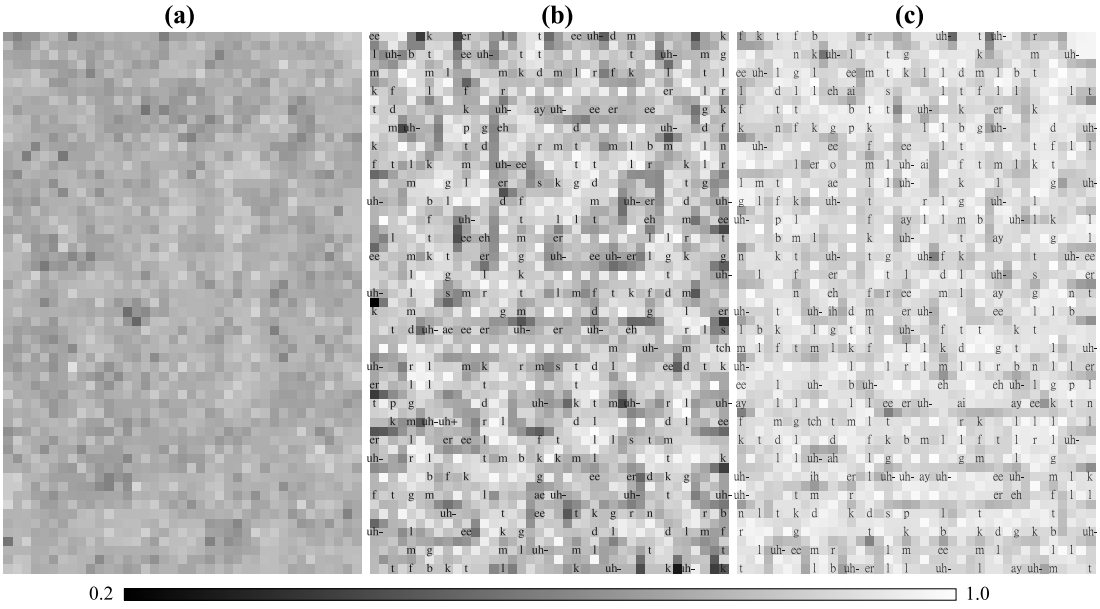

Figure 3.9: The U-matrices corresponding to the SOM in Figure 3.8. (a) and (b) are the results before and after training, respectively. (c) shows the result of training the SOM using blurred binary inputs (i.e., using values $\{0.1, 0.9\}$ to replace hard binary values $\{0, 1\}$). The shade of each cell represents its U-matrix value, which indicates the similarity (inner product) of the weight vectors between that node and its neighbors. The greater the value (the lighter the shade), the smoother the weight changes are in the local region. After training (i.e., (b), (c)), the U-matrix shows the emergence of clusters of smooth regions (light shade) separated by less smooth "walls" (dark shade). The overlaid phoneme labels are the same as in Figure 3.8, but omit the ones that are not strongly similar to a phoneme vector (i.e., inner product ≤ 0.9). ($\beta = 0$, $\theta = 20$)

right in Figure 3.10 (top). U-matrix analysis in Figure 3.10 (bottom) shows that some regions become smoother after training, although most of the other regions do not change significantly, possibly due to the large number of weights (2500).

3.3 Representing Continuous 2D Space

Experiments described above are limited in that they only consider a relatively small set of binary inputs, and that the same dataset was intentionally used for both training

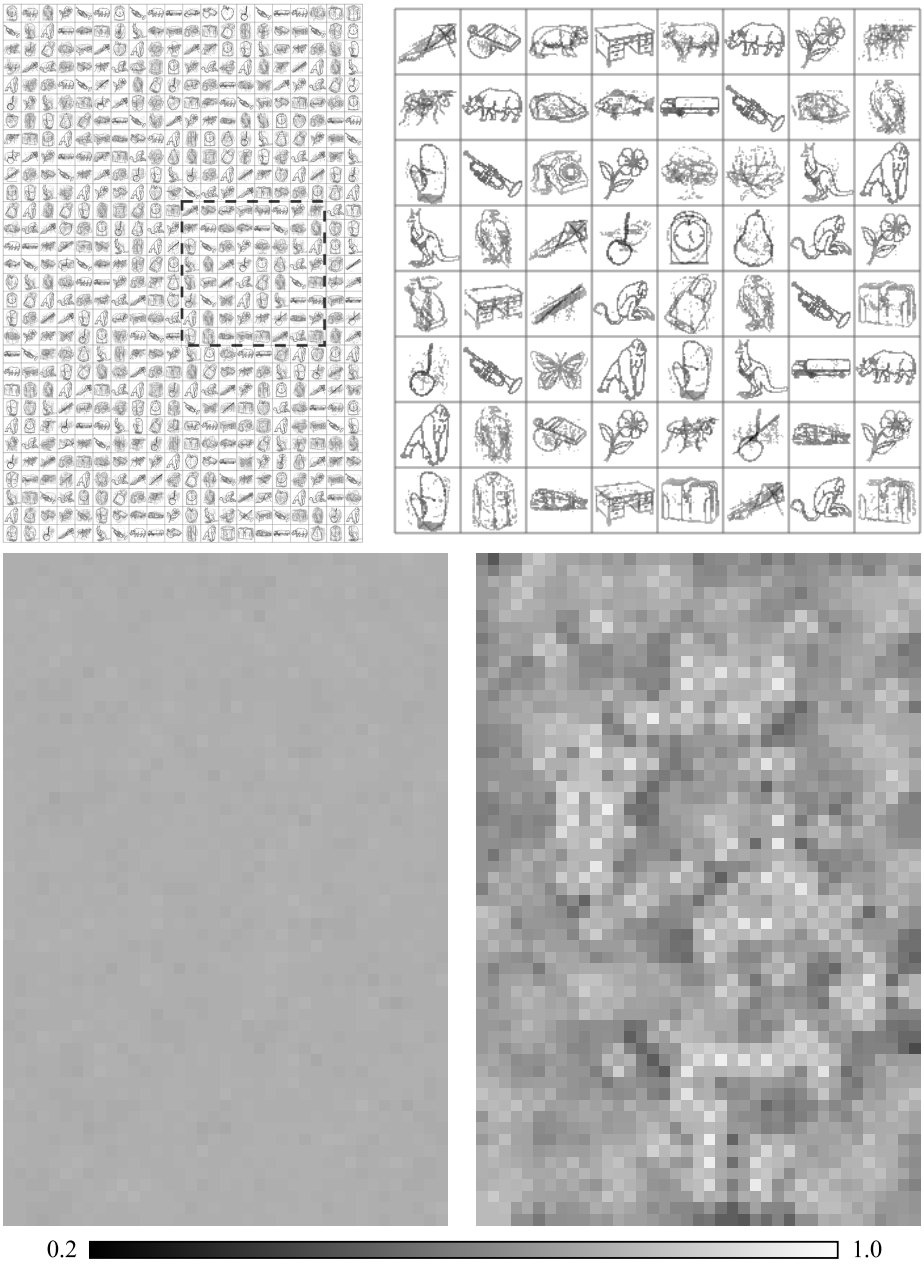

Figure 3.10: Top: Afferent weights of a 30×20 limit cycle SOM, trained using fixed 50×50 images as input ($\beta = 0, \tau = 20$). Afferent weights of each cell are visualized as a 50×50 image, where darker pixels in each cell indicate higher weight values. A threshold is used to filter out low weight values for clearer visualization. The left part shows an overview of the entire map. The region enclosed by the dashed lines is magnified and shown on the right. Cells showing clear images are strongly tuned to a single input image, while cells showing blurred or noisy patterns are the result of being tuned to multiple input images. Bottom: The U-matrices for the SOM before (left) and after (right) training. The U-matrix visualization method is identical to Figure 3.9.

SOMs and evaluating resulting limit cycles. The maps formed are best characterized as being feature maps. It therefore remains unclear whether limit cycle SOMs can also process real-valued data, as well as whether they can represent new inputs that they never see during training. The latter is known as the issue of generalizability, a fundamental issue for neural computation in general.

To address these issues, this section uses real-valued inputs that represent locations in continuous 2D space, which allows for generating a larger dataset as well as separate training and testing sets. It also examines the issue of whether limit cycle SOMs can produce topographic maps as well as the feature maps that were demonstrated above. This specific application (location in 2D space) is motivated in part by its obvious relevance to intelligent agents. Since it is unknown a priori, the research question being asked is whether short variable-length limit cycles can emerge in the state space of a SOM at all, as well as whether map formation will occur in the trained SOM, under these new conditions. If so, the goal is to evaluate the nature of the emerged limit cycles based on their uniqueness and their correlation to inputs. The latter refers here to the degree to which nearby locations are represented by similar limit cycles. This was not discussed in the last section partly because the distances between binary features are less continuous than real-valued inputs used here. Using real-valued spatial locations as inputs allows looking closely at how locations are organized in their representation space, which is composed of spatiotemporal limit cycle states. Importantly, such correlation to inputs can serve as a good indicator for assessing generalization ability: If a SOM generalizes well, this correlation between input and representation similarities should hold for new spatial location inputs that the SOM

has never been trained on.

3.3.1 Methods

For the specific spatial location application that is considered here, each input represents the 2D coordinates (x, y) of a point within a 1×1 square, where $0 \leq x < 1, 0 \leq y < 1$. In practice, as in Schulz and Reggia (2004), each point (x, y) is projected to a unit sphere in order to obtain normalized vectors. The actual input vector becomes (a, b, c) where

$$a = \frac{x}{\sqrt{2}}, \tag{3.16}$$

$$b = \frac{y}{\sqrt{2}}, \tag{3.17}$$

$$c = \frac{\sqrt{2 - x^2 - y^2}}{\sqrt{2}}. \tag{3.18}$$

The training data set is composed of 300 random points uniformly sampled within the 1×1 area. On the other hand, the evaluation, or testing, data set is composed of 100 grid points located at $x, y \in \{0, 0.1, 0.2, ..., 0.9\}$. Since training data is sampled randomly in a continuous space, it is very unlikely that the training data overlap with test data. Therefore, evaluating the SOM based on such test data reflects generalizability.

A SOM containing 40×30 nodes with $\beta = 0$ (same-node feedback connection weights) is simulated. The value of β is determined from the results in Section 3.2. A total of 20 independent simulations are performed with different initial random weights.

The SOM is trained for 1000 epochs using the training data set. In each epoch, each vector in the training data set is presented to the SOM for 5 time steps (starting from $t = 0$), after which the SOM continues to update its activation states and adapt for another $\tau = 5$ time steps (continuation time determined empirically) without external input, i.e., with afferent activity $\boldsymbol{x}(t) = \mathbf{0}$. The activation and learning rules are applied accordingly throughout the 10 time steps for each input, although during the last 5 time steps, only recurrent weights are updated while the afferent weights are unchanged since the afferent input is all zeros (see Equation 3.7).

After training, all weights are fixed, and the peak activity decay parameter is set to $\gamma = 0$, i.e., all winners have activation level 1 and non-winners 0. Each vector in the evaluation data set is presented to the SOM for 5 time steps, and the activity attractor naturally occurring afterwards (from $t = 5$ on) in the SOM is designated to be the dynamic representation that encodes the input coordinate vector. For brevity, the representation encoding a coordinate input (x, y) is denoted as $R_{(x,y)}$, which is an ordered list of spatial patterns. The encoding attractors $R_{(x,y)}$ can be potentially qualitatively classified into three types: fixed point, limit cycle, and complex (see Section 3.2).

3.3.2 Limit Cycle Formation

Before training, the evaluation data set yields complex attractors and long limit cycles. After training, small limit cycles are detected for all input vectors in the evaluation data set. A majority of the limit cycles are of length 2, while in a small number of

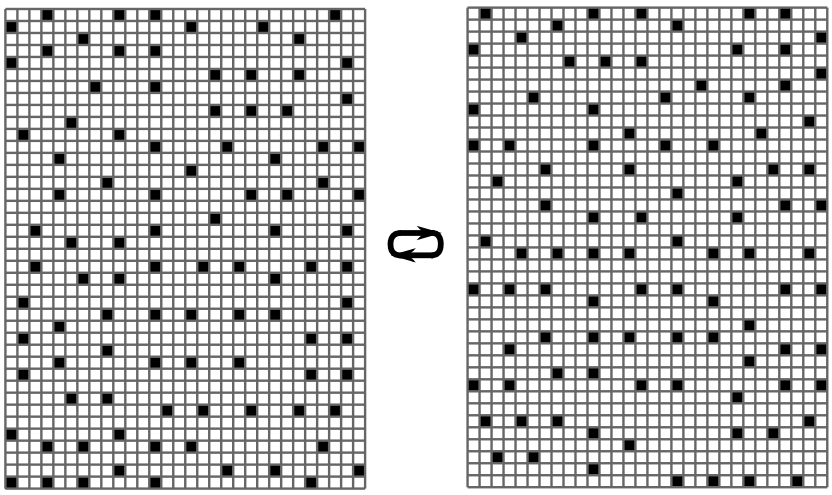

Figure 3.11: A typical activity limit cycle $R_{(0.1,0.1)}$ encoding input coordinates (0.1, 0.1). This limit cycle contains 2 alternating states, each of which is a sparsely coded activation pattern. Each cell represents a node in the SOM. Dark cells represent activated nodes; light cells are inactive.

cases those of length 4 also occur, resulting in an average length of 2.016 ($SD = 0.024$). On average, a limit cycle occurs at 3.7 time steps ($SD = 0.2$) after each input stimulus is removed ($t = 5$). A typical activity limit cycle of length 2 representing coordinates $(x, y) = (0.1, 0.1)$, i.e., $R_{(0.1,0.1)}$, is depicted in Figure 3.11.

3.3.3 Limit Cycles Properties

Figure 3.12 shows the differences between $R_{(0.1,0.1)}$ and $R_{(0.2,0.2)}$ ($R_{(0.9,0.9)}$). It can be observed that similar inputs, (0.1, 0.1) and (0.2, 0.2), yield similar limit cycles, where only a few nodes at the bottom-left corner have different activations (Figure 3.12a). On the other hand, distant inputs, (0.1, 0.1) and (0.9, 0.9), yield quite different limit cycles (Figure 3.12b).

To formally assess the similarity of limit cycles, the distance between two arbitrary limit cycles is measured to be the minimum one-norm distance between their constituent

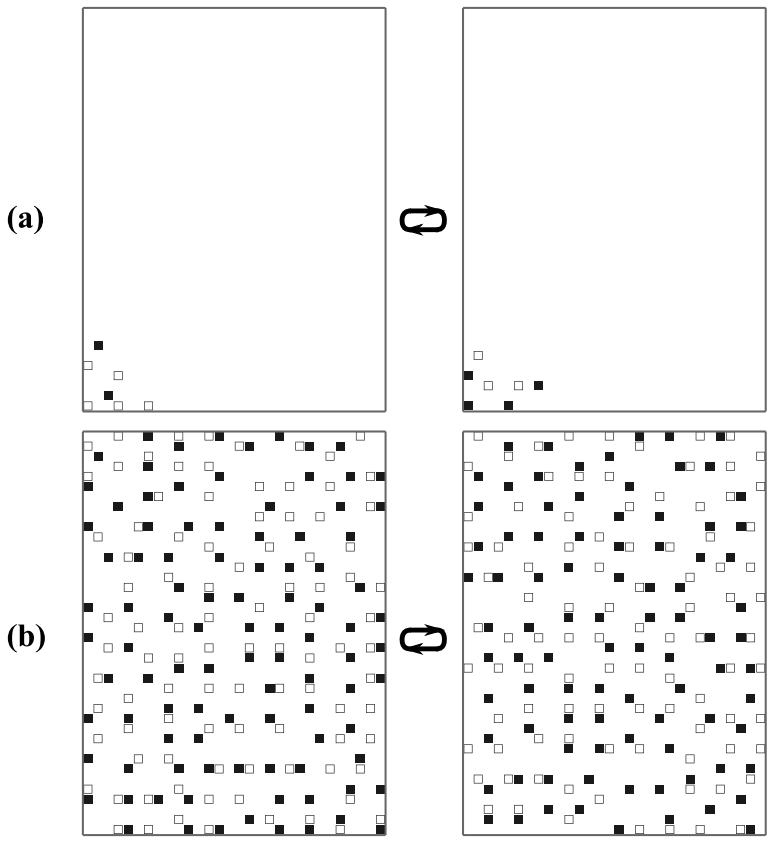

Figure 3.12: Comparisons of limit cycles (a) $R_{(0.2,0.2)}$, which is the representation of an external location close to (0.1, 0.1), and (b) $R_{(0.9,0.9)}$, which is for an external location far from (0.1, 0.1), with $R_{(0.1,0.1)}$ (the latter was shown in Figure 3.11). Only the differences of corresponding activation patterns are shown. Filled squares represent nodes that are activated in $R_{(0.2,0.2)}$ ($R_{(0.9,0.9)}$) but not in $R_{(0.1,0.1)}$; hollow squares represent nodes that are activated in $R_{(0.1,0.1)}$ but not in $R_{(0.2,0.2)}$ ($R_{(0.9,0.9)}$). The two states in $R_{(0.2,0.2)}$ and $R_{(0.9,0.9)}$ are aligned with those in $R_{(0.1,0.1)}$ such that the differences are minimized.

states (Equation 3.15). This distance reflects the least number of nodes whose activation must be inverted (i.e., changing 0 to 1 or vice versa) to cause one limit cycle attractor to be converted into the other. Figure 3.13 shows the distances between the limit cycles corresponding to all points in the evaluation data set and the limit cycles corresponding to the two selected points, (0.1, 0.1) and (0.5, 0.9). In both cases, the distances between limit cycles correspond quite well with the distances in the

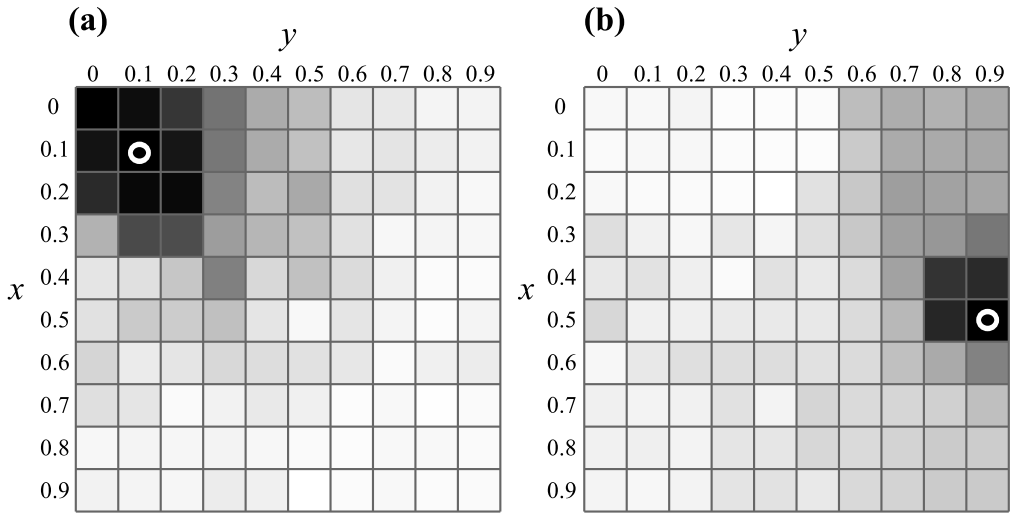

Figure 3.13: The distances between the limit cycle for each point in the evaluation data set and the limit cycle for (a) the point (0.1, 0.1) and (b) the point (0.5, 0.9). Each cell represents a point in the evaluation data set. Lighter colors represent greater distances away from the selected points, (0.1, 0.1) or (0.5, 0.9). The two points, (0.1, 0.1) in (a) and (0.5, 0.9) in (b), are highlighted using white circles.

input locations. A gradient, although not perfect, is formed such that as input moves away from the selected points, the corresponding limit cycle gradually becomes more different (indicated by gradually lighter shades of the cell).

To further verify this phenomenon, the distance correlations (Székely et al., 2007) are examined, that is, the correlations between (1) the distance between any pair of input coordinates, measured using Euclidean distance, and (2) the distance between their corresponding limit cycles. Additionally, the average distance between any pair of limit cycles is also calculated. The average distance reflects the uniqueness of each limit cycle representation. A generally desirable property for encoding a set of items is the resulting representations being as unique as possible. The more unique a representation is, the less likely its corresponding item is to be confused with other items in subsequent processing, such as in a classification task. The distance metric

Table 3.4: Distance correlation and average distance of limit cycles before and after training.

	Distance correlation	Average Distance
Before training	0.47 ($SD = 0.02$)	64.75 ($SD = 3.87$)
After training	0.89 ($SD = 0.01$)	156.29 ($SD = 1.55$)

in Equation 3.15 is used here.

Table 3.4 summarizes the results concerning distance correlation and average distance between the limit cycles. After training, both metrics are significantly increased. For distance correlation, the value obtained after training indicates that distances in the input space are highly correlated with distances in the limit cycle state space (a value of 1 indicates two variables being almost surely dependent, while 0 indicates statistical independence). In other words, there is a strong tendency that similar inputs will result in similar limit cycles, and distinct inputs will results in distinct limit cycles. At the same time, the distance between any pair of limit cycles increases after training. This indicates that each limit cycle becomes more unique. These two properties together demonstrate that the limit cycles encode coordinate inputs quite well.

3.3.4 Map Formation

The weight values are visualized in Figure 3.14. Map formation is obviously observed after training (right column), where a gradient is formed between clusters of bright cells (high weight values) and dark cells (low weight values). The existence of multiple bright (dark) clusters is caused by multi-winners-take-all activation, similar to the

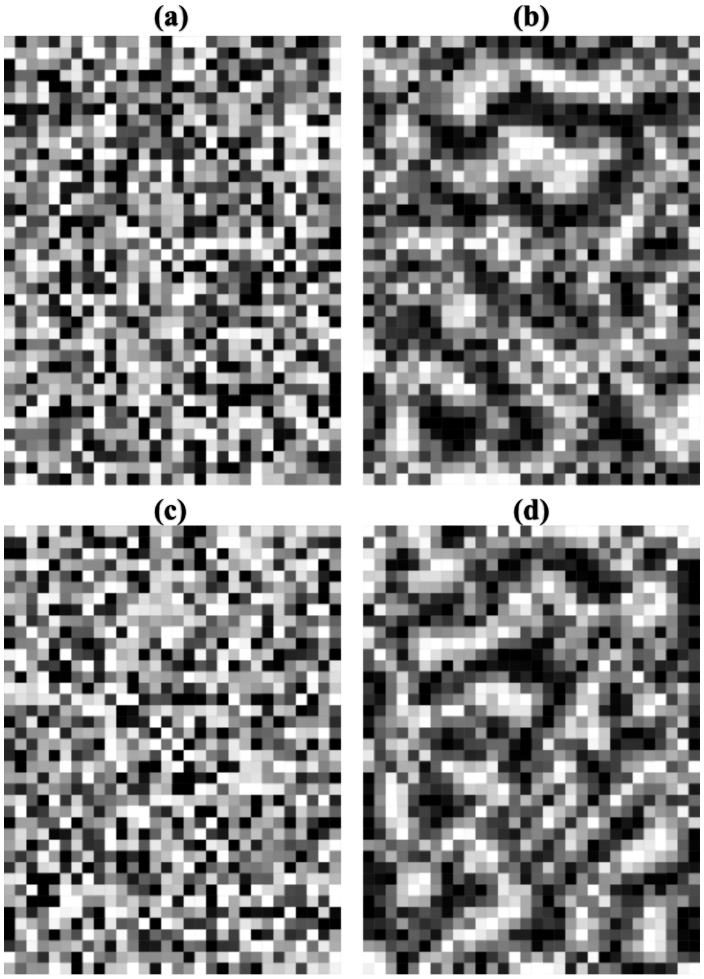

Figure 3.14: Map formation (a, c) before and (b, d) after training. (a) and (b) show the weight values corresponding to the x component of input coordinates; (c) and (d) shows the weight values corresponding to the y component. Each cell corresponds to a node in the SOM. Brighter cells indicate higher weight values.

observations in Section 3.2.7. While being different from using conventional single-winner SOMs, which are likely to form a single cluster of bright/dark cells, this map formation is reminiscent of biological cortical maps that form quasi-repetitive patterns, such as those observed in cat visual cortex (Swindale et al., 2000, Figure 1).

3.4 Summary and Discussion

The work described in this chapter determined whether and how short variable-length limit cycle attractors can emerge in activity states of multi-winner SOMs with locally recurrent feedback connections. Limit cycles were learned through self-organization using training continuation, both when they were used to encode binary sequence inputs and to encode real-valued static inputs, while using the same SOM model and learning rules. The learned limit cycles appear to be quite abstract, and their lengths do not appear to be related to the lengths of their corresponding input sequences. The oscillatory nature of individual map nodes that occurred due to the limit cycles are suggestive of biological cortical activity that is rhythmic.

An advantage of using limit cycles as SOM representations is that they are relatively stable. This means that they can resist certain amounts of noise in SOM activity, while static representation as used in conventional SOMs cannot. It was also shown that limit cycle representations became more unique after self-organization than static representations as well as fixed-point and complex attractors. This makes it easier to distinguish one limit cycle representation from another, which is a generally desirable property for encoding input stimuli using neural activity. Based on stability and uniqueness metrics, it was found that appropriate short limit cycle representations emerged consistently when $\beta = 0$, a situation where each map node does not connect recurrently to itself.

It was also shown that limit cycle activity in a SOM generalizes to new inputs that had never been used to train the SOM. This was done when the inputs are real-valued,

making the input space more continuous. In this case, the similarity of new limit cycles encoding new inputs was found to be well correlated to the similarity of the new inputs. This means that similar (distinct) inputs are encoded by similar (distinct) limit cycles. In other words, this showed that the SOM generalizes in that the input space can be nicely mapped via self-organization to limit cycle representation space.

Finally, map formation was found to be present robustly, regardless of whether the inputs were binary or real-valued and whether they were static patterns or temporal sequences of patterns. Map formation also occurred over a range of SOM parameter values. In contrast to conventional SOMs, where single clusters of high and low weights are usually formed, in limit cycle SOMs multiple clusters are formed, and they appear to be in the shape of multiple stripes. This is likely due to the use of multi-winner SOMs. Although a multi-winner map is not as smooth as a single-winner one, i.e., weight changes are greater between neighboring map nodes, it offers the potential for a much greater coding capacity, and the map is still much smoother after than before training.

— Chapter 4 —

Learning Associations Between Limit Cycles in Multi-SOM Architectures

For SOMs to be useful in neural architectures involving multiple cortical regions, these SOMs need to be able to interact with each other and exchange useful information in the form of hetero-associations. For example, while they do not involve limit cycles, neuroanatomically-based models of language and symbol grounding need to associate spoken names with seen objects (Monner and Reggia, 2012; Weems and Reggia, 2006). It is not obvious in advance that this can be achieved effectively in architectures of SOMs using limit cycle representations. This chapter investigates possible ways of applying limit cycle representations in two interacting SOMs where each SOM processes a distinct modality, e.g., auditory, visual, or proprioceptive stimuli. The goal here is to study the practicality of limit cycles to form associations. At a minimum, a represented entity in one SOM (e.g., a limit cycle representing the spoken name of a specific object) should be retrievable not only by repeating the original stimuli it represents, but also by an associated representation appearing alone in another SOM (e.g., a static or limit cycle representation encoding the image of the named object).

Despite the good properties demonstrated by isolated limit cycle representations

in Chapter 3, it is unclear in advance whether associations between representations in different SOMs can be learned when using limit cycles. With conventional single-winner SOMs, the associations can be learned simply as a one-to-one mappings between the two winners of the two SOMs. Simple Hebbian learning rules appear to be sufficient in this case. On the other hand, establishing representation associations between two multi-winner SOMs is a much harder problem, especially if one or both SOMs exhibit limit cycle dynamics. The reason is that such associations involve multiple activity patterns per limit cycle, and that each activity pattern contains multiple winning nodes. Each winning node in a limit cycle may also participate in many other limit cycles. Further, the limit cycles to be associated may be of different lengths, and explicit temporal alignment or detection of limit cycles should be avoided to reduce control overhead. On top of all these difficulties, it is unknown whether it is possible at all for the learned associations to generalize.

In the context of retrieving limit cycles in one SOM using corresponding representations in another SOM, the following describes three experiments where two SOMs learn to associate their respective activity. First, a baseline is established by learning to retrieve limit cycle representations using static representations. This shows that SOMs using a dynamic representation method are "backward compatible" with those using existing static representations. The application here is associating visual images with spoken names of the 50 objects used in Section 3.2. In the second experiment, both SOMs use limit cycle representations for the same application. The goal is specifically to learn to retrieve a limit cycle in a SOM representing phonemes, using a corresponding limit cycle in the other SOM representing images. Finally, in the

last experiment, generalizability is emphasized in the context of a robotic application: open-loop arm reaching. The goal is to retrieve proper joint angles for the arm to reach a given 3D spatial location. This involves representing and associating two continuous 3D spaces, joint angles and spatial coordinates, using limit cycles.

4.1 Retrieving Limit Cycles Using Static Activity

In this first experiment, the goal is to study whether a limit cycle can be retrieved at all without its afferent inputs. To establish a baseline, limit cycles are to be retrieved using static representations. The results will indicate if limit cycle SOMs can work when combined with conventional SOMs in the same neural architecture. Further, this allows comparisons between retrieving limit cycles and retrieving static representations, the latter being a control experiment. The application here is for a SOM to learn to recall the limit cycles that represent learned phoneme sequences, without being provided with the input sequences directly, but instead based on activation of another SOM which has previously learned independently to process stimuli in a different modality (visual images). This is a kind of "name that object" task, and it relates to the difficult issue of symbol grounding (Monner and Reggia, 2012).

4.1.1 Methods

Figure 4.1 shows the architecture used in this experiment. It consists of two 30×20 SOMs, each processing stimuli in a different modality, and each trained separately before being connected together. Associative connections exist between the two SOMs,

via a hidden layer of 20 nodes. For simplicity, the image SOM in this experiment has no recurrent connections, and thus its activation forms a static representation, driven directly by the image input. The question being asked is whether a static representation in one map can be associated with limit cycle representations in another map (or more generally, can limit cycle representation SOMs be combined effectively with more conventional static representation SOMs in a single system). The limit cycles that are to be recalled in this case are in the phoneme SOM, which has three channels. Channels 1 and 2 are phoneme input and topographic feedback, respectively, similar to the SOM in the previous experiment ($\alpha_1 = 0.64$, $\alpha_2 = 0.36$; see Equation 3.3). Channel 3 is composed of associative connections from the image SOM via the hidden layer (20 nodes). The goal of the task is, after learning, to recall the limit cycle representing each phoneme sequence P_k in the phoneme SOM (i.e., an object's name), when the architecture is provided with only an image stimulus V_k, where $\langle P_k, V_k \rangle$ describes the same object. The procedure in this experiment is outlined in Figure 4.2 and elaborated below.

Training is performed in two stages. In the first stage, the two SOMs are trained independently using the data in their respective modality. Channel 3 of the phoneme SOM is disabled ($\alpha_3 = 0$) during this time. The parameters for training the phoneme SOM are $\beta = 0, \tau = 20$. The image SOM does not have recurrent feedback connections, and thus it is trained in the same way as conventional SOMs (although multi-winners-take-all is used). At the end of the first stage, the static representations in the image SOM are recorded as a set of activation states $\{R_1^V, R_2^V, ...\}$, where V is for "visual", and the limit cycle representations in the phoneme SOM are recorded as $\{R_1^P, R_2^P, ...\}$,

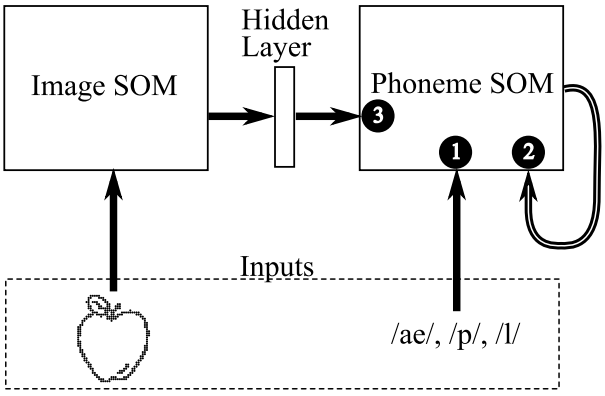

Figure 4.1: The architecture for the task of establishing associations between static and limit cycle representations. The two SOMs, one for images and the other for phoneme sequences, are connected via a hidden layer. The three channels of the phoneme SOM are numbered by circles that are referred to in the text. Full connections are drawn as solid arrows, and topographic connections are drawn as hollow arrows.

where P is for "phonemes". Although recording R_k^P requires detecting limit cycles, they are used for assessing performance in the testing phase only. Additionally, for each phoneme input sequence P_k, the single state in the phoneme SOM at $t = |P_k| + \xi$, where ξ is a discrete uniform random variable in $[0, 9]$, is recorded as S_k^P. This simply samples a random post-stimuli activity pattern, and it does not require detecting the onset/length of a limit cycle. In the second stage of training, each pair $\langle R_k^V, S_k^P \rangle$ is used as the input and the target output, respectively, to train the associative connections between the two SOMs (channel 3). Training is based on a standard error-backpropagation method, RPROP (Riedmiller and Braun, 1993). Error backpropagation is not biologically plausible, but its equivalence to a more biologically plausible contrastive Hebbian learning method has been established (Xie and Seung, 2003). Here I use error backpropagation mainly for its simplicity and readiness to implement the association learning methods.

Training Phase 1:

Disable the associative connections between the two SOMs
Train both SOMs independently using unsupervised learning
Temporarily store activation states for later use:
    foreach k in the dataset:
        $R_k^V \leftarrow$ the static state in the image SOM for input V_k
        $R_k^P \leftarrow$ the limit cycle of states in the phoneme SOM for input P_k
        $S_k^P \leftarrow$ a single state that occurs at $t = |P_k| + \texttt{random\_integer}\{0, ..., 9\}$ in the phoneme SOM for input P_k

Training Phase 2:

Train the associative connections between the two SOMs using error backpropagation with training data $\langle R_k^V, S_k^P \rangle$, $\forall k$

Testing Phase:

Enable the associative connections between the two SOMs
Disable the afferent input of the phoneme SOM
Test if the limit cycles in the phoneme SOM are restored by image inputs alone:
    foreach image V_k in the dataset:
        Provide V_k as input to the image SOM for the first t^{in} time steps
        Update activation states throughout the architecture for each time step until a limit cycle occurs in the phoneme SOM
        if the above limit cycle == R_k^P:
            Mark data entry k as "successful"

Figure 4.2: Experimental procedure for associating static representations with limit cycles.

In a separate control experiment, a model identical to that shown in Figure 4.1 is used, except that the phoneme SOM is based on static representations. The two training stages described above are performed on this model as well, except for the following differences. In the first stage, the phoneme SOM is trained using $\beta = 0, \tau = 0$. Each static representation $R_k^{P,\text{static}}$ in the phoneme SOM is taken to be the final activation pattern occurring at $t = |P_k| - 1$. In the second stage, each $R_k^{P,\text{static}}$, as opposed to a randomly selected state, is used as the target output for learning the associative connections.

Notice that with each S_k^P for training, the association is recorded at a random time step after the input phoneme sequence ends. The timing of S_k^P does not depend on the timing of the cycle R_k^P or its length. In fact, S_k^P may not even be a state in the cycle R_k^P. The intent here is to determine whether associative learning can be effective with this relaxation of timing requirements, and whether it can restore an entire limit cycle based on an association with part of it. In contrast, recording each static representation $R_k^{P,\text{static}}$ as has been done in some previous SOM models (Schulz and Reggia, 2004) for training the associations requires the knowledge of the exact time when the target activation state appears, i.e., at $t = |P_k| - 1$, something that in general would differ with different length input sequences.

After training, the architecture is tested by enabling channel 3 (associative connections; $\alpha_3 = 0.64$) and disabling channel 1 (afferent connections; $\alpha_1 = 0$) for the phoneme SOM. Consequently, the architecture receives only image stimuli via the image SOM. Each image V_k is presented for t^{in} (a constant parameter) time steps, during which the learned static pattern R_k^V occurs in the image SOM. The activation

of the phoneme SOM is triggered by R_k^V via the hidden layer during this time. To assess robustness of association formation, noise of amplitude ν (a constant parameter) is added in channel 3, in the same way as described in Equation 3.14. The added noise persists throughout the t^{in} time steps during which the image input is presented. If the eventual state trajectory of the phoneme SOM recreates the corresponding cycle R_k^P, the data entry k is considered successfully recalled. For the control experiment, a recall is considered successful if $\boldsymbol{a}(t^{in}-1) = R_k^{P,\text{static}}$ in the phoneme SOM.

4.1.2 Results

Figure 4.3 shows the percentages of all representations in the phoneme SOM that are recalled this way, with respect to different parameter values for β (same-node feedback weight), t^{in} (number of time steps the image input is presented), and ν (amplitude of noise added to channel 3 of the phoneme SOM). The results reported here are averaged over 20 independent simulations with different random initial weights. The highest recall rate is achieved with β values around 0, which is more evident when noise is added ($\nu > 0$). This echoes the observations in Section 3.2.3 that small cycles around $\beta = 0$ are robust.

When compared with using static representations in the phoneme SOM, using limit cycle representations consistently results in higher recall rates (Figure 4.3b, c). The limit cycle representations also require the image input to last for a shorter time period ($t^{in} = 2$) in order to reach the maximum recall rate. Notice that the static representations have very low recall rate for $t^{in} = 1$. This is the case even when

error-backpropagation training produces very low mean-square errors. The reason is that the multi-winners-take-all process can result in incorrect winners even when these nodes receive a very small amount of net input from channel 3, if their net inputs are the highest in their local neighborhoods. Allowing the phoneme SOM activation to update for multiple time steps ($t^{in} > 1$) via both the associative and the recurrent feedback connections helps eliminate some of these incorrect winners for both static and limit cycle representations, but only to a certain extent (the recall rates are somewhat the same for $t^{in} \geq 5$ in the case of static representations). For limit cycle representations, attractor dynamics provide an additional means to restore incorrect activation, which results in higher recall rate than static representations.

As noise amplitude ν increases, the recall rates for both the static and the limit cycle representations decline, although the limit cycle representations consistently maintain higher recall rate than the static ones (Figure 4.3). The *difference* in performance actually monotonically increases as ν increases until noise becomes quite substantial ($\nu > 0.3$). The limit cycles perform slightly better with a small amount of noise, which on closer examination, helps prevent incorrect winners in the attractor dynamics, and thus they are robust in the presence of noise.

In summary, this experiment shows that it is possible to recall limit cycle representations (in the phoneme SOM) using static activation patterns (in the image SOM), via associative connections. This suggests that SOMs based on limit cycle representations are compatible with and might be combined with the more prevalent SOMs using static representations. The training data for limit cycle associations are sampled at relatively arbitrary times. This shows one possible way in which timing requirements

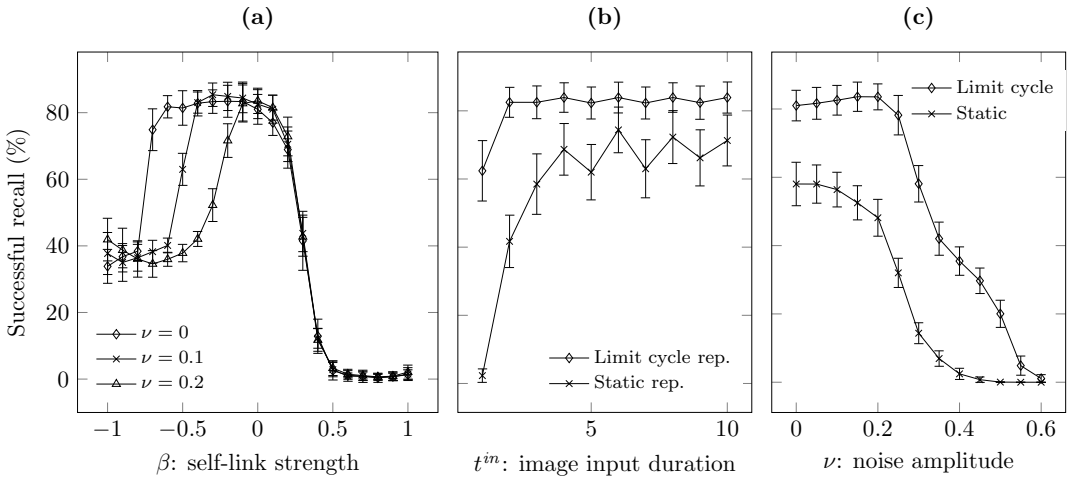

Figure 4.3: The percentages of the dynamic and the static representations recalled in the phoneme SOM of Figure 4.1 when image stimuli alone are provided to the architecture. The results in (a) and (c) are averaged over $t^{in} \in [1, 10]$. $\beta = 0$ in (b) and (c); $\nu = 0$ in (a) and (b); $\tau = 20$ in all cases. The error bars indicate one standard deviation over 20 independent simulations.

for limit cycle representations are fairly relaxed. Recalling limit cycle representations is consistently more successful than static representations. They require referencing the source of association for fewer time steps and are more resistant to noise.

4.2 Retrieving Limit Cycles Using Limit Cycles: A Dual-Route Approach

A neural architecture operating solely based on limit cycle representations must be able to establish associations between these attractors in different SOMs. This is a more difficult task than that considered in the last experiment, because the limit cycles in the different SOMs each contains multiple spatial patterns, and the limit cycles can be of different lengths and aligned temporally in different ways. This experiment aims to associate pairs of limit cycle representations, such that a limit cycle in one SOM

can be recalled by referencing only the associated limit cycle in another SOM. Unlike in the previous experiment, where a limit cycle is recalled by attempting to recreate one of its preceding or constituent states, here multiple states in the cycle are learned by the association. To demonstrate the relaxation of operation timing requirements, limit cycles in different SOMs are not explicitly aligned during and after training, as doing so requires explicitly detecting limit cycles.

4.2.1 Methods

Figure 4.4 shows that the architecture used in this experiment is similar to that in the previous experiment. Again, the goal after learning is to recall each limit cycle representing a phoneme sequence (name) in the phoneme SOM by giving the architecture only the corresponding image. However, the image SOM now also has topographic feedback connections in addition to afferent connections, and thus is able to generate limit cycle representations for image stimuli. Further, the associative connections between the two SOMs now contain two routes. Route I consists of direct topographic connections between the two SOMs (radius 2). Route II consists of indirect full connections via a hidden layer (60 nodes), which were the only associative connections in the previous experiment. This is partially inspired by the dual-route theory of language processing (Coltheart et al., 2001; Weems and Reggia, 2006), and due to empirical results from a set of preliminary pilot experiments indicating that having both routes works better than having each alone. These two routes are trained separately and tested jointly. The procedure of this experiment is outlined in

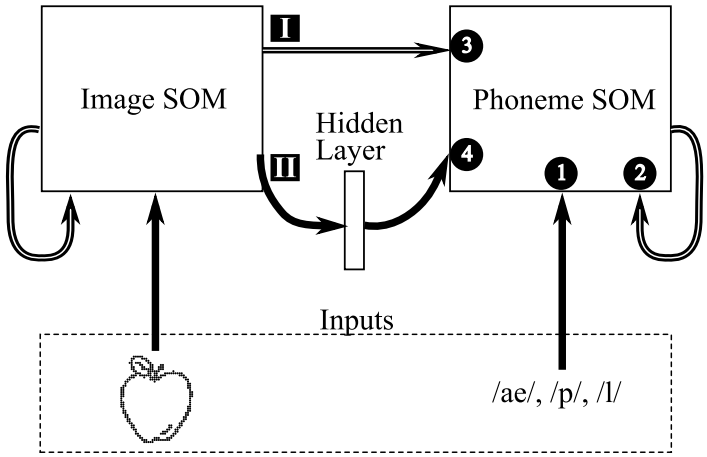

Figure 4.4: The architecture for the task of establishing associations between limit cycle representations occurring in two different SOMs. Both SOMs have direct feedback. The four channels of the phoneme SOM are numbered in circles. Full connections are drawn as solid arrows, and topographic connections are drawn as hollow arrows. The two SOMs are connected using associative connections containing two routes. Route I is topographic and involves direct connections. Route II is full and involves indirect connections through a hidden layer.

Figure 4.5 and elaborated below.

Training of this architecture is again divided into two stages. In the first stage, training of the two SOMs is performed independently (channels 3 and 4 of the phoneme SOM are disabled) to acquire limit cycle representations in their respective modalities. Small limit cycle attractors are learned independently in both SOMs. Each image stimulus V_k is represented by a limit cycle R_k^V, and each phoneme sequence P_k is represented by a limit cycle R_k^P, where the lengths of R_k^V and R_k^P are in general different.

In the second stage of training, each tuple of data $\langle V_k, P_k \rangle$ is presented to the architecture. The two SOMs are normally activated through time while $\alpha_3 = \alpha 4 = 0$ (i.e., activation in the image SOM does not affect that of the phoneme SOM). Both routes of the associative connections are then adapted within an *adaptation period*.

Training Phase 1:

Disable the associative connections between the two SOMs
Train both SOMs independently using unsupervised learning
$AdaptationPeriod \leftarrow t = 30, 31, ..., 34$ (exact timing not important; see text)
Temporarily store activation states for later use:
    foreach k in the dataset:
        $A_k^V \leftarrow$ states that occurs in *AdaptationPeriod* in the image SOM for input V_k
        $A_k^P \leftarrow$ states that occurs in *AdaptationPeriod* in the phoneme SOM for input P_k
        $R_k^P \leftarrow$ the limit cycle of states in the phoneme SOM for input P_k

Training Phase 2:

Train route (I) of the associative connections using Equation 4.1 and $\langle A_k^V, A_k^P \rangle$, $\forall k$
Train route (II) of the associative connections using error backprop and $\langle A_k^V, A_k^P \rangle$, $\forall k$

Testing Phase:

Disable the afferent input of the phoneme SOM
Test if the limit cycles in the phoneme SOM are restored by image inputs alone:
    foreach image V_k in the dataset:
        Provide V_k as input to the image SOM
        Disable both routes of the associative connections at $t = 0$
        Enable both routes of the associative connections at $t = t^{assoc}$ (parameter)
        Update activation states throughout the architecture for each time step until a limit cycle occurs in the phoneme SOM
        if the above limit cycle == R_k^P:
            Mark data entry k as "successful"

Figure 4.5: Experimental procedure for associating two sets of limit cycles.

The adaptation period is chosen rather arbitrarily, beginning at the 30th time step after each input and lasting for 5 time steps. The number 30 can instead be any number such that adaptation occurs sufficiently late, i.e., after the dynamics of both SOMs have settled into the limit cycle attractors R_k^V and R_k^P. The duration of 5 time steps can instead be any small number. Large numbers tend to prematurely over-fit the associative weights for a particular pair of limit cycles. The same duration of the adaptation period is used regardless of the different lengths of the limit cycles. The number 5 is chosen here to demonstrate that the length of the adaptation period does not have to be the same or a multiple of the lengths of any limit cycles (no limit cycles

of length 5 are known to form across all simulations; see Figure 3.4). This signifies that the timing and the lengths of the limit cycles are not important in training the associative connections.

For the direct route (I), weight adaptation for channel 3 is performed during the adaptation period using modified Equation 3.7 (for each node i, channel $j = 3$):

$$\hat{\boldsymbol{w}}_{ij}(t+1) = \boldsymbol{w}_{ij}(t) + \mu_j \left[a_i(t) - \boldsymbol{w}_{ij}(t) \cdot \boldsymbol{x}_j(t)\right] \boldsymbol{x}_j(t), \tag{4.1}$$

and Equation 3.8 (unchanged). The quantity in the square brackets here represents the difference between activation level a_i of node i in the phoneme SOM that is triggered by an input phoneme sequence, and $\boldsymbol{w}_{ij} \cdot \boldsymbol{x}_j$ that represents the net input from channel 3. Since $\alpha_3 = 0$, $\boldsymbol{w}_{ij} \cdot \boldsymbol{x}_j$ $(j = 3)$ does not contribute to a_i but adapts passively. For the indirect route (II), weight adaptation is again performed using RPROP, using the input-driven limit cycles in the phoneme SOM as target outputs. The training data are recorded within the adaptation period for each data entry $\langle V_k, P_k \rangle$. As a result, they contain multiple pairs (5 in this case, which is the length of the adaptation period) of input and target output states for each data entry k, as opposed to only one pair each in the previous experiment.

After training, only image input is provided to the architecture ($\alpha_1 = 0$ for the phoneme SOM). Each R_k^V is recalled in the image SOM as expected. The phoneme SOM needs to recreate the corresponding limit cycle representation R_k^P by referencing the limit cycle representation R_k^V in the image SOM, via both routes of the associative connections. Channels 3 and 4 are initially disabled ($\alpha_3 = \alpha_4 = 0$) to prevent pre-limit

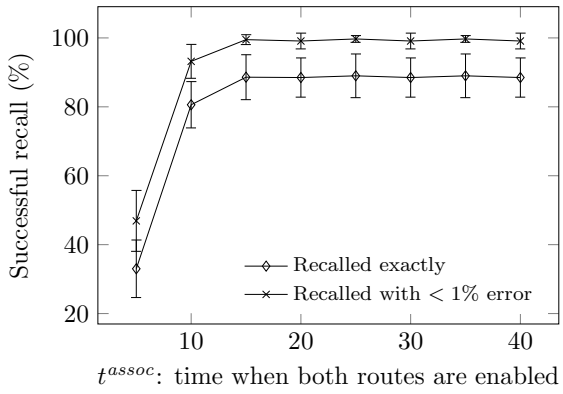

Figure 4.6: Percentages of limit cycle representations that are successfully recalled in the phoneme SOM and percentages that are nearly recalled (at most 5 nodes out of 600 having incorrect activation) when input is given only to the image SOM.

cycle states from passing through. They are then enabled at $t = t^{assoc}$ ($\alpha_3 = 0.1$, $\alpha_4 = 0.15$) and remain so, where t^{assoc} is a parameter. At this point the phoneme SOM starts to access the image SOM via both routes. Note that the information about the start time and the length of each cycle in the image SOM is not conveyed in any explicit way to the phoneme SOM. Only node activities are accessed.

4.2.2 Results

Percentages of limit cycle representations correctly recalled in the phoneme SOM are shown in Figure 4.6. The figure also shows percentages that include "closely recalled" cycles, which are within one-norm distance of 5 away from the correct cycles. This means that for each limit cycle to be considered closely recalled, there must be 5 or fewer nodes (out of 600) having different activation from the correct cycle R^P_k, which translates to less than a 1% difference. This threshold is also much smaller than the average distance ~ 70 between any limit cycles (see Figure 3.5).

As the results show, nearly all limit cycles are closely recalled, $\sim 89\%$ of which are exactly recalled, for $t^{assoc} \geq 20$. The exact time of t^{assoc} is not critical. It does not have to be set according to the timing information about the input sequence, the representations, or the training process, as long as it is large enough to allow the image SOM to settle into a limit cycle R_k^V. This robustness with respect to timing indicates that the model is able to access limit cycle representations at relatively arbitrary times. In summary, the results here indicate that it is possible to retrieve a limit cycle using another limit cycle in a different SOM, and the retrieval timing can be fixed quite arbitrarily without detect/referring to the onset/lengths of individual limit cycles.

4.3 Retrieving Limit Cycles Using Limit Cycles: Working with Continuous Spaces

Although experiments so far have shown that it is possible to associatively retrieve limit cycles using either static or limit cycle activity, they are limited in that they consider only binary inputs and the dataset is relatively small. How outputs can be generated based on limit cycles is not addressed either. More importantly, the ability to generalize to new and unseen data, a critical indicator of a successful neural architecture, remains unknown. This current experiment aims to address these issues in the context of an open-loop arm control task.

Arm control is an inverse kinematics problem, in which the goal is for the manipulator (hand, gripper, etc.) of an arm to be moved to a target spatial location. This problem is known to be ill-posed and non-linear, and has been and continues to be

studied intensively in robotics and neurosciences (Bullock et al., 1993; Colomé and Torras, 2015; Flash et al., 2013; Hutchinson et al., 1996). The arm in this context is characterized by multiple rigid segments connected by rotatable joints, which, when rotated, will change the location of the free end of the arm (i.e., the manipulator). In this experiment, an open-loop version of the problem is considered, meaning that the manipulator location during a reaching movement is not provided as feedback to the neural architecture. This is analogous to reaching for something with one's eyes closed. The goal thus becomes: to find a vector in the joint angle space (i.e., a set of rotation angles for all arm joints) that places the manipulator at a target spatial coordinates in Cartesian space. To this end, associations between limit cycles representing Cartesian space and those representing arm joint angles must be learned.

A major distinction of this task from that in previous experiments (image-phoneme association) is that both the joint angle and Cartesian spaces are continuous. This allows studying how limit cycle representations for real-valued inputs can be associated with each other, as well as observing how the learned associations generalize to new spatial locations. In other words, the architecture needs to be able to invoke a proper oscillatory activity sequence, and eventually a proper output, for a new input never seen during training. Further, this task requires behavioral outputs to be generated from limit cycle representations, so that one can assess performance and generalizability based on the outputs directly, rather than indirectly based on the degrees to which limit cycles are retrieved. Finally, this task allows demonstrating timing flexibility in a more practical situation, since an architecture based on dynamic neural activity can be much less robust if its operation relies on highly specific timing. More specifically,

the time at which the associative connections are activated and that at which outputs are generated need to be unrestrictive. They need to be independent of the exact timing of individual limit cycles (i.e., the onset and the length) as well, so that the architecture does not need to explicitly detect limit cycles.

4.3.1 Arm Model

To limit the complexity of the problem, a 3-degree-of-freedom (3-DOF) arm operating in a 3D Cartesian space is used, having two shoulder and one elbow joint freely adjustable. This arm is modeled in a virtual environment for training and testing the proposed neural architecture. Figure 4.7 shows a schematic of the arm. When given a set of joint angles $\mathbf{\Theta} = (\theta_1, \theta_2, \theta_3)$, the arm model determines the spatial location $\boldsymbol{X}^M = (x, y, z)$ of the manipulator using the Denavit-Hartenberg method (Hartenberg and Denavit, 1965, p.435) (superscript M stands for manipulator). The value range of each dimension in both (x, y, z) and $(\theta_1, \theta_2, \theta_3)$ is normalized to fit in the range $[0, 1]$, before being fed into the architecture.

4.3.2 Neural Architecture

An overview of the neurocognitive architecture is shown in Figure 4.8. This architecture contains two SOMs, the spatial map and the joint angle map, each taking input from spatial coordinates of the manipulator and joint proprioception of the arm, respectively. Connections between the two maps are used to associate the two corresponding modalities. The joint angle map also drives joint command output, through a neural

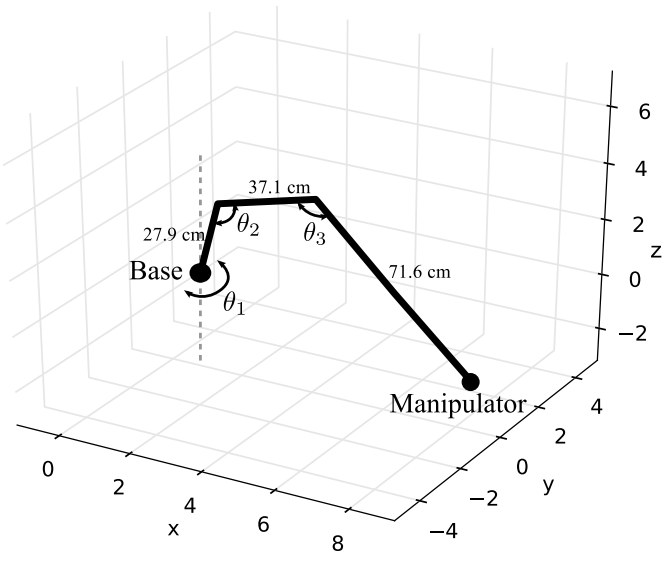

Figure 4.7: Schematic diagram of the arm model. The arm segment lengths shown are based on the actual size of a physical Baxter robot's arm.

network that provides temporal averaging of its activity. Finally, the arm model converts joint angles to manipulator coordinates to find spatial error, i.e., how far away the manipulator is from a target location.

The spatial map and the joint angle map are SOMs similar to those described in Section 3.1. The afferent connections connect their respective afferent inputs, i.e., (x, y, z) and $(\theta_1, \theta_2, \theta_3)$. The afferent input for the spatial map is either the manipulator location $\boldsymbol{X}^M$ during training, or a target location $\boldsymbol{X}^T$ during testing (superscript T stands for target). The recurrent connections in both maps exist between neighboring map nodes that are within a box distance of 2. The associative connections from the spatial map to the joint angle map (top of Figure 4.8) give the latter an additional set of incoming connections. The net inputs for each node i in the two maps at time t,

given afferent inputs $\boldsymbol{X}$ ($\boldsymbol{X}^M$ or $\boldsymbol{X}^T$) and $\boldsymbol{\Theta}$, is a modified version of Equation 3.3:

$$h_i^S(t) = -\alpha_{aff}^S(t)\left\|\boldsymbol{X} - \boldsymbol{w}_i^S\right\|^2 + \alpha_{rec}^S(t)\boldsymbol{a}^S(t-1)\cdot\boldsymbol{u}_i^S, \tag{4.2}$$

$$h_i^J(t) = -\alpha_{aff}^J(t)\left\|\boldsymbol{\Theta} - \boldsymbol{w}_i^J\right\|^2 + \alpha_{rec}^J(t)\boldsymbol{a}^J(t-1)\cdot\boldsymbol{u}_i^J + \alpha_{assoc}^J(t)f_{assoc}(\boldsymbol{a}^S(t)), \tag{4.3}$$

where superscripts S and J correspond to the spatial and the joint angle maps, respectively. Parameters α are relative strengths that gate inputs from different sources (namely, α_{aff} for afferent, α_{rec} for recurrent, and α_{assoc} for associative inputs), and can be different at different t. Trainable parameters $\boldsymbol{w}$ and $\boldsymbol{u}$ denote the afferent and recurrent connection weights, respectively. Note that the afferent input (first term) is now based on vector distances instead of inner product. The negative sign is for generating greater h_i for $\boldsymbol{w}_i$ that is closer to the input vector. The function f_{assoc} represents the fully-connected, two-layer feedforward net between the two maps. The activity patterns at each time step $\boldsymbol{a}^S(t)$ (spatial map) and $\boldsymbol{a}^J(t)$ (joint angle map) are determined using a multi-winners-take-all process as usual (Equations 3.5, 3.6).

In order to generate steady joint command output, the oscillatory activity in the joint angle map needs to be "smoothed out". For this purpose, a temporal average filter is added downstream of the joint angle map (Figure 4.8). This filter contains the same number of nodes as the joint angle map, and each node connects one-to-one to the nodes in the joint angle map. The activity of each node in the filter is a temporally moving average of the corresponding map node's activity in the last t^{filter} (a parameter)

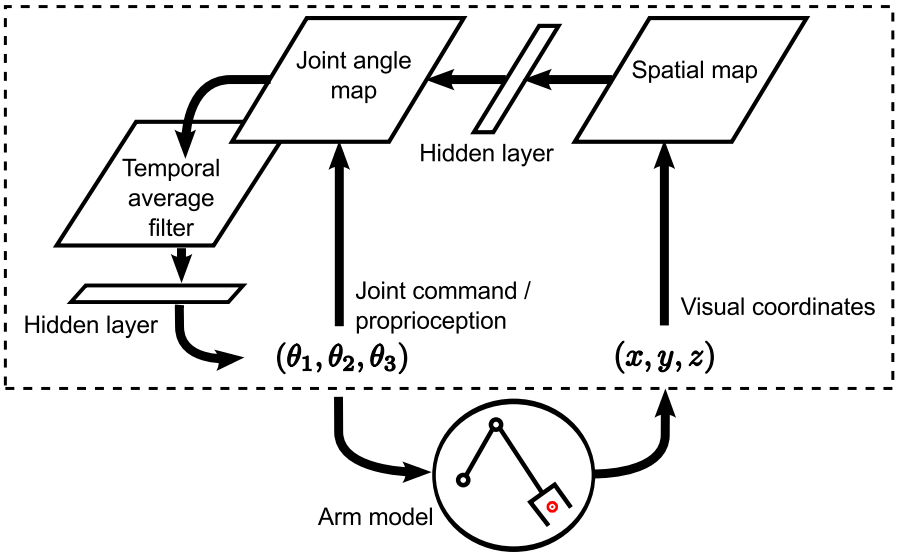

Figure 4.8: An overview of the neural architecture for open-loop arm control. The architecture contains two SOMs, the spatial map and the joint angle map, which are connected through a hidden layer. The temporal average filter computes the average of the activity in the joint angle map, and is connected to the output layer through a hidden layer. An arm model is also created for converting joint angles to spatial locations.

time steps, which can be expressed as:

$$a_i^F(t) = \frac{1}{t^{filter}} \sum_{t'=0}^{t^{filter}-1} a_i^J(t - t'). \tag{4.4}$$

Finally the joint angle output

$$\boldsymbol{\Theta}^{out} = f_{out}(\boldsymbol{a}^F(t^{out})), \tag{4.5}$$

is generated, where f_{out}, like f_{assoc} above, represents the fully-connected, two-layer feedforward net between the temporal average filter and the output nodes, and t^{out} is a timing parameter that controls the time when outputs are generated.

4.3.3 Training Methods

Training of the architecture is divided into three stages. The two maps are first trained separately. Then the the architecture learns to generate joint command output based on joint proprioception. Finally the inter-modality association is learned between the spatial and the joint angle maps.

4.3.3.1 Stage 1: Individual Map Training

In this first stage, the two maps are trained separately to obtain limit cycle representations for their respective afferent inputs. The training data for joint angle (proprioceptive) inputs are generated randomly, and they are run through the arm model to obtain data for spatial coordinates input. This is analogous to motor babbling in development stages of biological neural systems. Each data sample is presented to a map for 2 time steps from $t = 0$, after which the map continues to run and adapt for another $\tau = 4$ time steps (continuation time). This is done by specifying the gating parameters:

$$\alpha_{aff}(t) = \begin{cases} 0.64 & \text{if } 0 \leq t \leq 1 \\ 0 & \text{otherwise} \end{cases}; \; \alpha_{rec}(t) = \begin{cases} 0.36 & \text{if } 0 \leq t < 4 \\ 0 & \text{otherwise} \end{cases};$$

$$\alpha^{J}_{assoc}(t) = 0, \forall t. \tag{4.6}$$

At each time step, the afferent weights $\boldsymbol{w}_i$ are updated as (replacing Equations 3.7, 3.8):

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \mu_1 a_i(t)(I - \boldsymbol{w}_i(t)), \tag{4.7}$$

where μ_1 is a learning rate, and I denotes the afferent input, either $\boldsymbol{X}^M$ (spatial map) or $\boldsymbol{\Theta}$ (joint position map). This specifies a typical unsupervised SOM learning rule. The difference between Equation 4.7 and Equations 3.7, 3.8 is because the afferent inputs are now based on distances instead of inner products (Equation 4.2, 4.3). The recurrent weights are updated as usual using temporally asymmetric Hebbian learning based on Equations 3.9, 3.10. Upon completion of this stage, limit cycle representations are expected to occur in both maps when the maps are allowed to run for a longer period of time, e.g., by setting $\alpha_{rec}(t) = 0.36$ for $0 \leq t < 50$ in Eq. 4.6.

To test how an input is encoded by a limit cycle after training, spatial coordinates (x, y, z) are presented to the spatial map at $t = 0$ and 1. The activation parameter γ is fixed at 0 after training, meaning each non-winner has an activation value of 0 (inactive) while each winner has 1 (maximally activated). After the input is removed, the activity of the map goes through a brief period of irregular dynamics and eventually settles into a limit cycle attractor, a cyclically repeating sequence of activity patterns. This limit cycle is used as a representation of the corresponding afferent input, which, in this case, is spatial coordinates. As indicated in Section 3.3.3, similar inputs result in similar limit cycles. Figure 4.9 shows a sample limit cycle of length 6, where each activation pattern is sparsely-coded.

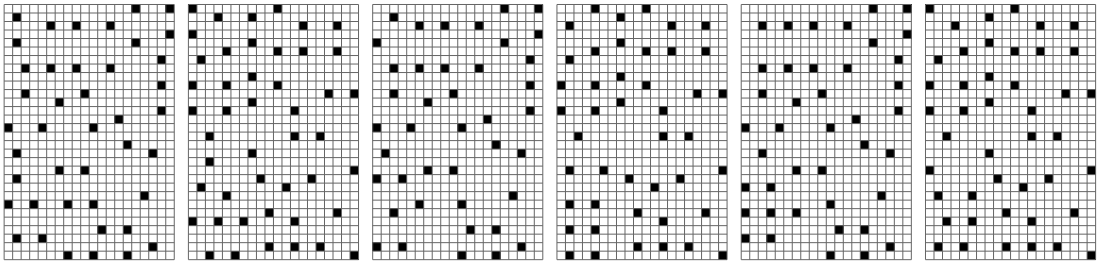

Figure 4.9: An example of a learned limit cycle of length 6. The patterns from left to right repeatedly occur at consecutive time steps. Each cell in each activity pattern represents a SOM node. Black cells represent "winner nodes". This particular limit cycle first appears at t = 9, meaning the left-most pattern appears at t = 9, 15, 21, $\cdots$, the second pattern at t = 10, 16, 22, $\cdots$, etc.

4.3.3.2 Stage 2: Joint Command Output

In this stage, the architecture learns to generate joint command outputs that match given joint proprioceptive inputs. Specifically, when a joint proprioception pattern $\mathbf{\Theta}$ is presented to the joint angle map, the goal is to eventually generate $\mathbf{\Theta}^{out}$ such that $\mathbf{\Theta}^{out} \approx \mathbf{\Theta}$. Again, the training samples are generated randomly. Each joint angle input in the training set is fed to the joint angle map, and the resulting limit cycle in the joint angle map activates the temporal average filter. The activity of the temporal average filter maintains a moving average of the joint angle map activity for the most recent t^{filter} time step, where t^{filter} is a parameter. The value of t^{filter} can be set rather arbitrarily, as long as it covers the lengths of most limit cycles. At a pre-specified output time t^{out}, output $\mathbf{\Theta}^{out}$ is generated by passing temporally averaged activity of the joint angle map through the two-layer feedforward net f_{out} (Equation 4.4). Again, t^{out} can be set rather arbitrarily, as long as it is late enough such that the activity dynamics of the joint angle map has entered a limit cycle.

This is because activity of a limit cycle is regular, and thus the results of temporal averages at different times are quite similar. Finally, f_{out} is trained using a resilient error-backpropagation (RPROP) method with input $\mathbf{\Theta}$ and error $\mathbf{\Theta} - \mathbf{\Theta}^{out}$. Notice that it is possible for different joint angles $\mathbf{\Theta}$ to be mapped to the same limit cycle due to SOM's discretization effect, and thus a limit cycle representing $\mathbf{\Theta}$ can correspond to multiple $\mathbf{\Theta}^{out}$ in training data. In this case, only the "most relaxed" joint angle among them, i.e., $\arg\min_{\mathbf{\Theta}} \|\mathbf{\Theta} - (.5, .5, .5)\|$, is used to train f_{out}.

4.3.3.3 Stage 3: Inter-Modality Associations

In this final stage, the associative connections between the spatial map and the joint angle map are trained. This stage is independent of the previous stage, and therefore could be performed before or in parallel with Stage 2. The goal of this stage is to learn the associations between spatial coordinates and joint angles, and to eventually be able to transform spatial inputs to joint outputs. To this end, a limit cycle in the spatial maps needs to be able to retrieve a corresponding limit cycle activity pattern in the joint angle map through the feedforward nets f_{assoc}. As noted earlier, this is a much harder problem than associating patterns for two conventional single-winner SOMs with static representations, because each limit cycle representation contains multiple activity patterns where each activity pattern contains distributed winning nodes.

Specifically, consider an activity sequence A^J in the joint angle map triggered by an arbitrary afferent input $\mathbf{\Theta}$, and the activity sequence A^S in the spatial map

triggered by the afferent input $\boldsymbol{X}^M$ corresponding to $\boldsymbol{\Theta}$, where $\boldsymbol{X}^M$ is the manipulator location by positioning the arm at $\boldsymbol{\Theta}$. Since A^J and A^S contain the limit cycles representing $\boldsymbol{\Theta}$ and $\boldsymbol{X}^M$, the goal is to train f_{assoc} to learn the associations between activity sequences in the two maps. That is, when the joint angle map's afferent input becomes unavailable, an activity sequence similar to A^J is to be retrieved in the joint angle map by the associative input sequence $f_{assoc}(A^S)$ alone (e.g., $\alpha^J_{aff} = 0$ and $\alpha^J_{assoc} > 0$). Here $f_{assoc}(A^S)$ represents the activity sequence resulting from passing each activity pattern in A^S through f^o_{assoc}.

To generate training data, a number of joint angles $\boldsymbol{\Theta}$ are randomly sampled, which leads to corresponding spatial locations $\boldsymbol{X}^M$. Each $\boldsymbol{X}^M$ and $\boldsymbol{\Theta}$ are then presented to their respective maps via afferent connections for 2 times steps, after which activity of the two maps continues being updated. Starting from a pre-specified time step t^{assoc}, the sequence of activity patterns in the next K (a fixed parameter) time steps in those maps are stored as ordered lists A^S and A^J, i.e., $A^{\cdot} = [\boldsymbol{a}^{\cdot}(t^{assoc}), \boldsymbol{a}^{\cdot}(t^{assoc}+1), \cdots, \boldsymbol{a}^{\cdot}(t^{assoc}+K-1)]$. Since the activity of the maps are limit cycles, the exact values of t^{assoc} and K are again not critical; they only need to be reasonably late and long enough to cover at least a large portion of a limit cycle.* As with the previous stage, it is possible for multiple different activity sequences in the joint angle map to correspond to the same activity sequences in the spatial map, due to discretization of SOMs. The activity sequence representing the most relaxed joint angles among them is selected as a training target.

*On the other hand, a large value for K slows down training significantly. Therefore, the value $K = 10$ is fixed empirically.

4.3.3.4 Learning Associations Between Sequences with Multi-Winner Patterns

In Stage 3, given training sequences A^S and A^J described above, each containing K ordered patterns, the goal is to map A^S to A^J through the feedforward network f_{assoc}. Training is based on error backpropagation. However, generic error backpropagation methods do not account for patterns resulting from multi-winners-take-all, and they do not naturally deal with sequence-to-sequence mapping with each sequence sampled from a limit cycle. Here new learning rules are described to address these two issues.

The standard error function calculates the squared distance between a target pattern $\boldsymbol{T}$ and an output pattern $\boldsymbol{O}$ as $E(\boldsymbol{T}, \boldsymbol{O}) = \sum_i (T_i - O_i)^2$. In the case of a multi-winner SOM, the target pattern $\boldsymbol{T}$ is a binary vector (elements $\in \{0, 1\}$) since $\gamma = 0$ (see Equation 3.6 and Figure 4.9). However, this error function does not account for the fact that the downstream component of the feedforward net is a SOM, which performs multi-winners-take-all for node activations, and is therefore too restrictive. The reason is that the generic error function unnecessarily drives O_i of a non-winning node i (i.e., $T_i = 0$) toward 0, while in fact, all that is needed is that i is not selected as a winner. To achieve this, O_i only needs to be smaller than the greatest value in its local neighborhood, i.e., $O_i < \max_{k \in \mathbb{N}_i} O_k$. This is realized by defining the following

alternative error function to guide learning:

$$E(\boldsymbol{T}, \boldsymbol{O}) = \sum_i (T_i' - O_i)^2,$$
$$T_i' = \begin{cases} 1; & \text{if } T_i = 1, \\ \zeta \max_{k \in \mathbb{N}_i} O_k; & \text{if } T_i = 0, \end{cases} \tag{4.8}$$

where $\zeta = 0.7$ is a discount parameter whose value is determined empirically.

Another adjustment is needed to address the fact that the goal of training is to retrieve a target sequence output, by feeding each consecutive pattern in an input sequence to the feedforward net to be trained. A naive approach would be to pair each pattern in the input sequence with the pattern of the same index in the target sequence for training. However, since both the target and input sequences are sampled from limit cycle dynamics, they both contain periodically repeating or at least partially repeating patterns, and therefore it is possible to train based on a cyclically rotated target (or input) sequence in order to generalize better.

Note that the input sequence A^S and the target sequence A^J each contains K patterns, $A^{\cdot} = (A^{\cdot}_1, A^{\cdot}_2, \cdots, A^{\cdot}_K)$. Let $0 \leq \delta < K$ be an alignment parameter such that the j-th pattern in A^S is paired with the $(j + \delta \mod K)$-th pattern in A^J, $j = 1, 2, \cdots, K$. There are K possible values for δ and thus K possible cyclic alignments between A^S and A^J. It is found that training with the alignment δ^* minimizing the sum of errors across all pattern pairs can eventually generalize better.

That is,

$$\delta^* = \arg\min_{\delta} \sum_{j=0}^{K-1} E(A^J_{[j+\delta \mod K]}, f_{assoc}(A^S_j)), \tag{4.9}$$

where the error function E is defined in Equation 4.8. The value of δ^* is updated for each epoch of training, and the resulting input-target pairs, $\langle A^S_j, A^J_{[j+\delta^* \mod K]} \rangle, j = 1, 2, \cdots, K$, are used to adapt weights of f_{assoc}. As will be shown later, the values of δ^* converge during training, and is critical for f_{assoc} to generalize.

4.3.4 Experimental Methods

After training, a new set of spatial coordinates for testing is provided as input to the architecture, specifying the target location to be reached by the arm. Each target location $\boldsymbol{X}^T$ (superscript T stands for target, not transpose) is generated by manually rotating the arm joints to each of a $10 \times 10 \times 10$ grid of points in the joint angle space (i.e., each $\theta_i = \{.05, .15, .25, \cdots, .95\}$), and then recording the manipulator's locations. These targets contain some extreme locations that are hard to reach. Since the space is continuous, it is nearly impossible that these test inputs appear in the training dataset, which is generated randomly, and therefore the results indicate how well the learned associations generalize. To evaluate the performance of the neural architecture, each target location $\boldsymbol{X}^T$ is presented as input to the architecture, and its output $\boldsymbol{\Theta}^{out}$ is passed through the arm model of Figure 4.7 to obtain resulting manipulator coordinates $\boldsymbol{X}^M$. The distance $\|\boldsymbol{X}^T - \boldsymbol{X}^M\|$ indicates spatial error.

The gate timing of the architecture is illustrated in Figure 4.10. The afferent input of the spatial map receives fixed target coordinates during the first two time steps,

while that of the joint angle map remains shut. The latter ensures that the architecture receives only target spatial coordinates as input. The recurrent connections for both maps remain open. From $t = 2$, the activity of the spatial map starts to settle in a limit cycle attractor. The associative connections between the two maps, initially closed, are opened at a fixed time $t = t^{assoc}$, at which point on the changing activity of the spatial map starts to drive the activity of the joint angle map, which is initially silent. In addition to the input from the spatial map, the activity of the joint angle map is also affected by itself through its own recurrent connections. Finally, at a fixed time $t = t^{out}$, the output of the temporal average filter, which maintains an average of the most recent t^{filter} activity patterns in the joint angle map, is open, and then a joint output is generated. Note that, as with training, the values of t^{assoc}, t^{out}, and t^{filter} can be set rather arbitrarily, only that they are sufficiently late such that the activity of the maps has entered a limit cycle attractor. More importantly, they do not depend on the exact timing of individual limit cycles (i.e., the exact onset times and the lengths), and thus the boundaries of limit cycles do not need to be detected by the architecture.

The results reported below are obtained using an architecture with 40×30 nodes in each map, and 200 nodes in each hidden layer. A total of 10 independent simulations are performed with different initial random weights. The average results are reported below. Unless otherwise noted, the timing parameters are: $t^{assoc} = 50$, $t^{out} = 130$, and $t^{filter} = 30$.

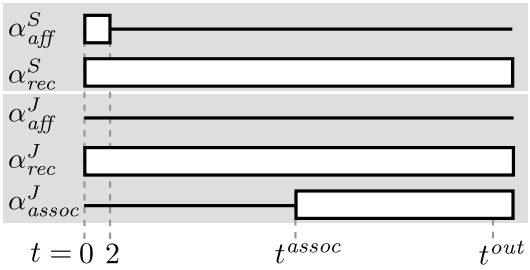

Figure 4.10: Summary of gate timing. The horizontal axis represents time. Parameters α are those in Equations 4.2, 4.3. A rectangular region indicates that the gate is open (connections enabled), while a horizontal lines indicates that the gate is closed. Values α_{aff} and α_{rec} denote the gates for the afferent and the recurrent connections of the SOMs, respectively. Value α^J_{assoc} denotes the gate for the incoming associative connections of the joint angle map. Values t^{assoc} and t^{out} indicate fixed timing parameters when α^J_{assoc} is open and when output joint angles are generated, respectively.

4.3.5 Map and Limit Cycle Formation

The two maps are separately trained during the first stage of training. Figure 4.11 shows the individual afferent weights of both maps after training. The initially random weights become self-organized into quasi-repetitive patterns of high and low value clusters, forming dark and light interleaved stripes. Although this result is certainly less smooth than conventional SOMs, because multi-winner activation is used and the maps are trained for limit cycles, its appearances are qualitatively similar to some cortical maps in biological neural systems that exhibit quasi-repetitive patterns, e.g., Swindale et al. (2000, Figure 1).

Figure 4.12 summarizes the lengths of the limit cycles formed in each map. On average, about 60% of the testing data results in a limit cycle of length 2 in each map, although there is high variation among different simulations, as indicated by the error bars (standard deviations). Other common lengths include 4, 6, 10, and 12. The

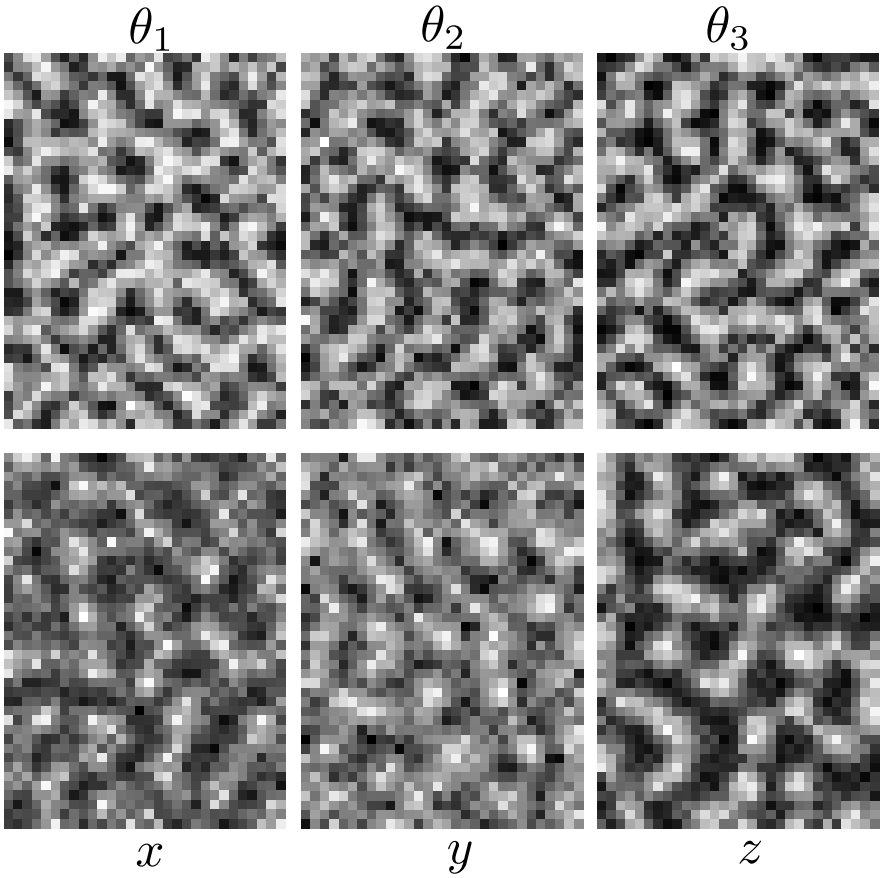

Figure 4.11: Map formation. Each subgraph plots a weight component of a map, namely θ_1, θ_2, and θ_3 of the joint angle map (top row) and x, y, and z of the spatial map (bottom row). Each cell in each subgraph corresponds to a node in a map. Lighter shade indicate a higher value, while a darker shade indicate a lower value.

lengths of limit cycles tend to be multiples of 2, 3, and 5, where smaller factors appear more frequently. Note that the architecture does not need to detect the lengths of limit cycles. They are shown here for illustration purposes only.

4.3.6 Convergence of Pattern Alignment

During the third stage of training (Section 4.3.3.3), when the inter-modality associative connections are being trained, an alignment parameter δ^* is used to align the input and output activity sequences. Since the value of δ^* is updated every training epoch

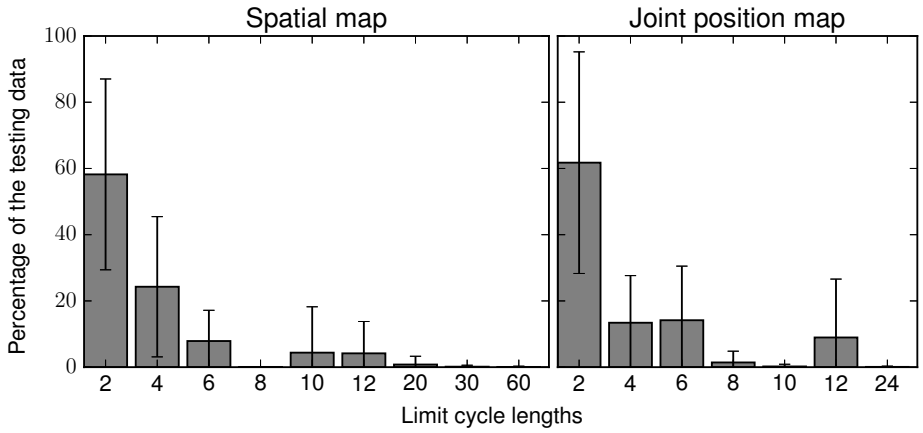

Figure 4.12: Lengths of limit cycles formed in both maps for the testing data.

(Equation 4.9), if its value keeps changing every epoch, the same input pattern will correspond to a different target pattern when adapting the weights, potentially causing the training process to diverge. To investigate this issue, the changes of alignment in a typical simulation are plotted in Figure 4.13. In the first 7 epochs, alignment changes occur with about 20% or higher of the training data, and the percentage may even rise, e.g., from epoch 3 to 7. Later, the alignment changes eventually drop to a low percentage and become stabilized, about 2% at epoch 30 and about 1% at epoch 40.

Such convergence can be considered as indicating that a "consensus" alignment has been reached among the training data, which initially prefer different alignments. Here "prefer" means the alignment that results in the least error (Equation 4.9). Suppose that initially each activity sequence prefers to align differently, there will be an alignment that is preferred by slightly more sequences than other alignments, resulting in slightly more overall influences on weight adaptation. Then the adapted weights in turn encourage more sequences in the training data to prefer this alignment, forming a positive feedback process. This process continues until nearly all sequences

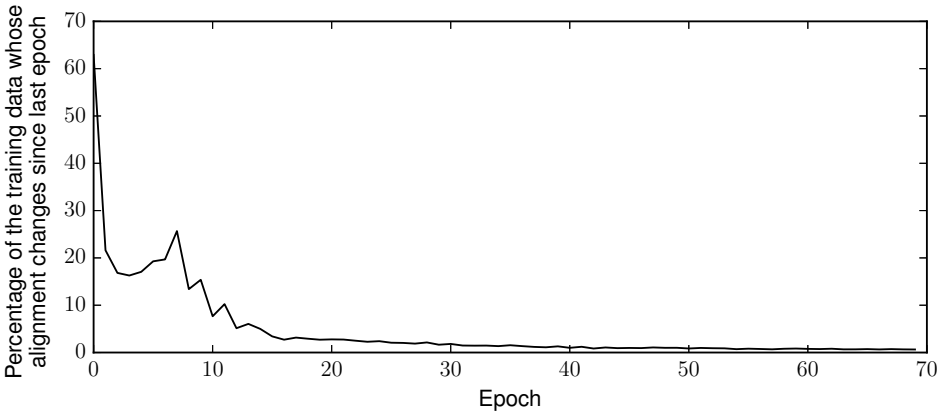

Figure 4.13: Convergence of limit cycle alignments in the course of training.

prefer the same alignment.

4.3.7 Spatial Error

Figure 4.14 shows a comparison of overall spatial errors $\|\boldsymbol{X}^T - \boldsymbol{X}^M\|$ before versus after training. Since the test dataset is never used during training, the results can be used to evaluate the degree of generalization. The value of spatial coordinates are normalized such that each of x, y, and z is within $[0, 1]$, so the spatial errors reported here are relative to a unit cube. Before training, the spatial errors of the testing data were widely distributed across the value range, while after training, they are clustered at low values. Errors less than 0.1 account for about 70% of the testing data. Numerically, the average spatial error before training is 0.5578 (SD=0.0522), while after training the value becomes 0.084 (SD=0.0031) quite consistently. The post-training error is about 15% of the pre-training error and 1/12 of the length of each axis. The architecture clearly generalizes to the test data.

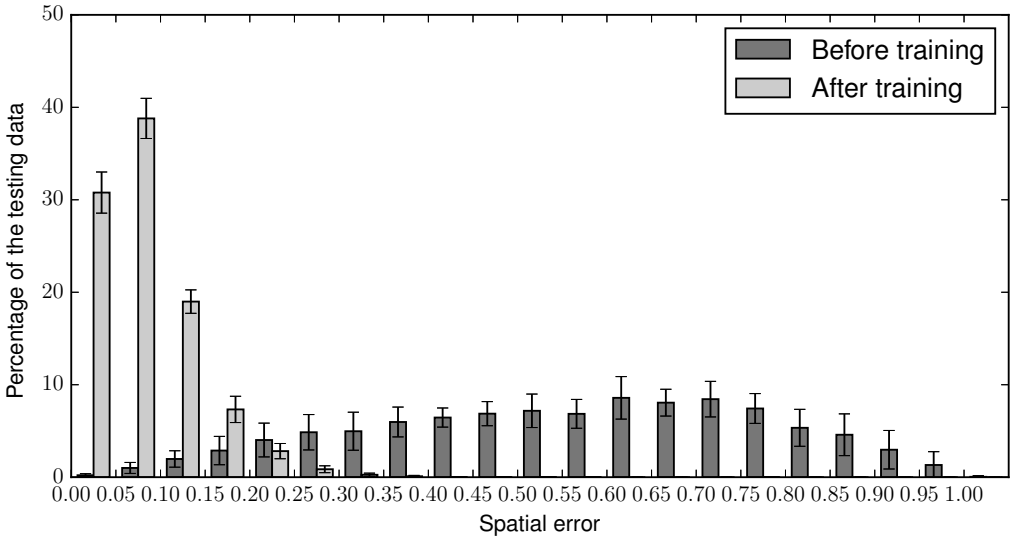

Figure 4.14: Distribution of spatial errors $\|\boldsymbol{X}^T - \boldsymbol{X}^M\|$ for the test data before and after training.

4.3.8 Sensitivity to Timing Parameters

Three timing parameters have been introduced in the architecture (Section 4.3.4), namely t^{assoc}, the time at which the associative connections between the spatial map and the joint angle map becomes active, t^{out}, the time at which a joint command output is generated, and t^{filter}, the number of time steps over which the temporal average filter computes average activity. In an architecture based on constantly changing activity, the overall performance could be very sensitive to these operation timings. This would be undesirable because the system becomes unstable and requires highly accurate control mechanisms. To determine the sensitivity, here the three timing parameters were varied in a typical simulation after training when all weights are fixed. Therefore, this also evaluates the situations where timing parameters during testing differ from those during training.

Figure 4.15a shows that the overall performance is insensitive to t^{assoc} after 20 time steps, although there are slight fluctuations. This result indicates that the choice of t^{assoc} is quite unrestrictive. The only requirement is that it is late enough such that the dynamics in the spatial map have entered a limit cycle. Similarly, Figure 4.15b indicates that t^{out} can be chosen quite arbitrarily, only that it is later than t^{assoc} by at least $t^{filter} = 30$ time steps, such that the temporal average filter gathers all 30 patterns from the joint angle map. Another important question is how stable the outputs are, since the internal activity of the architecture is non-fixed point. To examine the stability, the architecture continually sends output joint angles to control the arm for a number of time steps, and the manipulator location at each time step is recorded. Figure 4.15c shows the changes of manipulator locations at each time step compared with the previous time step. The fluctuations are quite small, around 5×10^{-5}. Finally, Figure 4.15d shows the effects of t^{filter} on the spatial error. Small t^{filter} results in generally worse performance, although even t^{filter} values tend to perform better than odd ones. This is because the lengths of the limit cycles tend to be multiples of 2. As t^{filter} increases, this effect diminishes and the value of t^{filter} becomes insensitive to even or odd numbers. Therefore, like the other two timing parameters, the choice of t^{filter} can be rather arbitrary, as long as it is reasonably large.

4.4 Summary and Discussion

This chapter discussed how associations can be learned to retrieve limit cycle representations. This is important because forming associations is a basic operation shared

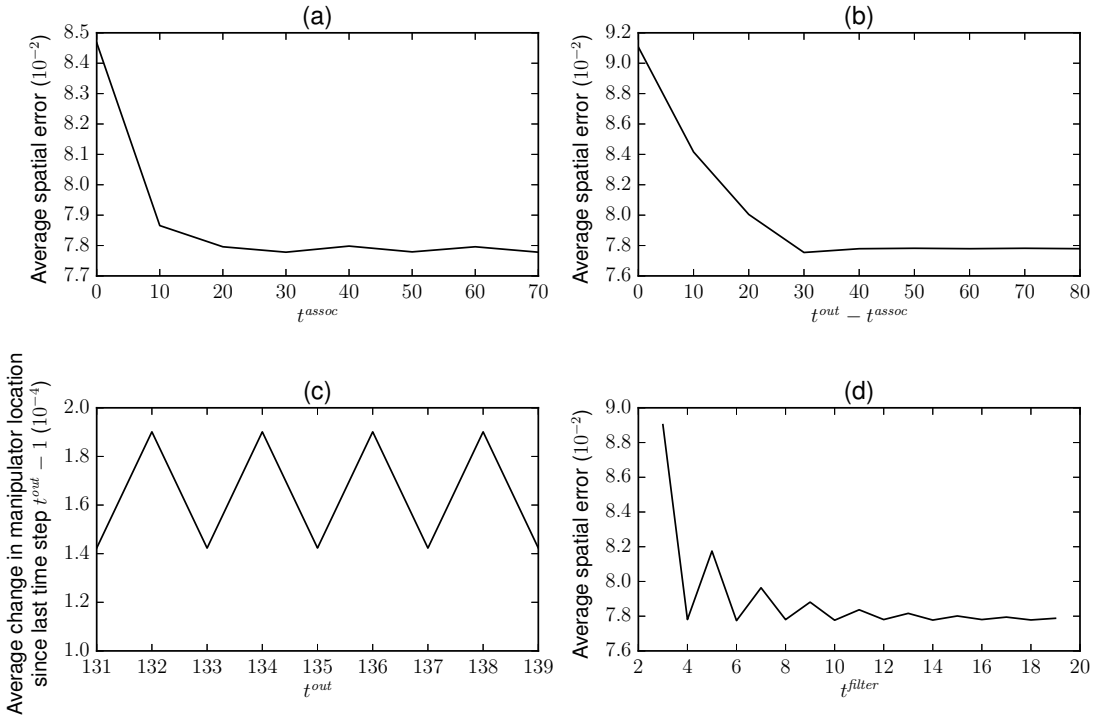

Figure 4.15: Effects of timing parameter values. The baseline values are: $t^{assoc} = 50$, $t^{out} = 130$, and $T^F = 30$. (a–b) and (d) show the effects of t^{assoc}, $t^{out} - t^{assoc}$, and T^F on the spatial error, respectively. (c) shows how much the manipulator moves in each output time step, indicating how stable the output is.

by many neural architectures and the brain, and it was not known a priori whether learning associations is possible at all for maps having limit cycle activity. In the first experiment, limit cycles were shown to be retrievable by static representations in an object naming task. The second experiment showed that, in the same task, a limit cycle can be retrieved by another limit cycle using a dual-route connectivity between SOMs. Finally, in the third experiment, limit cycle associations are learned using revised learning rules to allow *generalized* mapping between continuous input and output spaces of an arm control problem. To this end, associations from limit cycles to static outputs must be learned too, which is an inverse problem of that in the first

experiment.

In addition to that associations *can* be learned to retrieve limit cycle representations, such associations appeared to be robust. This robustness was evident in the first experiment in that associative connections driven by a fixed input pattern were enough to recall a limit cycle. Such retrieval of limit cycles was also shown to be resistant to the presence of noise, a distinctive advantage over retrieving static representations. This property can be attributed in part to the stability of limit cycle attractors, as has been discussed in Section 3.2.3.

While timing control may not be an important issue for fixed-point neural architectures, it becomes much more relevant in architectures based on dynamic activity patterns, since the activity patterns are constantly changing. In the latter case, if operation of the architecture relies on highly specific timing, not only does it require a highly accurate control mechanism that usually incurs a large overhead, but the system also becomes less robust, because missing a specific activity pattern can potentially cause the system to fail. Throughout the three experiments in this chapter, timing controls were shown to be quite relaxed. For example, associative connections between two SOMs can be enabled at a rather arbitrary time and never disabled for the same input, so that one does not have to consider exactly when to enable and when to disable the connections. Additionally, in no situation did the timing parameters depend on the timing of individual limit cycles, i.e., the onsets/lengths. This means that limit cycles do not need to be explicitly detected, and thus timing parameters can be fixed. Results also showed that varying timing parameters within reasonable limits generally did not significantly result in deteriorating performance. All these properties

are encouraging in terms of enabling a temporally-robust neural architecture to be built based on limit cycle associations.

Similarly to Chapter 3, the critical issue of generalization was studied in the third experiment involving continuous spaces, i.e., whether a learned association can invoke proper limit cycle activity, and thus generate a proper output, when given a new input it never saw during learning. This was investigated in the context of an open-loop arm control problem using a neural architecture involving two SOMs and additional neural circuitry for generating fixed outputs. It contained 5 layers of nodes in the output generation path, resulting in many degrees of freedom. The simulation results indicated that the architecture did generalize reasonably well using a 3-stage training method. It did so by self-organizing limit cycle representations in individual maps, by self-organizing alignments between limit cycles in the two maps during training, and by temporally averaging limit cycles when generating constant persisting motor outputs. Note that not only associations between the two limit cycle SOMs, but also those between a limit cycle SOM and the static output layer, are learned in this experiment. One possible source of error was the discretization error of a SOM. Although the representational capacity of multi-winner limit cycle activity is much greater than single-winner activity, there are still different inputs being grouped into the same limit cycle. Other possible sources of error came from generalization error of the associative connections between the two maps and between the temporal average filter and the output layer. To further reduce the error, spatial error feedback must be accounted for by the architecture, forming a closed-loop system.

— Chapter 5 —

A Limit-Cycle SOM Architecture for Stable Arm Control

Given that a SOM can encode input stimuli using limit cycle attractors (Chapter 3), and that associations can be learned between limit cycle SOMs (Chapter 4), the practicality of limit cycle representations in applications remains uncertain. For example, it is unclear whether such attractor activity can be harnessed to drive a neural architecture containing multiple SOMs. It seems plausible that an architecture based on oscillatory activity could be trained to generate oscillatory output patterns that are coupled with its internal activity. However, it is less clear whether such oscillatory activity can be used at a more abstract level for more general, *static* non-oscillatory computations, such as holding a robot's arm in a fixed position, and whether an architecture based on limit cycle SOMs can generalize effectively to new situations.

Traditional non-oscillatory SOMs have been used in solving inverse kinematics arm control problems, which is to find appropriate joint angles for an arm to reach a target spatial location (Barreto et al., 2003). In approaches using a single conventional SOM, the SOM is usually trained using concatenated vectors of joint angles and Cartesian coordinates (Lallee and Dominey, 2013; Saxon and Mukerjee, 1990). Similar strategies can be found in (Angulo and Torras, 2008; Walter and Ritter, 1996), but these discrete

SOM nodes are interpolated to sample a continuous manifold. In (Kumar and Behera, 2010; Martinetz et al., 1990), each SOM node stores extra information (e.g., a matrix) for performing locally linear transformations from the Cartesian space to the joint angle space. In approaches based on neural architectures containing multiple non-limit cycle SOMs, it is typical that inputs in the joint angle space and those in the Cartesian space are processed separately by two unimodal SOMs first. These unimodal SOMs are then associated directly or through other SOMs, such that appropriate joint angles can eventually be retrieved by Cartesian coordinates (Kajić et al., 2014; Lallee and Dominey, 2013). A more biologically-realistic architecture for visually-guided arm reaching is characterized by iterative competition among SOM nodes (Ménard and Frezza-Buet, 2005).

This current chapter builds on prior results and creates a neural architecture containing multiple limit cycle SOMs for physical robotic arm control, where a manipulator (hand, gripper, etc) of an arm is to be moved to a *fixed* target location and to be *maintained there* in a steady fashion despite the continually oscillating SOM activity associated with limit cycle dynamics. An advantage of choosing this task is that the performance can be physically validated using a robot. The arm control problem has been studied intensively in robotics and cognitive science (Bullock et al., 1993; Colomé and Torras, 2015; Flash et al., 2013; Hutchinson et al., 1996). While Section 4.3 partially addressed this in a preliminary way, the architecture it used lacked the spatial precision that is required for practical use, it contained solely an open-loop controller, and the system behaviors were not well understood (e.g., arm trajectory). The arm considered here is composed of multiple rigid segments connected

by rotatable joints. Thus, the key question becomes whether it is possible for a neural controller consisting of multiple interconnected limit cycle SOMs to arrive at a vector in the joint angle space (i.e., a set of rotation angles for all joints) that brings an arm manipulator to target spatial coordinates in 3D space, *and then holds the manipulator fixed there* in spite of the continuously changing activity in the limit cycle SOMs.

While the intention is not to build a neuroanatomically accurate model of the brain, the neural architecture created here is strongly inspired by neurophysiological findings, including self-organization of cortical maps, rhythmic neural oscillations, and different feedforward/feedback modes of human motor control. A primary finding is that the architecture generalizes with high spatial precision to new spatial targets that were never seen during training. The robustness of this limit cycle architecture is also explored. It is shown that when a part of the neural system is disabled or disrupted in a variety of ways, the overall functionality is impaired but not completely destroyed. For example, timing control is simple and forgiving: each neural component's activity is readable by downstream components at a relatively arbitrary time without overall performance being substantially compromised. The architecture's operation is found to be independent of the exact timing of individual limit cycles (i.e., their start time and length), so that the overall neural architecture does not need to explicitly detect onset of limit cycles. Three other key results are also presented. First, in spite of the fact that the internal activity of the system is oscillatory, its outputs can be stabilized so as to drive the arm to a fixed target in a smooth trajectory without apparent oscillations and then maintain it at that position. Second, the concurrent open-loop and closed-loop control methods, each having its own limitations in arm movement

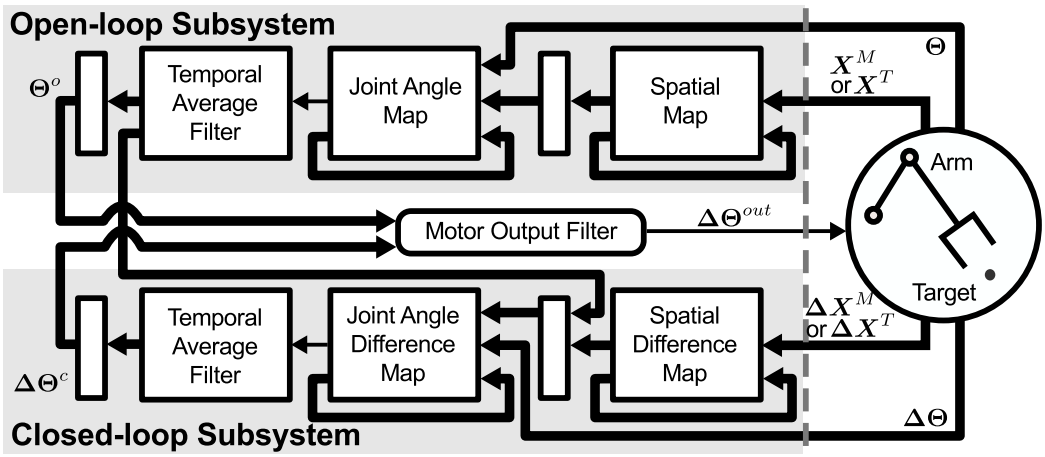

Figure 5.1: An overview of the neural architecture considered here. To the left of the gray dashed line, the neural architecture is divided into open-loop and closed-loop subsystems that contain four SOMs with locally-recurrent feedback connections. Narrow rectangles without labels represent hidden layers. Thick arrow lines represent trainable synaptic connections. The input/output of the architecture is from/affects the state of the arm model (right of the gray dashed line).

control when used in isolation, together collectively achieve a precise and smooth reaching movement. Finally, the results of disabling different parts of the architecture are reminiscent of symptoms of human patients or animal experiments when different brain regions are impaired.

5.1 Overview

Figure 5.1 gives an overview of the system. To the left of the vertical dashed line is the neural architecture considered here, including open-loop and closed-loop subsystems. To the right is an external environment, including an arm and a target location to be reached. During training, the arm joints are moved randomly in small steps, resembling motor babbling (Bullock et al., 1993; Guenther and Barreca, 1997). The neural architecture reads, from the external environment, arm joint angles $\mathbf{\Theta}$, Cartesian

coordinates of the manipulator $\boldsymbol{X}^M$, and their corresponding difference vectors $\boldsymbol{\Delta\Theta}$ and $\boldsymbol{\Delta X}^M$, the latter generated by subtracting the preceding joint angles and manipulator locations from the current ones during each motor babbling step. After training, the neural architecture is given a target set of Cartesian coordinates $\boldsymbol{X}^T$ to reach for (superscript T stands for "target", not transpose). The architecture runs iteratively to guide a reaching movement step-by-step, where each iteration corresponds to a small step in the arm trajectory. In addition to $\boldsymbol{X}^T$, the external environment also provides current spatial error $\boldsymbol{\Delta X}^T$, a unit vector in Cartesian space pointing from the manipulator location towards the target location. Based on $\boldsymbol{X}^T$, $\boldsymbol{\Theta}$, and $\boldsymbol{\Delta X}^T$, the architecture generates a joint command $\boldsymbol{\Delta\Theta}^{out}$ at each iteration dictating joint rotations to be made by the arm, and eventually guiding the manipulator towards the target. While the manipulator location $\boldsymbol{X}^M$ can be derived using current joint angles $\boldsymbol{\Theta}$ (i.e., based on proprioception), determining target location $\boldsymbol{X}^T$ is a non-trivial visual image processing task in physical robotics. To simplify the problem, here it is assumed that a proper vision system is in place to determine Cartesian coordinates of both the target and the manipulator, and is also able to compute spatial difference vectors between them (i.e., $\boldsymbol{\Delta X}^T = \boldsymbol{X}^T - \boldsymbol{X}^M$).

5.2 A Simulated Environment

The schematic of the arm used here is identical to that described in Section 4.3 (Figure 4.7). A 3-degree-of-freedom (3-DOF) arm operating in a 3D Cartesian space is used, having two shoulder and one elbow joint freely adjustable. This arm is modeled

in a virtual environment named SMILE (Simulator for Maryland Imitation Learning Environment) for training and testing the neural architecture, before a real robotic arm is used. While I originally built SMILE for the purposes of robot imitation learning, the virtual robotic arm in SMILE can be used alone for generating training data and for testing and visualizing arm movement control in this work. The current joint angles of the virtual arm are represented by $\mathbf{\Theta} = (\theta_1, \theta_2, \theta_3)$, where each component is linearly scaled to be within $[0, 1]$. In contrast to Section 4.3, where the arm is used to translate joint angles to spatial coordinates in one step, the arm is enhanced here to integrate small pieces of joint angle differences to form a trajectory. Specifically, when given a joint command $\mathbf{\Delta\Theta}^{out} = (\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)$, each joint angle θ_i is updated to be $\theta_i + \lambda\Delta\theta_i$ and then clamped within $[0, 1]$, where $\lambda = 1/60$ is the step size. Since the lengths of all arm segments are known, the manipulator location $\boldsymbol{X}^M = (x, y, z)$ can be determined exactly using the Denavit-Hartenberg method (Hartenberg and Denavit, 1965, p.435) based on current joint angles $\mathbf{\Theta}$. Each component of $\boldsymbol{X}^M$ is also linearly scaled to $[0, 1]$.

5.2.1 Simulator for Maryland Imitation Learning Environment (SMILE)

For the purpose of visually inspecting robot arm movements, a software environment named SMILE is described here. I originally built SMILE for supporting robot imitation learning (outside the scope of this dissertation), but its functionalities can also be used separately for other purposes, such as generating training data and visually validating neural controllers for robotic arms as are done here.

SMILE is an integrated virtual 3D environment where a robot and an invisible human demonstrator (user) coexist (Huang et al., 2015a,b). They can both interact with/manipulate a variety of task-related objects placed on a tabletop. An example view of the simulated world as seen by the human demonstrator is given in Figure 5.2, which contains a 3D environment and overlaid graphical user interface (GUI) controls (Figure 5.2; top-right). The environment can be observed through a freely navigable perspective (Figure 5.2; main window) and/or through the robot's perspective (Figure 5.2; bottom-right). Real-time rigid body physics simulation is included based on the Bullet physics library.

SMILE features a Matlab programming interface for fast prototyping of robot behaviors. A user program acts like the "brain" of the robot. That is, a user program receives sensory inputs in the form of vision (images) and proprioception (joint angles) information, and provides motor outputs in the form of motor commands (joint velocities). This framework is intended to facilitate work on bio-inspired robot controllers. Since all object states (e.g., location, orientation, shape, etc.) are known exactly in the simulation, a user program can elect to bypass visual image processing altogether and instead focus on other parts of robot behaviors while using exact object states as sensory inputs. This provides the flexibility for developing robot controller that may not be possible in physical environments. For testing limit cycle SOM architectures, a target spatial location $\boldsymbol{X}^T$ can be set to be the exact 3D coordinates of an object, and $\boldsymbol{\Delta X}^T$ can be calculated exactly as well, both without actual image processing that is out of the scope of this research.

The other features of SMILE, although not directly related to multi-SOM architec-

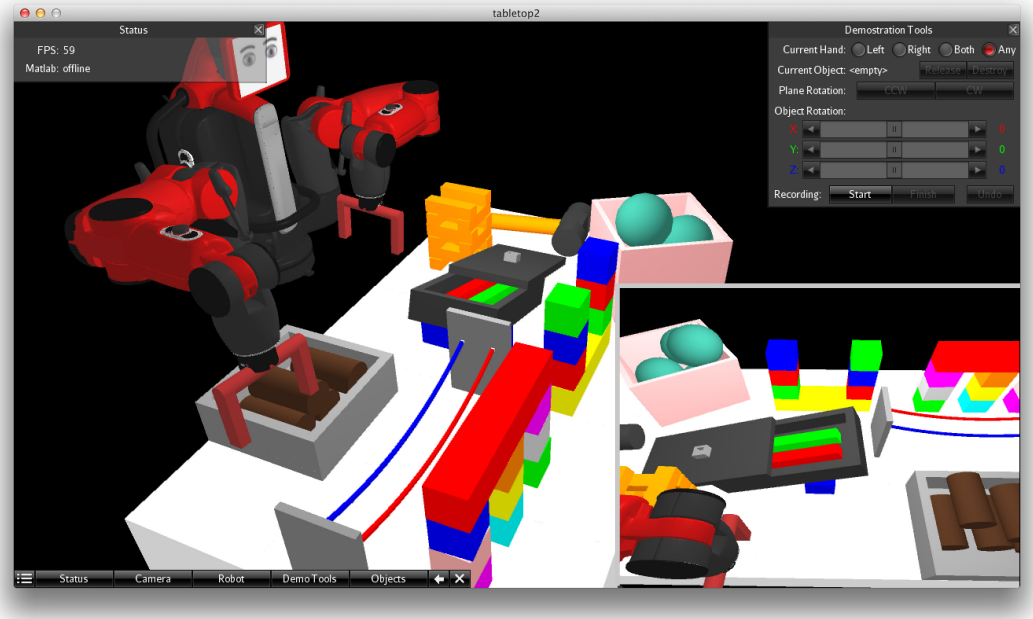

Figure 5.2: An example view of the SMILE software simulation. The main screen shows a simulated environment containing a tabletop and a variety of objects (block, boxes, lids, strings, etc.), as they are seen by the human demonstrator. The avatar for a two-armed programmable robot is also embedded in the environment. The bottom-right corner shows the environment as it is seen by the robot. The top-right corner shows a sample GUI window. The user can manipulate the environment, including picking up, moving, rotating, and releasing objects, as well as changing the viewpoint and creating objects, and does so to demonstrate procedures to the robot.

tures that this research investigates, include an object initialization interface and a demonstration interface. The object initialization interface provides XML semantics to create objects in the virtual environment. Apart from several predefined simple objects, a user can define complex objects by composing simple objects, or by importing a shape file in STL format (Figure 5.3). On the other hand, the demonstration interface allows a human user to demonstrate how a task is done by manipulating objects. Objects can be picked up, released, moved, and rotated via mouse control and overlaid GUI (Figure 5.4). A demonstration is recorded as a sequence of object movements that

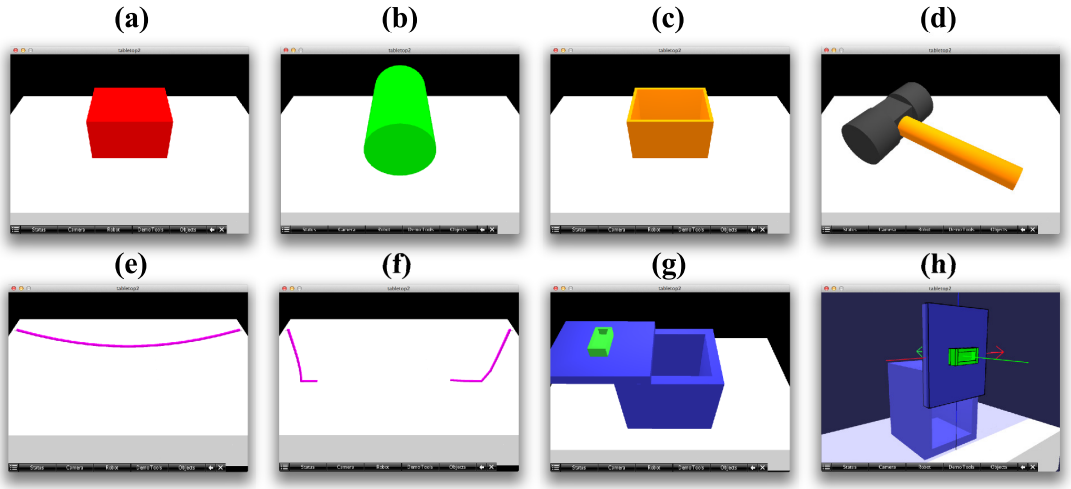

Figure 5.3: Sample objects generated via XML files. (a)–(c) show simple objects: a block, a cylinder, and a box container. (d) shows a composite object, a hammer. (e), (f) show a string that when cut dynamically falls onto the table. (g), (h) show a box container with a sliding lid.

is then used for robot imitation learning. The body movements of the demonstrator are purposely hidden from the robot for testing the virtual demonstrator hypothesis: In many situations, effective imitation learning of a task can be achieved when the demonstrator is invisible (Huang et al., 2015a,b).

An example workflow when using SMILE for an imitation learning task is briefly described here. The goal of this task is to use a given set of 3D blocks to construct letters "UM" (for University of Maryland). First, a user instantiates the set of blocks using the XML object initialization interface. The user then demonstrates how the blocks should be arranged using the demonstration interface of SMILE (Figure 5.5). A robot agent then learns from the recorded demonstration (the mechanisms of which are beyond the scope of this dissertation), and then attempts to perform the task in the same 3D environment in SMILE (Figure 5.6). Finally, the robot agent is ported to

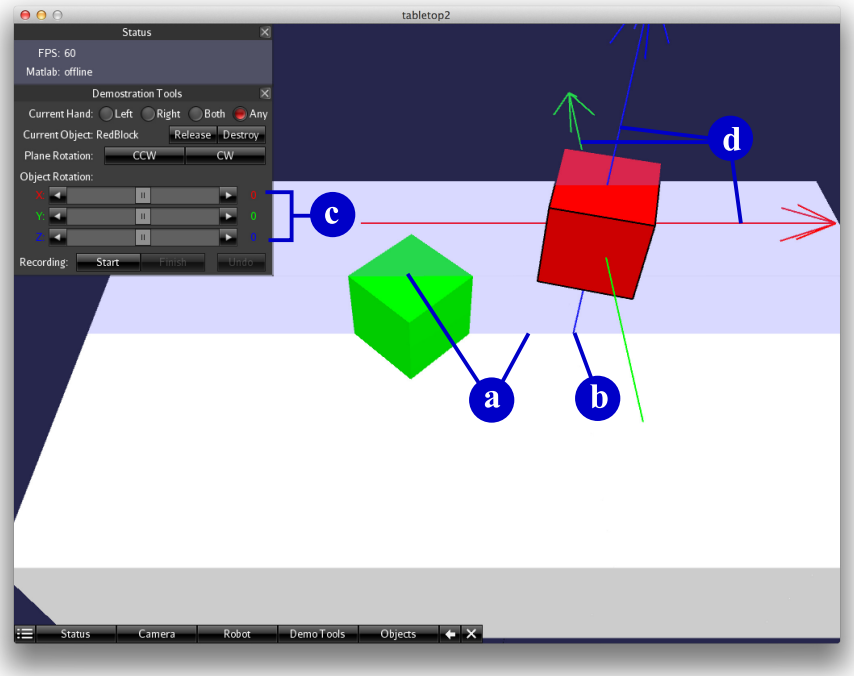

Figure 5.4: The demonstration interface. The red block (right) is currently grasped by the demonstrator and raised above the tabletop. (a) A restricting plane perpendicular to the tabletop that guides movement of the grasped object in the 3D world. The plane can be rotated around the vertical axis to select different moving paths. (b) A virtual shadow indicating the location of the object, represented by a vertical line connecting the object's center of mass and the tabletop. (c) Sliders that control object rotations around (d) the three primary axes (color coded).

a physical robot and performs the task in the real world (Figure 5.7). The goal state as demonstrated in SMILE and that as performed by the robot agent are qualitatively similar (Figure 5.8).

Although SMILE mainly concerns robot imitation learning, its robot programming interface can serve for fast prototyping robot controllers and visualizing their behaviors. In this research for example, the simulated robot is used to visualize the behaviors of a neural architecture based on limit cycle SOMs before the architecture is ported to a physical robot. This lowers the risks of accidentally damaging the robot due to

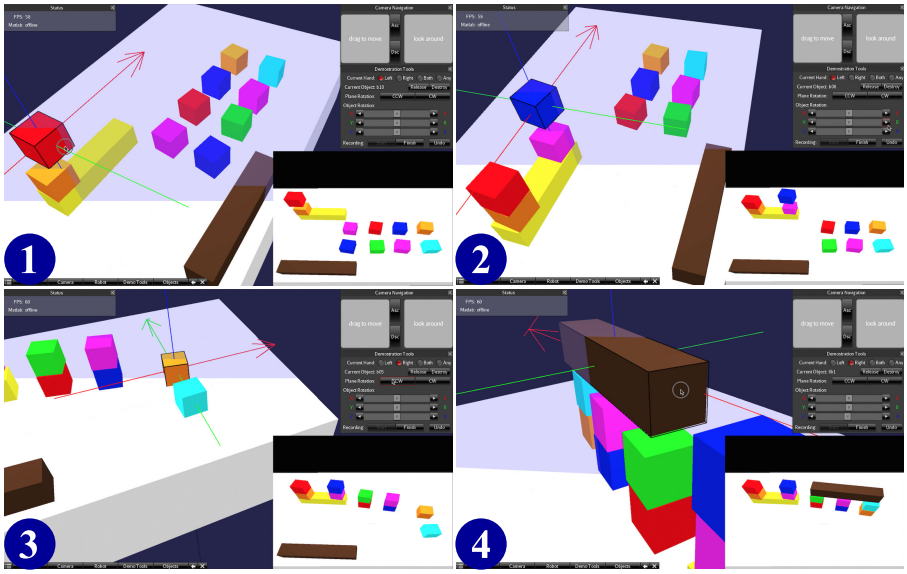

Figure 5.5: Screenshots during a demonstration of stacking blocks into the letters "UM" using the GUI and mouse inputs. The screenshots are temporally ordered in (1)–(4). The main view of each image shows the demonstrator's view of the simulated environment. The lower-right corner of each image shows the robot's view of the demonstration.

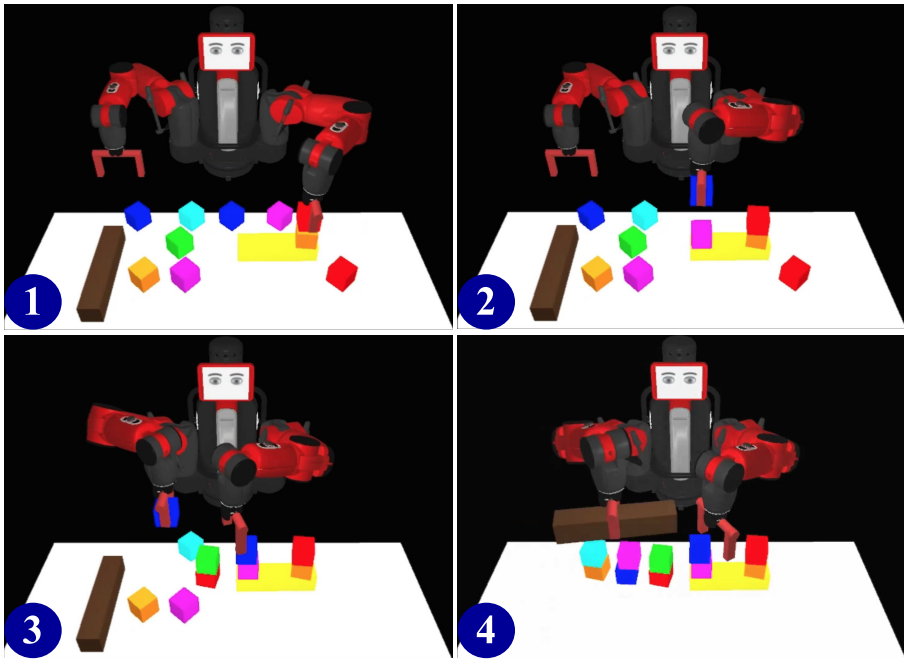

Figure 5.6: The robot's successful imitation of stacking blocks into the letters "UM". Screenshots are taken in the order of (1)–(4). The robot's learning mechanisms are out of the scope here.

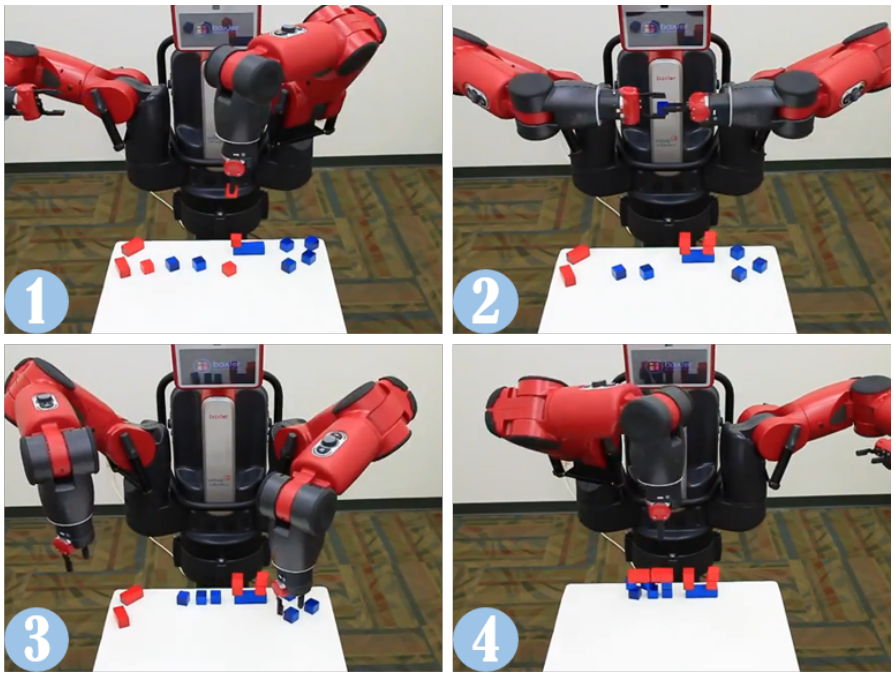

Figure 5.7: The physical robot's successful imitation of stacking blocks into the letters "UM". Pictures are taken in the order of (1)–(4). The robot's learning mechanisms are beyond the scope of this dissertation.

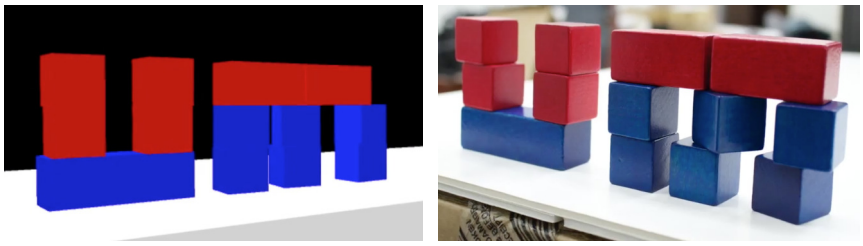

Figure 5.8: The goal state demonstrated in SMILE (left) and the one executed by the robot (right).

inappropriate controller design or buggy code.

5.3 Neural Architecture

This section describes the neural architecture for the stable arm control problem. The output of the neural architecture is in the form of a joint difference vector $\mathbf{\Delta\Theta}^{out}$, which serves as a joint command that rotates the arm joints for a short period of time (i.e., ideally 1/60 second). The result is a small-step contribution to arm movement. To generate a complete arm trajectory for a reaching movement, the architecture is run multiple times, each of which is referred to as an *iteration*. In each iteration, the architecture undergoes multiple *time steps*, each of which is a discrete time unit when neural activity in the architecture is updated. This allows neural activity to dynamically change within an iteration.

Existing methods for robotic arm control utilize either open-loop control, closed-loop control, or a combination of both. *Open-loop methods* perform one-step computations to transform target spatial coordinates to a motor plan. There is no feedback control that refines the motor plan based on the outcome of motor execution. This approach avoids continuously monitoring the manipulator location and computing motor plan refinement at the cost of lower accuracy. In contrast, *closed-loop methods* continuously compare the current manipulator location with the target location, and use the differences as feedback to generate and refine motor plans. Close-loop methods typically use cameras to obtain visual feedback (i.e., visual servoing (Chaumette and Hutchinson, 2006; Hutchinson et al., 1996)). The accuracy of reaching is much better

in closed-loop approaches, although instability sometimes occurs as a result of latency between sensing and acting, feedback bandwidth issues, inaccurate estimation of 3D locations, etc. Carefully tuned control gain is necessary to avoid instability. Some past work combines open-loop and closed-loop methods to better control reaching movements (Gaskett and Cheng, 2003; Nori et al., 2007), and the proposed architecture is based on this latter approach.

The output of the neural architecture at each iteration is the result of integrating individual outputs of the open-loop and the closed-loop subsystems (see Fig. 5.1). The open-loop subsystem is weighted more heavily in the early iterations, and the closed-loop subsystem is weighted more heavily in later iterations. The open-loop subsystem takes a target spatial location $\boldsymbol{X}^T$ and generates a target joint angle output $\boldsymbol{\Theta}^o$, similar to the architecture described in Section 4.3. Since the target, and thus the input $\boldsymbol{X}^T$, remains the same throughout the reaching movement, in practice the open-loop subsystem needs to be run only once in the first iteration. However, the accuracy of such an approach tends to be low because it does not take any feedback about how well the arm performs. On the other hand, during each iteration the closed-loop subsystem takes a spatial difference unit vector $\boldsymbol{\Delta X}^T$ indicating the direction of spatial error from the manipulator to the target, as well as the current joint angles $\boldsymbol{\Theta}$ (indirectly via the joint angle map), and generates a joint angle difference output $\boldsymbol{\Delta\Theta}^c$. This output determines how the arm joints are to be rotated in the current iteration, based on feedback information about spatial error between the manipulator and the target. Therefore, the closed-loop subsystem can achieve higher accuracy, although it has to recompute output at each small movement step. In general, the output of the

open-loop subsystem has a critical role in the early phase of the reaching movement, to bring the manipulator towards the target without peripheral feedback processing. The output of the closed-loop subsystem is then gradually weighted more later in the movement, which is crucial to fine-tune the manipulator location to approach the target with relatively higher accuracy. Intuitively, this roughly corresponds to motor controls that progressively switch from a feedforward mode to a feedback mode (Jordan and Wolpert, 1999).

5.3.1 Open-loop Subsystem

The open-loop subsystem corresponds to spatial-to-motor memory that makes a one-step decision about the joint angle that is associated with the given target location. It does not rely on visual feedback and has relatively low accuracy. Since the given target location is assumed to remain unchanged throughout arm reaching, the open-loop subsystem is required to run once only per reaching movement. The neural architecture uses an open-loop subsystem in the early stages of an arm trajectory to quickly guide the manipulator to the vicinity of the target.

The open-loop subsystem contains two SOMs, as illustrated in Figure 5.1 (top). This part of the architecture is identical to the neural architecture described in Section 4.3 and is briefly recapitulated here. The open-loop subsystem contains two SOMs: a *spatial map* encoding Cartesian coordinates $\boldsymbol{X} = (x, y, z)$ and a *joint angle map* encoding joint angles $\boldsymbol{\Theta} = (\theta_1, \theta_2, \theta_3)$ (Equations 4.2, 4.3). They are connected by a set of fully-connected two-layer associative connections via a hidden layer that is

represented as function $f^o_{assoc}(\cdot)$ (note the added superscript o standing for open-loop). In order to generate steady joint outputs in spite of oscillatory joint angle map activity, a temporal average filter is added downstream of the joint angle map. The filter contains the same number of nodes as the joint angle map, and each filter node connects one-to-one to each map node. The activity of each filter node maintains a moving average of its corresponding map node activity for the most recent t^{filter} time steps, and finally the open-loop output is generated (note the notation changes from Equations 4.4, 4.5; superscripts o are added):

$$a_i^{F,o}(t) = \frac{1}{t^{filter}} \sum_{t'=0}^{t^{filter}-1} a_i^J(t - t'), \tag{5.1}$$

$$\boldsymbol{\Theta}^o = f^o_{out}(\boldsymbol{a}^{F,o}(t^{out})), \tag{5.2}$$

where f^o_{out}, like f^o_{assoc} above, represents the fully-connected, two-layer feedforward net between the temporal average filter and the motor output filter. Vector $\boldsymbol{a}^{F,o}(t)$ represents the activity pattern of the temporal average filter at time step t. Parameter t^{out} represents a fixed time step at which the output is taken. Although here the output is taken at a specific time step, it has been shown in Section 4.3.8 that continuous outputs with an open system alone yield stable joint angles with very small oscillations (e.g., around 5×10^{-5} in a normalized Cartesian space). This means that t^{out} can be chosen quite arbitrarily, but it must be late enough such that the values of $\boldsymbol{a}^{F,o}(t)$ become stabilized.

5.3.2 Closed-loop Subsystem

The closed-loop subsystem iteratively transforms spatial errors to joint displacements, guiding the manipulator towards the target in small steps, each of which is called an iteration. Keeping track of spatial errors requires visual and proprioceptive feedback to compare the location of the manipulator with that of the target. The neural architecture mostly uses the closed-loop subsystem in later iterations of an arm trajectory to fine-tune the location of the manipulator, and thereby achieve higher accuracy. Specifically, in each iteration, the closed-loop subsystem runs for multiple time steps starting from $t = 0$ and eventually generates an output vector. This output vector represents a desired direction for rotating arm joints (a unit vector in the joint angle space) during the current iteration, based on the currently observed spatial error direction (a unit vector in the Cartesian space) as well as the current joint angles. In the next iteration, t is reset to 0 and the process is repeated.

Like the open-loop subsystem, the closed-loop subsystem contains two SOMs (Figure 5.1, bottom), a *spatial difference map* and a *joint angle difference map*, which encode unit difference vectors in Cartesian and joint angle spaces ($\boldsymbol{\Delta X}$ and $\boldsymbol{\Delta\Theta}$), respectively. During training, these difference vectors are generated by adding small random perturbations to current joint angles (motor babbling), and the resulting differences of manipulator locations are recorded (Bullock et al., 1993). These joint angle and spatial difference vectors are then normalized so that they specify the directions but not the magnitudes of the spatial or joint angle differences. This reduces the necessary number of training samples and network sizes needed because they do not

need to account for different magnitudes in the same difference direction. Therefore, the two SOMs in the closed-loop subsystem contain fewer nodes than those in the open-loop subsystem, although they are otherwise structurally identical to those open-loop SOMs. There is also a set of fully connected associative connections via a hidden layer between the two SOMs. However, a distinction in the closed-loop subsystem is that there is an additional set of incoming connections from the open-loop subsystem's joint angle map via its downstream temporal average filter (see Figure 5.1), which conveys neural activity encoding current joint angles to the closed-loop subsystem joint angle difference map. As such, the net inputs h_i for each node i in the two closed-loop maps at time step t, given unit vectors $\boldsymbol{\Delta X}$ and $\boldsymbol{\Delta\Theta}$ as afferent inputs, are similar to Eqs. 4.2 and 4.3:

$$h_i^{SD}(t) = -\alpha_{aff}^{SD}(t)\left\|\boldsymbol{\Delta X} - \boldsymbol{w}_i^{SD}\right\|^2 + \alpha_{rec}^{SD}(t)\boldsymbol{a}^{SD}(t-1)\cdot\boldsymbol{u}_i^{SD}, \tag{5.3}$$

$$\begin{aligned} h_i^{JD}(t) = &-\alpha_{aff}^{JD}(t)\left\|\boldsymbol{\Delta\Theta} - \boldsymbol{w}_i^{JD}\right\|^2 + \alpha_{rec}^{JD}(t)\boldsymbol{a}^{JD}(t-1)\cdot\boldsymbol{u}_i^{JD} \\ &+ \alpha_{assoc}^{JD}(t) f_{assoc}^{c}(\boldsymbol{a}^{SD}(t), \boldsymbol{a}^{F,o}(t)), \end{aligned} \tag{5.4}$$

where superscripts *SD* and *JD* correspond to the spatial difference map and the joint angle difference map, respectively, and f_{assoc}^{c} represents the fully connected, two-layer feedforward net between the two maps (superscript c represents closed-loop). Notice that f_{assoc}^{c} takes an additional input vector $\boldsymbol{a}^{F,o}$, the activity of the temporal average filter downstream of the joint angle map. The output activation of the two maps follows Equations 3.5, 3.6.

Similarly to Equations 5.1, 5.2, oscillatory activity in the joint angle difference

map needs to be stabilized to near-static outputs, so it is passed through a temporal average filter and then a two-layer feedforward net. The output of the closed-loop subsystem for each iteration is

$$a_i^{F,c}(t) = \frac{1}{t^{filter}} \sum_{t'=0}^{t^{filter}-1} a_i^{JD}(t - t'), \tag{5.5}$$

$$\mathbf{\Delta\Theta}^c = f_{out}^c(\boldsymbol{a}^{F,c}(t^{out})), \tag{5.6}$$

where t^{filter} is the same parameter as in Equation 5.1, and f_{out}^c represents the fully-connected, two-layer feedforward net connecting the temporal average filter to the motor output filter. The activation of f_{out}^c is designed to generate unit vectors, and the learning rules are derived accordingly. Therefore, $\mathbf{\Delta\Theta}^c$ is a unit vector informing the direction in the joint angle space in which the arm joints are to be rotated. Without considering the magnitude, the network is focused on the direction aspect of joint angle differences, and is expected to generalize better in that regard. The parameter t^{out}, as in Equation 5.2, is selected such that it is late enough that the values of $\boldsymbol{a}^{F,c}(t)$ become stabilized.

5.3.3 Motor Output Filter

At time step t^{out} of each iteration, the motor output filter (see Figure 5.1) aggregates the outputs from both the open-loop and the closed-loop subsystem and generates a

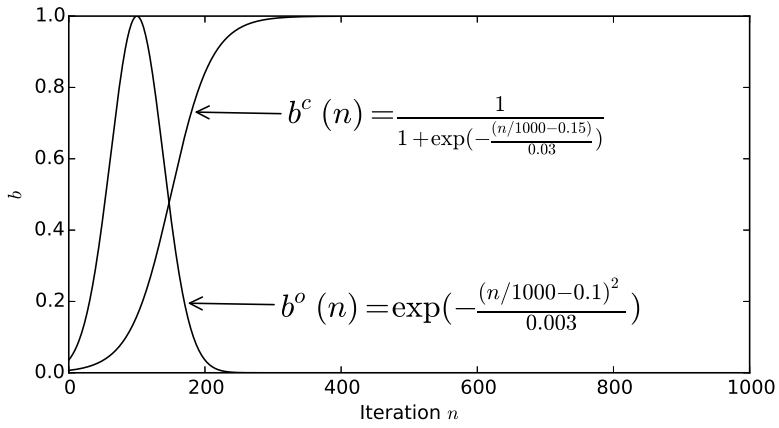

Figure 5.9: Weighting functions that modulate the influences of the open-loop (b^o) and closed-loop (b^c) outputs on the final output of the architecture.

motor output. A motor output at each iteration n is computed as

$$\boldsymbol{\Delta\Theta}^{out}(n) = b^o(n)\left(\boldsymbol{\Theta}^o - \boldsymbol{\Theta}\right) + b^c(n)\boldsymbol{\Delta\Theta}^c(n)\left(\sigma\left\|\boldsymbol{X}^T - \boldsymbol{X}^M\right\|\right), \tag{5.7}$$

where $\boldsymbol{\Theta}^o$ is the output of the open-loop subsystem and $\boldsymbol{\Delta\Theta}^c(n)$ is that of the closed-loop subsystem at iteration n, and $\boldsymbol{\Theta}$ is the current joint angle. The value of $\boldsymbol{\Theta}^o$ is independent of n and remains fixed throughout the whole arm trajectory because its input, the target location, remains fixed. In practice the open-loop subsystem is run once at the beginning of an arm reaching event, where its output $\boldsymbol{\Theta}^o$ is cached in the motor output filter. Functions $b^o(n)$ and $b^c(n)$ serve as multiplicative weights/gains that modulate the influences of the open-loop and closed-loop subsystems on the final motor output for each iteration n. Their values are given in Figure 5.9. Similar modulatory weights b have been used in past neural models to control the degree to which a motor plan is converted to actions, the mechanism of which is believed to be

related to the basal ganglia or the prefrontal cortex (Gentili et al., 2015). In early iterations, influences of both subsystems increase, with the open-loop subsystem rising more rapidly and contributing more. In later iterations, the situation reverses, with the closed-loop subsystem gradually dominating the motor output. Intuitively, this corresponds to an initially open-loop coarse movement progressively shifting to a closed-loop fine-tuning movement. Also notice that since the closed-loop output $\boldsymbol{\Delta\Theta}^c(n)$ is a unit vector informing direction but not magnitude, the value of $\sigma \left\| \boldsymbol{X}^T - \boldsymbol{X}^M \right\|$ serves as the magnitude of joint rotations, where $\sigma = 10$ is a constant scaling factor determined empirically. Notice that as the manipulator approaches the target, the amount of joint rotation becomes smaller, and this helps stabilize the manipulator near the target.

5.4 Training Methods

The organizational similarity of the open-loop and the closed-loop subsystems allow for similar training processes. Training of the architecture is divided into three stages, similarly to those described in Section 4.3.3. The four maps are first trained individually using unsupervised learning. The two subsystems are then trained separately to generate outputs based on the joint angle map and the joint angle difference map. Finally the inter-modality associations are learned between the spatial map and the joint angle map, as well as between the spatial difference map and the joint angle difference map. Training data are generated using a virtual robotic arm modeled in SMILE (Section 5.2.1). The end result is that when given a spatial location

input, the open-loop subsystem can generate a proper joint angle output, and that when given a spatial difference input, the closed-loop subsystem can generate a proper joint angle difference output.

5.4.1 Stage 1: Individual map training

In this first stage of learning, the four maps in the architecture are trained individually using unsupervised learning, to obtain limit cycle representations for their respective afferent input domains (see Figure 5.10a). Each map independently forms internal representations for external stimuli in different modalities: spatial location, spatial difference, joint angle, and joint angle difference. The training data are sampled randomly in each modality, where the spatial difference and joint angle difference data are then normalized to unit lengths. Each data sample is presented to each map for 2 time steps at $t = 0$ and $t = 1$, after which the map continues to run and adapt for another $\tau = 4$ time steps (continuation time). Detailed gate timing and learning rules are described in Section 4.3.3.1. Upon completion of this stage, limit cycle representation is expected to occur in all four maps when the maps are allowed to run for a period of time after inputs end at $t = 1$, e.g., using a large continuation $\tau = 50$.

5.4.2 Stage 2: Joint command output

In this stage, the two subsystems learn to generate outputs that match given joint proprioceptive inputs (see Figure 5.10b). Specifically, for the open-loop subsystem,

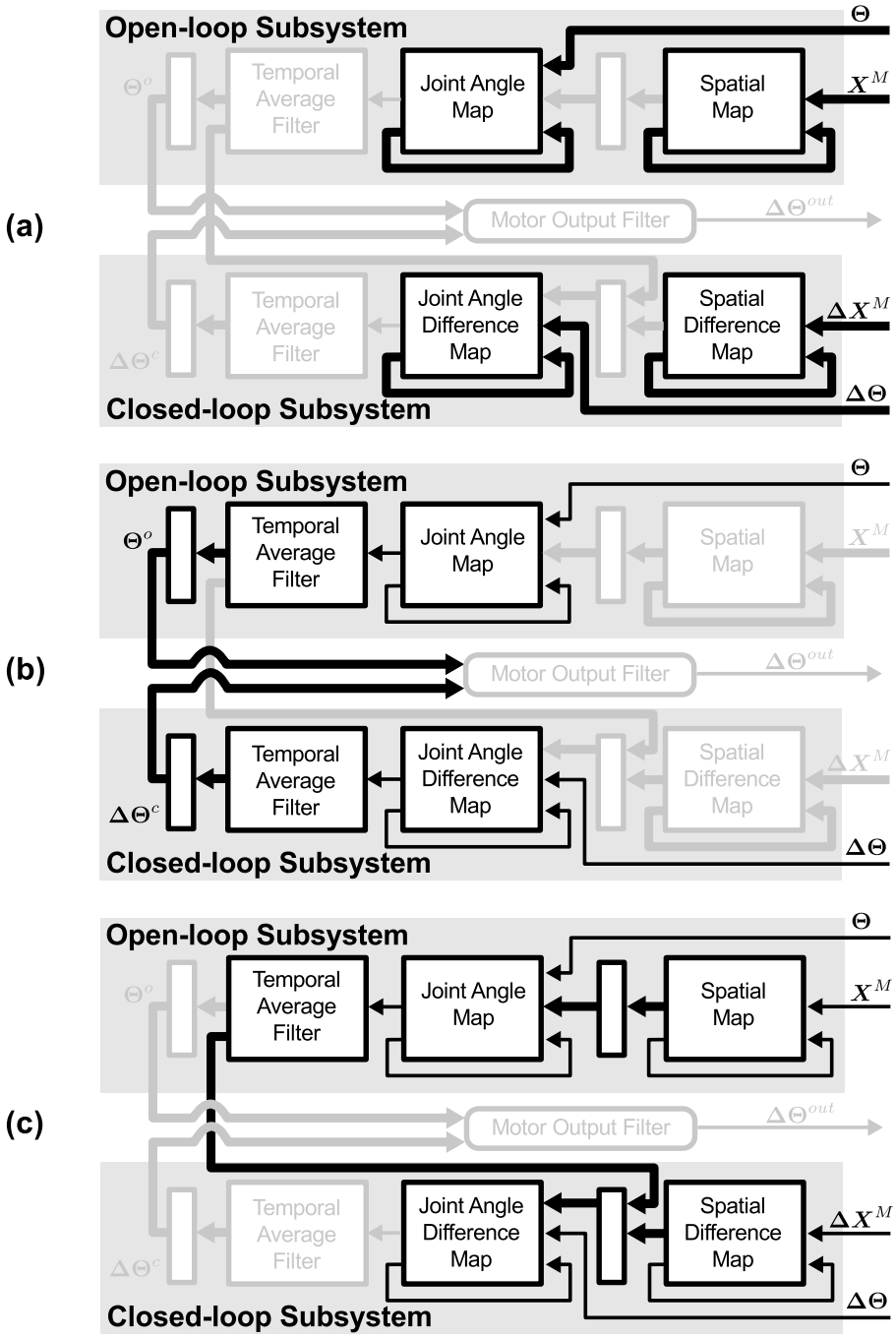

Figure 5.10: Schematic illustration of the three training stages. Temporarily disabled components and connections are grayed out. Thick arrow lines indicate the connections that are being adapted. Thin arrow lines indicate the connections that are used to spread neural activity but are not being adapted. (a) Stage 1: individual map training. (b) Stage 2: learning joint command outputs. (c) Stage 3: learning inter-modality associations.

when a joint proprioception pattern $\mathbf{\Theta}$ is presented to the joint angle map, the goal is to eventually generate $\mathbf{\Theta}^o$ such that $\mathbf{\Theta}^o \approx \mathbf{\Theta}$. Similarly, the closed-loop subsystem is to learn to generate $\mathbf{\Delta\Theta}^c \approx \mathbf{\Delta\Theta}$, where $\mathbf{\Delta\Theta}$ is a proprioceptive input to the joint angle difference map. The training samples $\mathbf{\Theta}$ and $\mathbf{\Delta\Theta}$ are again generated randomly. Output networks f^o_{out} and f^c_{out} are trained using an error backpropagation method with inputs $\mathbf{\Theta}$ and $\mathbf{\Delta\Theta}$ as well as errors $\mathbf{\Theta} - \mathbf{\Theta}^o$ and $\mathbf{\Delta\Theta} - \mathbf{\Delta\Theta}^c$, respectively (see Section 4.3.3.2 for more details). In the case that multiple inputs ($\mathbf{\Theta}$ or $\mathbf{\Delta\Theta}$) are mapped to the same limit cycle due to SOM's discretization effect, the "most relaxed" $\mathbf{\Theta}$, i.e., $\arg\min_{\mathbf{\Theta}} \|\mathbf{\Theta} - (.5, .5, .5)\|$, is selected for training f^o_{out}, and an arbitrary $\mathbf{\Delta\Theta}$ is selected for training f^c_{out}.

5.4.3 Stage 3: Inter-modality associations

In this final stage, the associative connections between the spatial map and the joint angle map (open-loop), as well as those between the spatial difference map and the joint angle difference map (closed-loop), are trained (see Figure 5.10c). This stage is independent of the previous stage, and therefore could be performed before or in parallel with Stage 2. The goal of this stage is to train f^o_{assoc} and f^c_{assoc} to associate limit cycle activity in the spatial and spatial difference maps with that in the joint angle and joint angle difference map, respectively. The training methods are described in detail in Sections 4.3.3.3, 4.3.3.4 and are summarized below.

Training data are generated by first sampling random joint angles $\mathbf{\Theta}$. Corresponding

spatial locations $\boldsymbol{X}^M$ can be obtained using the arm model:

$$\boldsymbol{X}^M = Arm(\boldsymbol{\Theta}). \tag{5.8}$$

For each $\boldsymbol{\Theta}$, a set of small joint angle differences $\boldsymbol{\Delta\Theta}$ are also generated. The set of spatial differences $\boldsymbol{\Delta X}^M$ can then be obtained by:

$$\boldsymbol{X}' = Arm(\boldsymbol{\Theta} + \boldsymbol{\Delta\Theta}), \tag{5.9}$$

$$\boldsymbol{\Delta X}^M = \boldsymbol{X}' - \boldsymbol{X}^M. \tag{5.10}$$

Each $\boldsymbol{X}^M$ and $\boldsymbol{\Theta}$, as well as normalized $\boldsymbol{\Delta X}^M$ and $\boldsymbol{\Delta\Theta}$, are then presented to their respective maps. Starting from a pre-specified time step t^{assoc}, the sequence of activity patterns in the next $K = 10$ time steps in those maps are stored as ordered lists A^S, A^J, A^{SD}, and A^{JD}. Additionally, the activity sequence of the temporal average filter downstream of the joint angle map is recorded as $\overline{A^J}$. Note that generating these five sequences of training data does not involve detecting limit cycles. In the case that the same A^S (A^{SD}) corresponds to multiple different A^J (A^{JD}), only the most relaxed A^J (an arbitrary A^{JD}) is used.

The goal here is to train f^o_{assoc} and f^c_{assoc} such that $f^o_{assoc}(A^S) \approx A^J$ and $f^c_{assoc}(A^{SD} \oplus \overline{A^J}) \approx A^{JD}$, where $\oplus$ represents vector concatenation since f^c_{assoc} takes additional inputs from the open-loop temporal average filter. Each pattern pairs from input and output sequences are used for training based on an error backpropagation method. However, two issues must be addressed to make it work (Section 4.3.3.4). First, since

the target patterns are the results of a multi-winners-take-all process, the error function must be replaced by Equation 4.8 to account for multi-winner patterns. Second, since the training sequences are sampled from limit cycle activity, their constituent patterns can be aligned differently. An "optimal" alignment that minimizes overall errors for each training sequence must be selected for each training epoch.

5.5 Experimental Methods

After training, a set of 3D spatial locations are selected as targets for arm reaching for validation and evaluation purposes. The procedures described below are first conducted using the virtual robotic arm in SMILE (Section 5.2.1), followed by a physical robotic arm. Each test target $\boldsymbol{X}^T$ is generated by manually rotating the arm joints to each of a $10 \times 10 \times 10$ grid of points in the joint angle space (i.e., each $\theta_i = \{.05, .15, .25, \cdots, .95\}$), and then recording the manipulator's locations. This ensures that the targets are reachable and include some extreme locations that are hard to reach. Note that each $\boldsymbol{X}^T$ may be reachable by multiple joint angles, and therefore the joint angles generated by the trained architecture may be different from those at the $10 \times 10 \times 10$ grid points.

The neural architecture is run for multiple iterations until the spatial error, measured as the Cartesian distance between the manipulator and the target location $\|\boldsymbol{X}^T - \boldsymbol{X}^M\|$, stays below a predetermined threshold ϵ for 20 consecutive iterations, where ϵ represents a target accuracy. The value $\epsilon = 0.001$ is used in the following. The stopping criteria are that the manipulator not only is positioned close to the target,

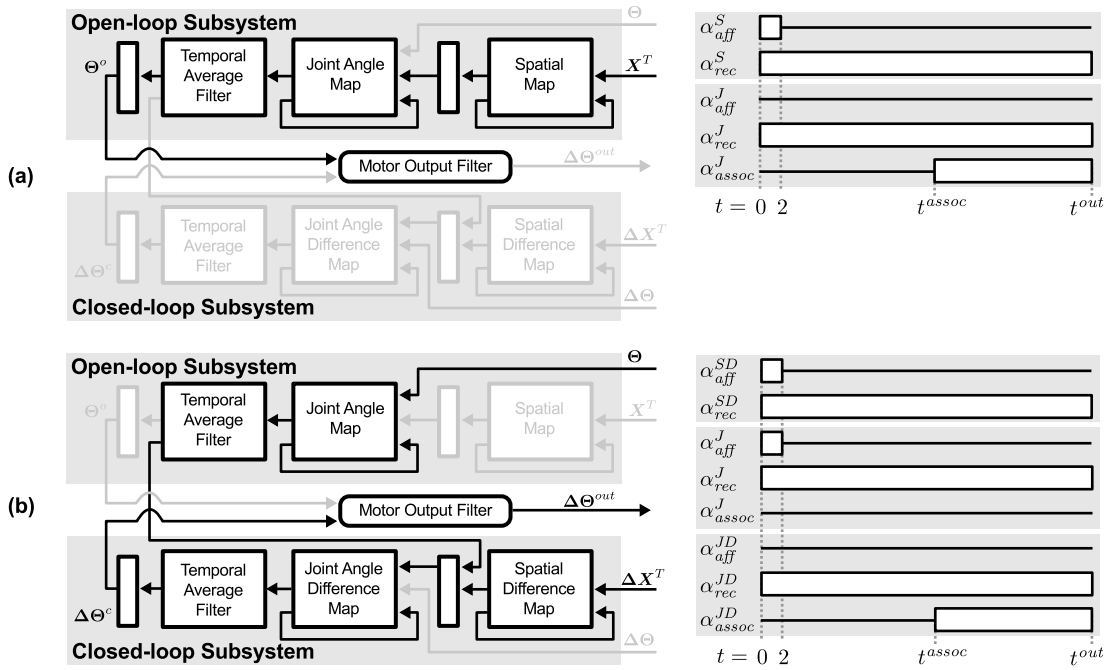

Figure 5.11: Schematic illustration of the execution of (a) the open-loop and (b) the closed-loop subsystems. The left column shows neural components and connections involved, and the right column shows the gate timing of the enabling (thick bar) and the disabling (thin line) of connections. See Equations 4.2, 4.3, 5.3, and 5.4 for the roles of α.

but also that it maintains that position for a period of time, which implies stability of the arm. If the stopping criteria are never met, the architecture stops at the 1000th iteration. In either case, the final spatial error and the number of iterations used are reported as performance metrics of the architecture.

For each target, the open-loop subsystem is run with input $\boldsymbol{X}^T$ and generates output $\boldsymbol{\Theta}^o$. Since the input to the open-loop subsystem is never changed for the same spatial target, practically it is run once only in the first iteration and the output $\boldsymbol{\Theta}^o$ is cached in the motor output filter. On the other hand, the closed-loop subsystem is run once per iteration, taking inputs $\boldsymbol{\Delta X}^T = (\boldsymbol{X}^T - \boldsymbol{X}^M)/\|\boldsymbol{X}^T - \boldsymbol{X}^M\|$ (spatial error direction to the target) and $\boldsymbol{\Theta}$ (current joint angles) and generating outputs $\boldsymbol{\Delta\Theta}^c$. At

the motor output filter, the outputs from both subsystems are integrated according to Equation 5.7 and a final motor output $\mathbf{\Delta\Theta}^{out}$ is generated for each iteration. This output is then used to modify the arm joints as:

$$\mathbf{\Theta} \leftarrow \max(\min(\mathbf{\Theta} + \lambda\mathbf{\Delta\Theta}^{out}, 1), 0), \tag{5.11}$$

where $\lambda = 1/60$ is a step size, and the combination of min and max functions clamps the value of $\mathbf{\Theta}$ in $[0, 1]$. As a result, running the architecture for an iteration causes the arm to move a small step. The updated arm joint angles are then used to calculate a new $\boldsymbol{X}^M$, and thus a new $\mathbf{\Delta}\boldsymbol{X}^T$, for the next iteration.

Gating and timing for running the open-loop subsystem are shown in Figure 5.11a. The spatial map receives as afferent input the target location $\boldsymbol{X}^T$ in the first two time steps, while the afferent input for the joint angle map remains disabled (see α_{aff} rows). The recurrent connections for both maps remain open (α_{rec} rows). From $t = 2$, the activity of the spatial map starts a brief irregular dynamics and then settles in a limit cycle attractor. The associative connections f^o_{assoc} between the two maps, initially closed, are opened at a fixed time $t = t^{assoc}$ (see α^J_{assoc}). From this point on, the changing activity of the spatial map starts to drive the activity of the joint angle map, which is initially silent. In addition to the input from the spatial map, the activity of the joint angle map is also affected by itself through its own recurrent connections. Finally, at a fixed time $t = t^{out}$, the output $\mathbf{\Theta}^o$ is generated by passing the activity of the temporal average filter, which maintains an average of the most recent t^{filter} activity patterns in the joint angle map, through the output feedforward net f^o_{out}.

A similar process is performed for the closed-loop subsystem (see Figure 5.11b), except that the joint angle map also participates in computing output $\mathbf{\Delta\Theta}^c$. That is, the current joint angle $\mathbf{\Theta}$ is fed to the joint angle map at the first two time steps (α^J rows). Then, starting from $t = t^{assoc}$, averaged joint angle map activity, together with limit cycle activity in the spatial difference map, jointly triggers limit cycle activity in the joint angle difference map that eventually generates $\mathbf{\Delta\Theta}^c$.

As with training, the values of t^{assoc} and t^{out} can be set rather arbitrarily anytime after the activity of the maps has entered limit cycle attractors. The value of t^{filter} can also be fixed rather arbitrarily. Importantly, they do not depend on the exact timing of individual limit cycles (i.e., the exact start time and the length), and thus the boundaries of limit cycles do not need to be detected by the architecture.

The results reported below are obtained using an architecture with $40 \times 30 = 1200$ nodes in each of the spatial and joint angle maps, $30 \times 20 = 600$ nodes in each of the spatial difference and joint angle difference maps, 200 nodes in each of the hidden layers of the open-loop feedforward nets f^o_{assoc} and f^o_{out}, and 150 nodes in each of the hidden layers of the closed-loop feedforward nets f^c_{assoc} and f^c_{out}. A total of 10 independent simulations are performed in each computational experiment with different initial random weights each time. The average results are reported below. Unless otherwise noted, the timing parameters used are: $t^{assoc} = 50$, $t^{out} = 130$, and $t^{filter} = 30$.

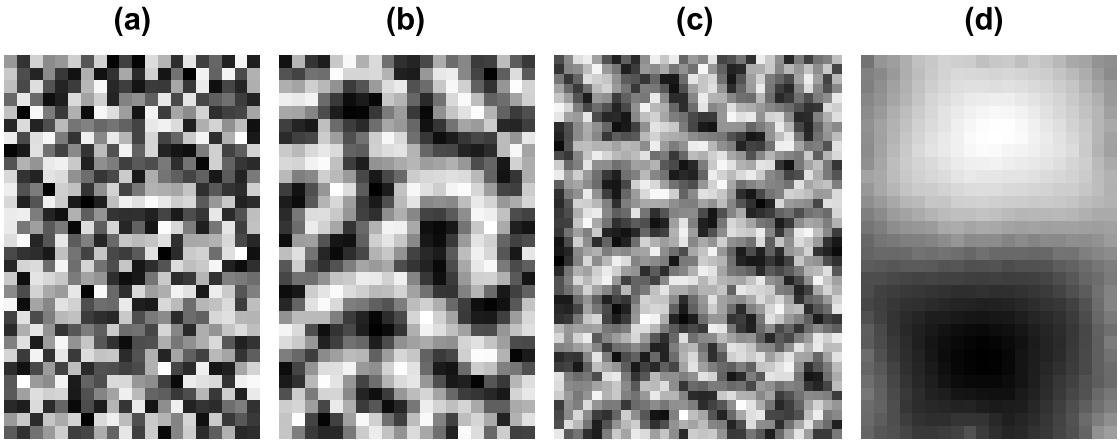

Figure 5.12: Examples of map formation. Each cell represents the value of the same selected afferent weight for each node. Brighter shades indicate higher values. (a) The random pre-training weight values corresponding to input Δy in the spatial difference map. Coordinate y is defined in Figure 4.7. (b) The post-training weight values for (a). (c) The post-training weight values corresponding to y in the spatial map, which contains more nodes than the spatial difference map. (d) For comparison purposes, the results of training a conventional SOM is also plotted. This SOM is trained with exactly the same data as in (b), but using single-winner activation and without continuation time during training.

5.6 Results

This section reports the results obtained using computer simulations, while the next section describes additional results with a physical robot. Individual map formation, overall performance, and arm trajectories are examined first in this section. Experiments are then conducted to assess robustness by using one subsystem alone, disrupting the SOMs or the arm, and varying control timing.

5.6.1 Map and Limit Cycle Formation

In the first stage of training, all four maps in the architecture are trained separately using unsupervised learning with random inputs. Figure 5.12 shows examples of how

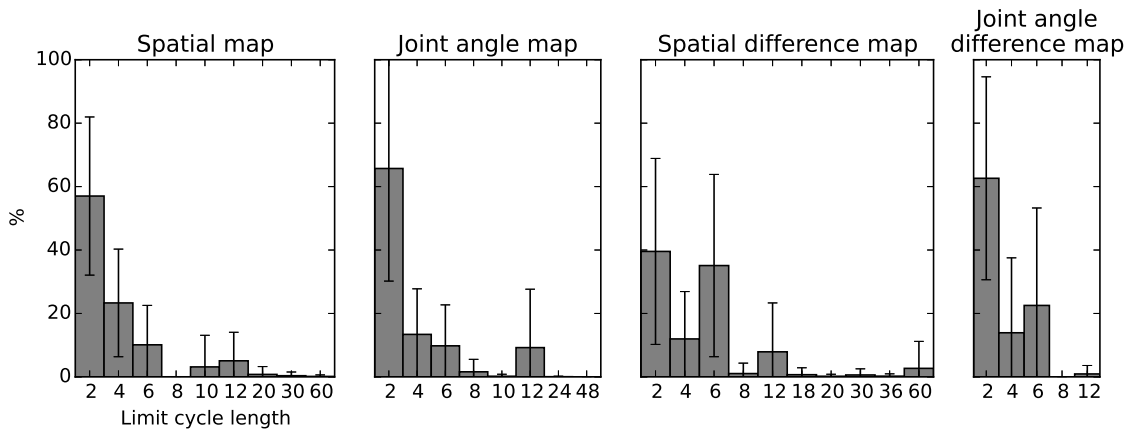

Figure 5.13: Distribution of the lengths of limit cycles representing 1000 random inputs after training. Each error bar indicates one standard deviation over the 10 independent simulations.

the maps become self-organized after training. In Figure 5.12a, the weights in the spatial difference map that correspond to input Δy (y is a Cartesian coordinate defined in Figure 4.7) are initially random. After training (Figure 5.12b), the weights become self-organized into quasi-repetitive patterns of high and low value clusters, forming dark and light interleaved stripes. In Figure 5.12c, the weights after training in the spatial map that correspond to input Cartesian coordinate y shows a similar organization as in Figure 5.12b, despite a different map size and that the input coordinates are not normalized as in the spatial difference map. Other weight organizations not shown are qualitatively similar. Although this result is less smooth than many conventional SOMs (e.g., Figure 5.12d), because multi-winner activation is used and the maps are trained for limit cycles, its appearances are quite similar to some cortical maps in biological neural systems, such as maps in cats' visual cortex (for example, see (Swindale et al., 2000, Figure 1)).

Figure 5.13 summarizes the lengths of the limit cycles formed in each map, given 1000 random inputs following training. On average, about 60% of the random inputs results in a limit cycle of length 2 in three of the maps, although there is high variation among different simulations, as indicated by the error bars (standard deviations). Other common lengths include 4, 6, 10, 12, etc. The lengths of limit cycles tend to be multiples of 2 and 3, where smaller factors tend to appear more frequently. Before the dynamics settle in a limit cycle, there are on average 6.94 (SD=2.52) time steps of irregular/aperiodic activity. Note that the architecture does not need to detect the lengths nor the onset time of limit cycles. They are shown here for illustration purposes only.

5.6.2 Arm Performance and Trajectory

As described in Section 5.5, the 1000 spatial targets used for testing are generated by sampling a grid in the 3D joint angle space. The performance against the test data can serve to indicate how well the architecture generalizes to new inputs, since test inputs are very unlikely to be in the training data, which were generated randomly. For each test target, the architecture is run for a maximum of 1000 iterations to drive the arm toward the target. Whenever the spatial error $\|\boldsymbol{X}^T - \boldsymbol{X}^M\|$ has remained consistently below a predetermined threshold $\epsilon = 0.001$ for 20 consecutive iterations, the arm is considered to have reached the target. In this case the execution is terminated early. At the end of reaching each target, the spatial error and the number of iterations required to reach termination are recorded to measure the accuracy and speed of a

Table 5.1: Summary of performance results. Numbers in parentheses indicate standard deviations.

All test data (1000 targets)	spatial error	error in cm	#iterations
pre-training	.551 (.0807)	117.1 (17.96)	1000 (0)
post-training	.011 (.0024)	2.2 (.49)	554 (21.6)
open-loop only	.083 (.0031)	16.7 (.67)	1000 (.2)
closed-loop only	.025 (.0022)	5.2 (.52)	570 (22.5)

Typical workspace (800 targets)	spatial error	error in cm	#iterations
pre-training	.566 (.0835)	119.9 (18.44)	1000 (0)
post-training	.003 (.0007)	.7 (.14)	505 (18.8)
open-loop only	.084 (.0036)	16.9 (.76)	1000 (.3)
closed-loop only	.009 (.0034)	1.9 (.74)	493 (25.4)

reaching movement. The whole workspace of the arm is linearly transformed into a unit cube, i.e., x, y, z coordinates are each within $[0, 1]$. This allows measuring spatial error relative to workspace size and independent of actual dimensions of the arm. For size reference, the extent that a Baxter robot's arm can reach, inferred from a 3D model, are roughly of length 170 cm, 235 cm, and 218 cm in the x, y, and z directions, respectively. A fully extended arm is about 137 cm long (see Figure 4.7).

The average spatial error after training is 0.011 (SD=0.0024), decreasing from 0.551 (SD=0.0807) before training (see Table 5.1). This corresponds to 1.1% of the length of each workspace axis, or 2.2 cm in a Baxter robot's physical space. The average number of iterations required is decreased to 554 (SD=21.6) from the maximum 1000 before training. Fig. 5.14 plots spatial errors for all test targets after training. Each line segment connects a spatial target and the corresponding final location of the manipulator. Therefore, shorter line segments indicate more accurate reaching. There are clearly two 3D regions where the errors are larger. A sideways view (see Figure 5.14b) reveals that these two regions are behind and below the base, and

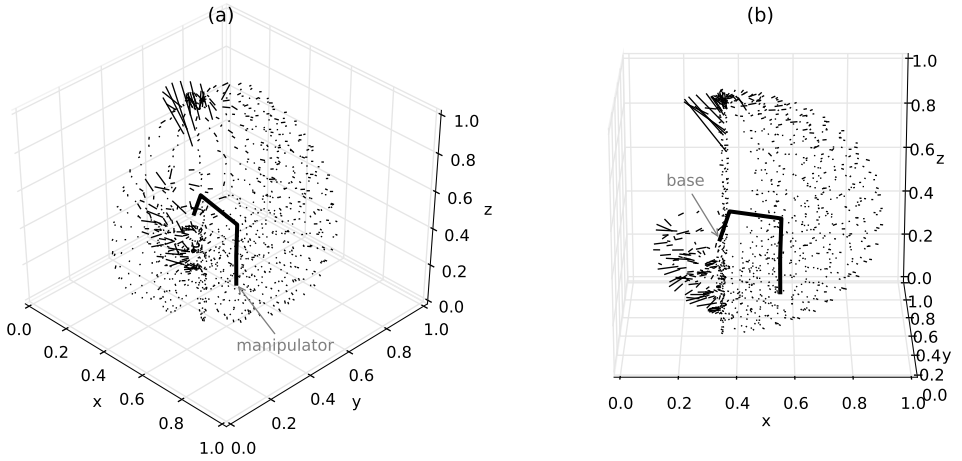

Figure 5.14: Distribution of spatial errors seen from two different view angles. Each line segment connects a target location and a corresponding final manipulator location. Longer line segments indicate greater spatial errors. The thick line segments illustrate the starting position of the arm.

near the top, which are relatively difficult to reach. Targets in these regions are likely to require difficult arm positions, i.e., they lie around "singularities", which require joint positions that make moving in certain spatial directions difficult if not impossible (Guenther and Barreca, 1997). However, these regions are usually not in the workspace in a typical robotic application. For example, unlike with the simulations done here that attempt to include all possible positions, the physical space behind and below the arm base is typically occupied by a robot's torso and thus not accessible in any event. Reaching for these extreme targets are less critical because objects to be acted on are typically placed in front of a robotic arm at a moderate height, and therefore excluding them may reflect more practical situations. Specifically, by excluding 200 targets whose x coordinates are less than that of the arm base (i.e., behind the arm base), as well as those with $z > 0.96$, the average spatial error for the remaining 800 targets in a typical workspace drops to 0.003 (SD=0.012), which

corresponds just 0.7 cm in the Baxter robot's scale.

Figure 5.15 shows a typical arm trajectory when reaching for one of the test targets. The spatial trajectory of the manipulator appears to be smooth and reasonably natural (Figure 5.15a). Figure 5.15b indicates that the spatial error decreases drastically from 0.7 to below 0.03 in early iterations (e.g., first 150 iterations), during which the open-loop subsystem dominates (refer to Figure 5.9). From there on, the closed-loop outputs start to out-weigh the open-loop ones to perform fine-tuning for the time until the target spatial error $\epsilon = 0.001$ is met. Figures 5.15c,d show the joint angles and joint velocities during reaching. The joint angle trajectory appears to be overall smooth and sigmoid-shaped, and the velocity profiles are single-peaked and near bell-shaped. This is comparable to and consistent with the results in past studies focusing on human arm movements, which suggests that joint trajectories are highly stereotyped and contain invariant spatio-temporal features including sigmoid joint displacement and bell-shaped velocity profiles (Gentili et al., 2010, 2015).

5.6.3 Standalone Open-Loop and Closed-Loop Subsystems

To better understand the individual contributions of the open-loop and the closed-loop subsystems after training, each of the subsystems is disabled in turn, forming an open-loop-only system and a closed-loop-only system. Specifically, the open-loop-only system is obtained by setting the modulatory weight functions $b^c_{open}(n) = 0$ and $b^o_{open}(n) = b^o(n) + b^c(n)$. The latter is for maintaining the same total modulatory weight as that of the full architecture, so that the results are fairly comparable.

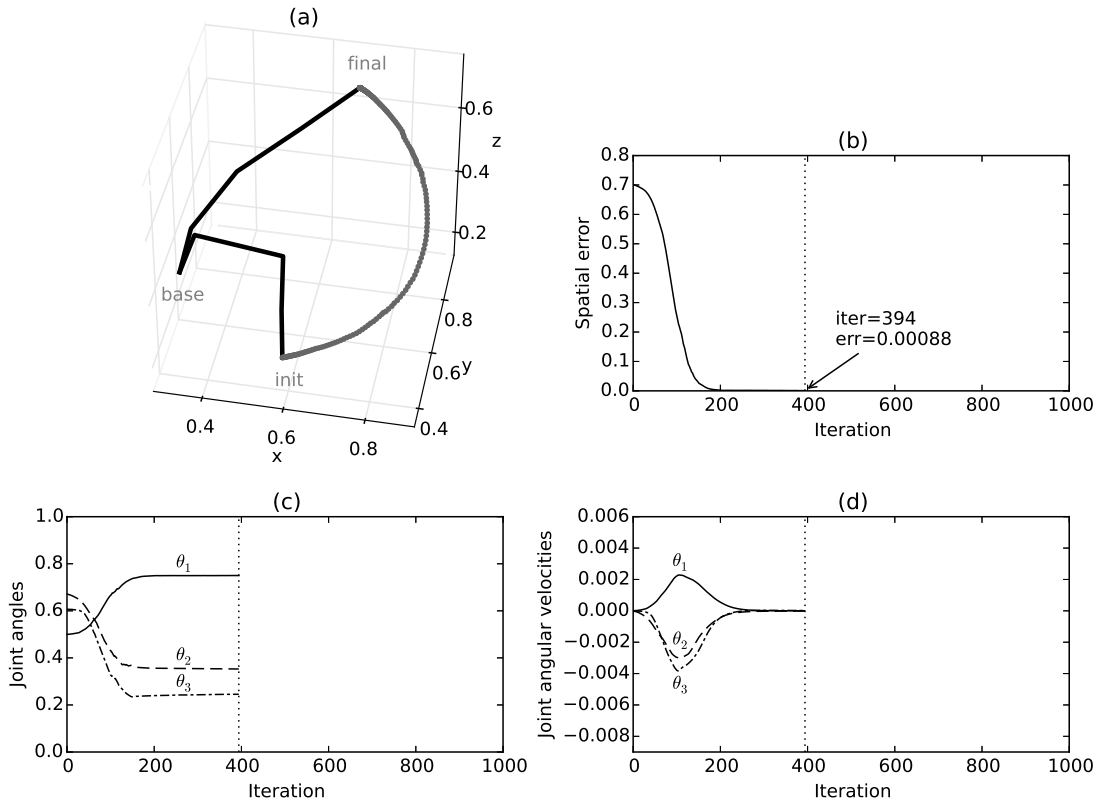

Figure 5.15: Arm trajectory for a typical reaching movement. (a) The spatial representation of the trajectory, where the initial and final arm positions are drawn using line segments. The label "base" indicates the fixed base of the arm. The labels "init" and "final" mark the initial and final manipulator locations. The manipulator location at each iteration is plotted using a dot, forming a curve connecting "init" and "final". The 3D coordinates are the result of linearly transforming each axis of the arm-reachable Cartesian space to $[0, 1]$. (b) The spatial error at each iteration. The final iteration and spatial error are labeled at the bottom right. (c) The joint angles of the arm at each iteration, where the three joints θ_1, θ_2, and θ_3 correspond to those in Fig. 4.7. The joint angles are represented by linearly transforming the whole angular range of each joint to $[0, 1]$. (d) The angular velocities of the joints at each iteration, low-pass filtered. The vertical lines in (b)–(d) indicate the terminating iteration.

Similarly for the closed-loop-only system, $b^o_{closed}(n) = 0$ and $b^c_{closed}(n) = b^o(n) + b^c(n)$. The same test data are used to compare the open-loop-only and closed-loop-only systems against the full architecture.

Since each of the three system variants may perform quite differently under different

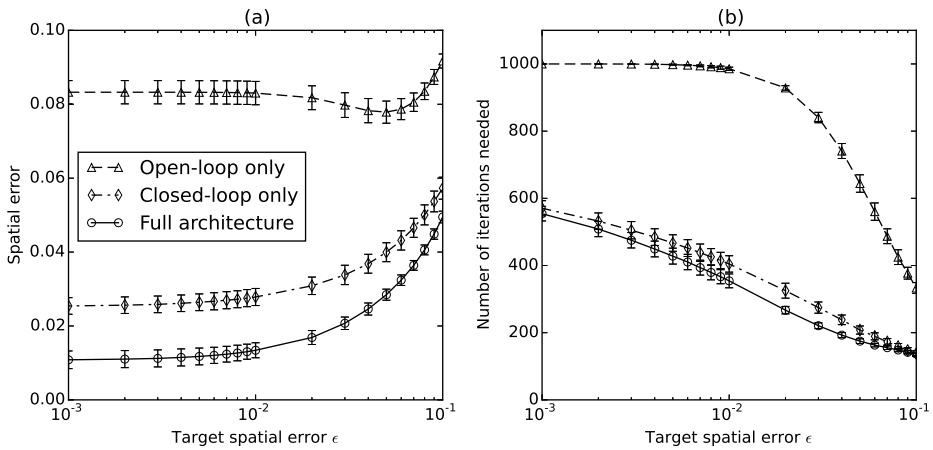

Figure 5.16: Performance comparisons among an open-loop-only system, a closed-loop-only system, and the full architecture. (a) Average actual spatial error with respect to target spatial error ϵ. (b) Average number of iterations required, upper-bounded by 1000, to reach the target spatial error. The lines are labeled in (a). Error bars indicate standard deviations.

target spatial error ϵ, a wide range of ϵ values (i.e., 0.001–0.1) is explored to determine relative performance of the systems and how it depends on ϵ. Figure 5.16 shows the performance comparisons in terms of spatial error and number of iterations with respect to varying target errors ϵ. The bottom two rows of Table 5.1 list numerical results for $\epsilon = 0.001$. The full architecture always outperforms the other two systems in terms of spatial error, indicating a clear benefit of integrating open-loop and closed-loop subsystems. In terms of number of iterations, or speed of a reaching movement, the full architecture performs significantly better than the open-loop-only system, which mostly takes the maximum number of iterations for $\epsilon < 0.01$, implying that the target errors are never met. Compared to the closed-loop-only system, the full architecture requires slightly fewer iterations, where the differences become significant starting from $\epsilon = 0.01$. However, when ϵ exceeds 0.06, the target error becomes so

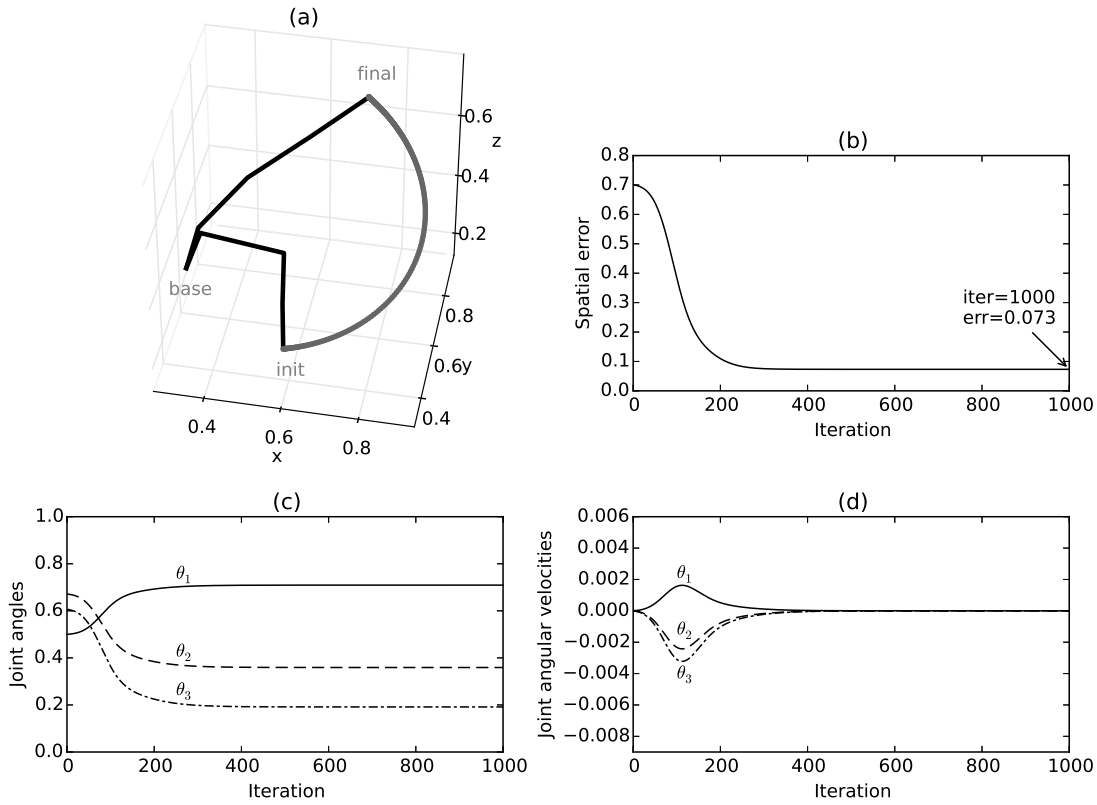

Figure 5.17: Arm trajectory using the open-loop subsystem alone. Same arrangement as in Figure 5.15. While this trajectory is marginally smoother than in Figure 5.15, the final spatial error is much worse.

large that both the full and closed-loop-only systems require very few (i.e., < 100) iterations, and the differences become insignificant. This indicates that compared to the closed-loop-only system, the full architecture can acquire targets with moderate precision (e.g., $0.01 \leq \epsilon \leq 0.06$, roughly 2–12 cm) faster.

Figure 5.17 shows the arm trajectory driven by the open-loop-only system, where the initial arm position and the spatial target are the same as those used in Figure 5.15. The arm trajectory is similar to that of the full architecture (Figure 5.15), except that the trajectory is slightly smoother and that the final spatial error 0.073 is substantially larger. The slightly smoother trajectory is a result of lacking movement regulations that

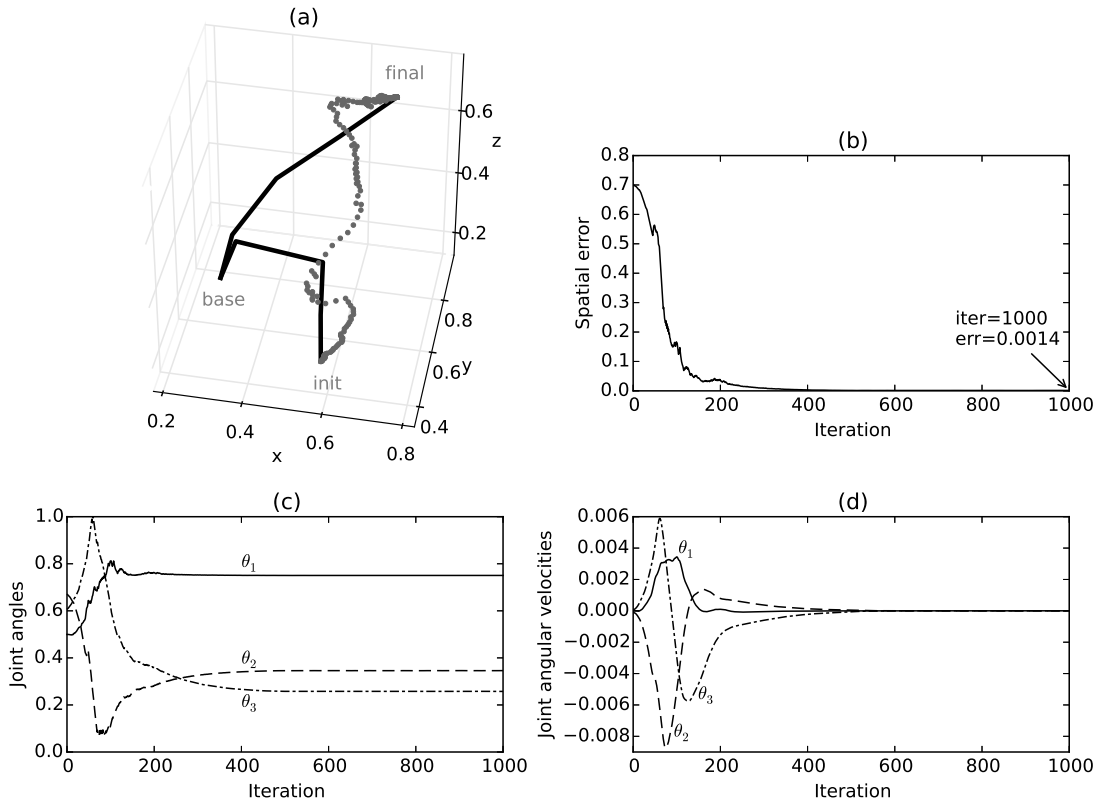

Figure 5.18: Arm trajectory using the closed-loop subsystem alone. Same arrangement as in Figure 5.15. While this trajectory has a final spatial error comparable to that of Figure 5.15, the smoothness here is much worse.

are normally generated by the closed-loop subsystem. The low accuracy is expected since the open-loop subsystem here does not have a perfect internal representation, and thus the movement is likely to be erroneous without being regulated by error feedback along the trajectory (Jordan and Wolpert, 1999). On the other hand, Figure 5.18 shows the arm trajectory driven by the closed-loop-only system, where the initial arm position and the spatial target remain the same. Although $\epsilon = 0.001$ is not met, and thus the maximum number of iterations is used, the final spatial error is acceptable. However, the spatial trajectory is not smooth. The joint angle plots (Figures 5.18c,d) also contrast with Figure 5.15, showing apparent signs of non-monotonic movement

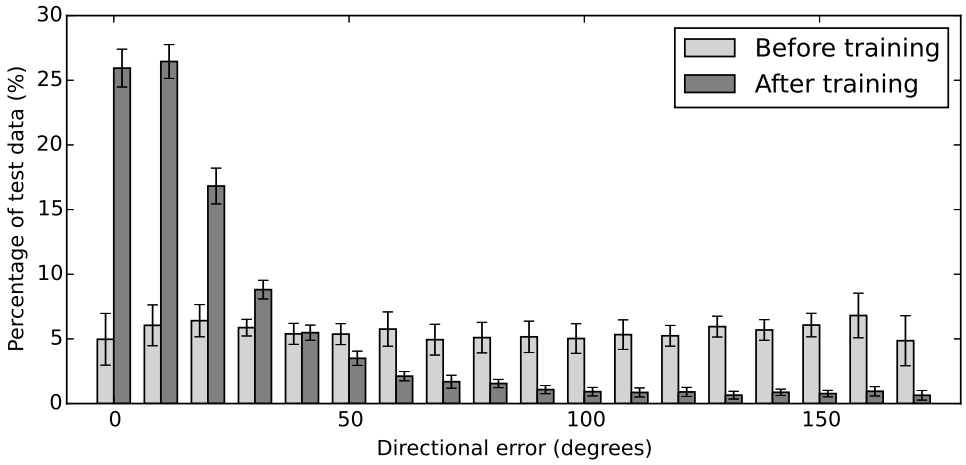

Figure 5.19: Distribution of per-iteration angular error for the closed-loop subsystem. The angular error is the angle between the spatial direction from the manipulator location towards the target location, and the spatial direction as a result of the joint command generated by the closed-loop subsystem. Each error bar indicates one standard deviation over 10 independent simulations.

towards the target. This can be observed by the peaks in the joint angle plot, as well as the velocity profiles having multiple (positive and negative) peaks. This jerky motion is likely caused by the closed-loop subsystem's constantly regulating movement to compensate for the lack of an initial "push" that is normally provided by the open-loop subsystem (Jordan and Wolpert, 1999).

The zig-zag path generated by the closed-loop-only system also indicates that the learned mapping from spatial differences to joint angle differences is not perfect. To further understand this, 1000 random spatial difference vectors (paired with random joint angles) were generated to serve as test inputs to the closed-loop subsystem. The resulting joint angle difference outputs are then converted to spatial vectors using the model arm. The angles between the resulting spatial vectors and the input spatial vectors indicate per-iteration directional errors of the closed-loop subsystem.

Figure 5.19 plots the distribution of the directional error in degrees before and after training the closed-loop-only system, averaged over 10 independent trials with different random initial weights. Before training, the angular errors are evenly distributed across the 0–180 degree range, with an average of 90.3 degrees (SD=1.2) and a median of 90.2 degrees (SD=2.6). After training, the errors are distributed mostly among small angles, with an average of 30.9 degrees (SD=1.4) and a median of 18.9 degrees (SD=0.8), substantially lower than the pre-training case but clearly not perfect. One possible source of error is that the architecture is operating on a finite number of discrete states, while space coordinates and joint angles are continuous. However, the trained closed-loop subsystem can still reach a given target eventually with high accuracy, because to do so, the only requirement is that the manipulator moves closer to the target in each iteration, which corresponds to per-iteration directional errors less than 90 degrees. The results show that per-iteration errors less than 90 degrees account for 92.4% (SD=0.9) of the test data, meaning that there is good chance that the manipulator moves closer to the target in each iteration.

To summarize, the standalone open-loop and closed-loop systems each has limitations in spatial accuracy, speed of target acquisition, and/or trajectory smoothness. In spite of this, the integrated architecture successfully learns to draw upon the strengths of the two types of control mechanisms to produce accurate, fast, and smooth trajectories, as compared with the standalone versions.

5.6.4 Performance in the Presence of Noise

Biological nervous systems routinely operate under many sources of noise. Inspired by that fact, and that the neural architecture is based on limit cycle attractor states, which were shown earlier can resist certain degrees of perturbation, this section studies how the arm control architecture responds to internal and external interference. The question being asked here is not only how the architecture performs under different levels of interference, but also what parts of the architecture are more susceptible to such interference. To this end, three types of internal and external disruption are introduced: transient perturbation to activity in each of the four maps, permanently disabling, or "damaging" map nodes, and sudden perturbations of arm positions that simulate unexpected external forces.

First, I study the effects of transient activity noise in the architecture by temporarily perturbing activity in each map (spatial, joint angle, spatial difference, joint angle difference maps), and observing how perturbation to each of these maps affects spatial error. When a map activity pattern is being perturbed, the activity of each node is given a probability to flip from 0 to 1 or 1 to 0 (i.e., making active nodes inactive and vice versa), where the probability of node activity flipping is a parameter. Perturbation to a map is temporary, meaning that for each iteration, perturbation occurs only at a predetermined time step. For the spatial and spatial difference maps, the perturbation time is set to be at $t = 30$ for two reasons. One is that since these maps start receiving inputs at $t = 0$, by $t = 30$ their activity is very likely to have entered a limit cycle. The other reason is that by $t = t^{assoc} = 50$, their downstream maps start to read from

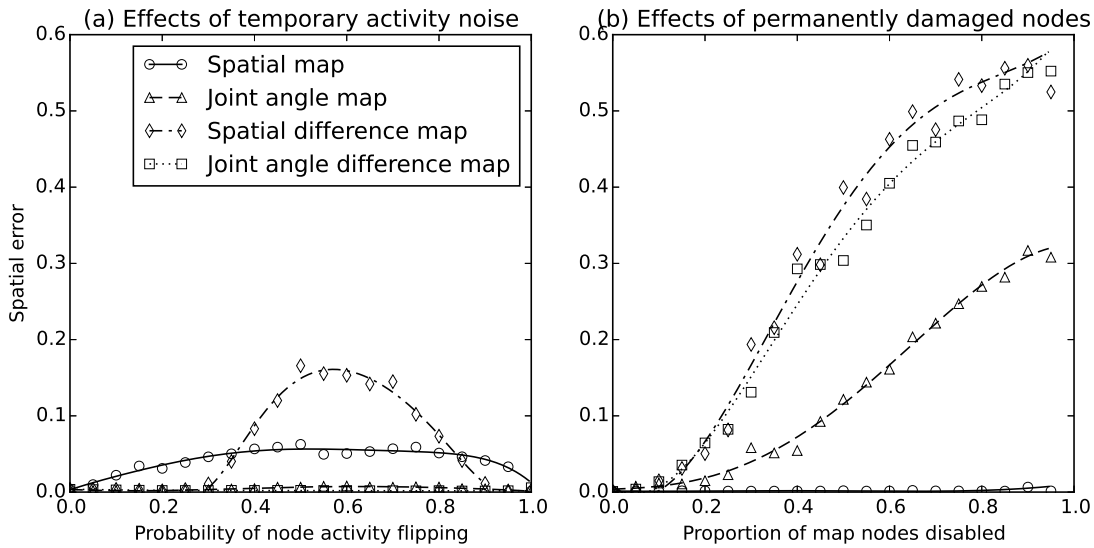

Figure 5.20: Effects of internal interference. (a) Spatial error as a result of perturbing activity in each map. The horizontal axis represents the probability that node activity is flipped from 0 to 1 or 1 to 0 during perturbation. (b) Spatial error as a result of permanently disabling map nodes, using the same notations as in (a). In both figures, each data point indicates the average of 100 simulations of random activity perturbation or random selections of disabled nodes. Curves are polynomially fitted to the data points to aid interpretation. Variances (omitted for better readability) are generally small and they tend to be larger for larger mean spatial errors.

them, so this allows them 20 time steps to recover from noise. For similar reasons, the perturbation time for the joint angle and the joint angle difference maps is set to be $t = 80$, which is 30 time steps after they start receiving inputs.

Figure 5.20a shows the spatial errors as a result of perturbing different maps with different activity flipping probabilities, where the initial arm position and the spatial target are the same as those used in Figure 5.15. Each data point in the figure is the average of 100 independent simulations with different random seeds for perturbation. Polynomial curves are fitted to the data points to highlight the trends. Overall, activity noise in the spatial and spatial difference maps causes much greater

impact to the performance, compared with noise in the joint angle and joint angle difference maps. A reason for this is that the latter receive inputs continuously from their upstream maps, which helps mitigate their own activity noise, while the former receive inputs only briefly. For all maps, the spatial error tends to be greater as the activity flipping probability is in the mid-range, when the uncertainty about a node's activity is the highest. Note that very high probability of activity flipping also decreases spatial error. This can be understood as follows. Consider an extreme case where the activity flipping probability is 1, and where all winner nodes take on value 0 and non-winner nodes value 1. Since topographical recurrent connections of each node covers its local neighborhood except for itself, in the next time step after activity flipping, a previous winner node is guaranteed to receive all 1 recurrent inputs, while a previous non-winner node may receive some 0 recurrent inputs for the presence of winner nodes in its neighborhood (which is very likely). As a result, a winner node in an inverted activity pattern will almost always win in the next time step because all recurrent weights are positive. This cancels the perturbation. Another observation from Figure 5.20a is that even a small activity flipping probability in the spatial map is enough to cause the spatial error to rise. The maximum spatial error it causes is limited at around 0.05. On the contrary, spatial error is rather insensitive to small activity flipping probability in the spatial difference map, but once the probability exceeds around 0.3, spatial error rises more rapidly and may reach above 0.15, which is poor but not devastating compared with pre-training error around 0.55 (Table 5.1). Perturbation to the joint angle map and the joint angle difference map have much less effects on spatial error. A reason is that they receive inputs continuously from their

upstream maps, which help mitigate their activity noise, while the other two maps receive inputs only briefly.

Next, consider the impact of permanently disabling map nodes. This alternative simulates damaged neurons in a cortical region. A disabled map node always has a zero activity level, and does not compete with its neighbor nodes during the multi-winners-take-all process. Compared with activity perturbation, which is a transient effect, disabled map nodes remain so in all time steps throughout a reaching movement, and thus cause a much larger disruption to the architecture. In this experiment, the initial arm position as well as the target are again the same as used in Figure 5.15. Figure 5.20b summarizes the results of disabling different proportions of map nodes, where each data point is the average of 100 simulations, each having different random map nodes disabled. Disabling nodes in the spatial difference map and the joint angle difference map is the most detrimental, causing the spatial error to rise rapidly above 0.1 (a borderline acceptable value) when 30% of the nodes or more are disabled. Both of these maps belong to the closed-loop subsystem that is more essential to spatial accuracy. Note that since the joint angle map also participates in generating closed-loop outputs, disabling nodes in the map also cause the spatial error to rise, although less rapidly. The error rises above 0.1 only when 50% of the joint angle map nodes or more are disabled. Disabling nodes in the spatial map has little effect on spatial errors, since the spatial map is related to open-loop outputs only, and as discussed in Sect. 5.6.3 (see also Table 5.1), the architecture can still achieve high accuracy without open-loop outputs, although this also leads to jerky trajectories.

Arm trajectories as a result of an impaired open-loop or closed-loop subsystem can

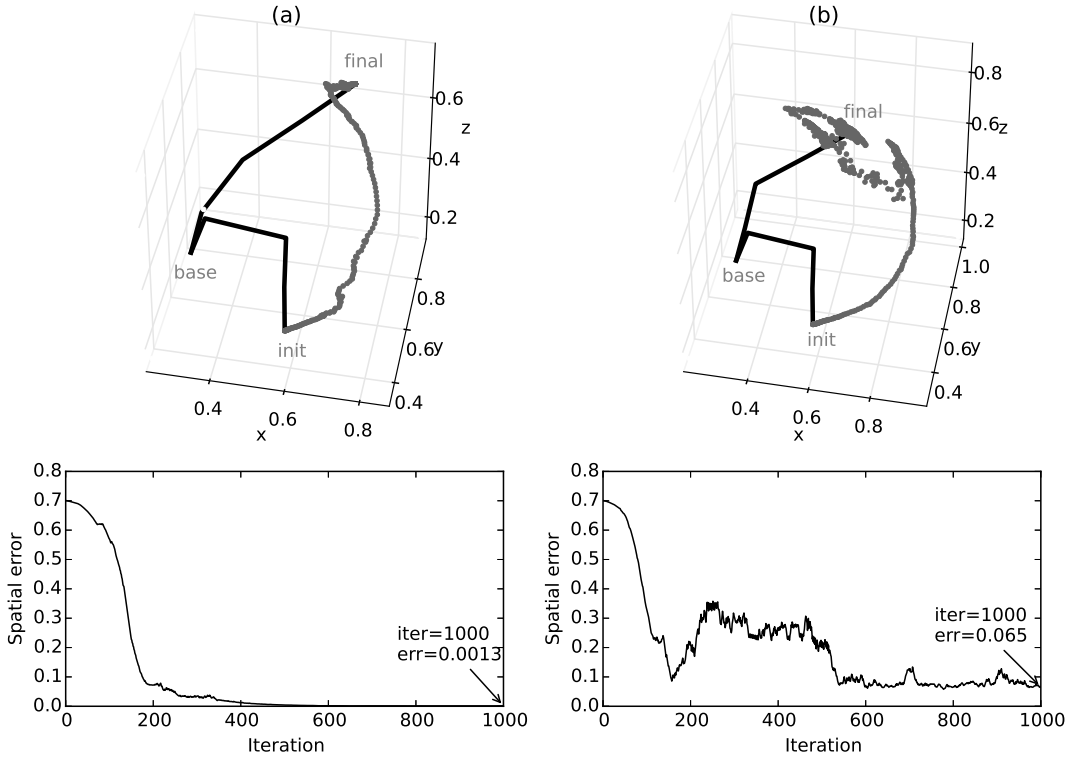

Figure 5.21: Arm trajectories when 30% of random nodes are disabled in the (a) spatial or (b) spatial difference map. They each indicate how the neural architecture performs with an impaired open-loop or closed-loop subsystem, respectively. The top row shows the spatial trajectories of the manipulator, and the bottom row shows the spatial error at each iteration.

be illustrated by permanently damaging 30% of the spatial map nodes or 30% of the spatial difference map nodes, respectively. The resulting arm trajectories are shown in Fig. 5.21, where the initial arm position and the spatial target are kept the same as used in Fig. 5.15. When the open-loop subsystem alone is impaired, the spatial trajectory appears to be somewhat S-shaped, indicating signs of overshooting. The spatial error decreases relatively slowly in the first 100 iterations, during which the movement is primarily guided by the open-loop subsystem. The intact closed-loop subsystem then takes over, and the error eventually drops to a relatively low value. On the other hand, when the closed-loop subsystem alone is impaired, the spatial trajectory is smooth in early iterations, and then the manipulator starts to exhibit a "tremor" towards the end of the trajectory. The spatial error in the first 100 iterations decreases quickly (at a similar rate to Fig. 5.15) since the manipulator is guided primarily by the intact open-loop subsystem. Then the error starts to oscillate and remains at relatively high values.

Finally, I study the effects of applying abrupt external forces to the arm, which is simulated by adding a random perturbation vector to the arm joints at a given iteration. With the angular range of each joint being normalized to be in $[0, 1]$, the length of each random vector is fixed at 0.1, or 10% of each axis length in the joint angle space. The question being asked here is whether the arm can resume reaching while maintaining a low spatial error in the presence of such an external perturbation. The initial arm position and the target are again made the same as in Figure 5.15, such that the results shown in Figure 5.15 can serve as a control for comparison. Figure 5.22 shows the results of applying external perturbations at different iterations during a

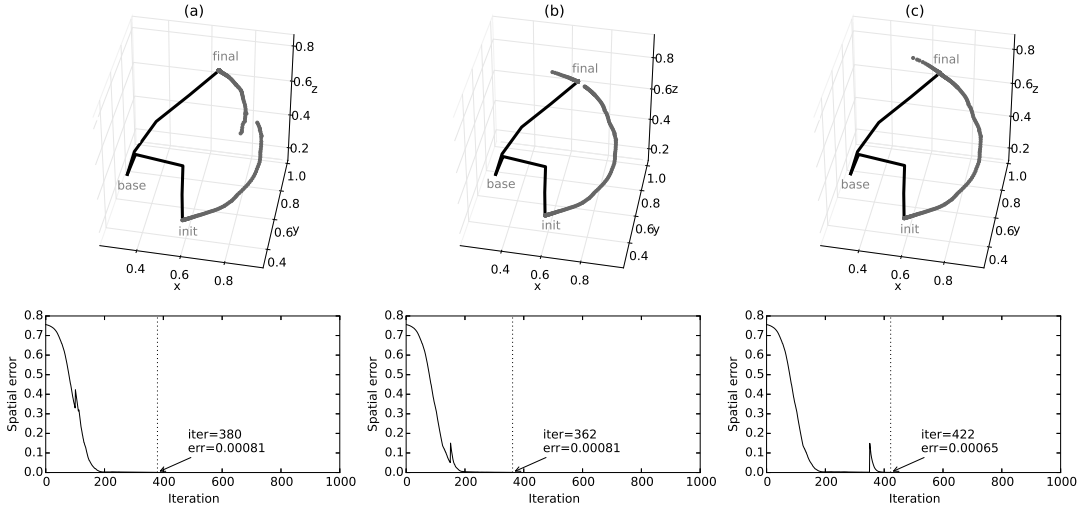

Figure 5.22: Arm trajectories for abrupt arm joint perturbation during a reaching movement. Columns (a), (b), and (c) show the results of perturbing at iterations 100, 150, and 350, respectively. The top row plots the spatial trajectories of the manipulator. The discontinuities of the trajectories indicate where the perturbations occur. The bottom row shows the spatial error in each iteration.

reaching movement: (a) at iteration 100 when the open-loop subsystem dominates, (b) at iteration 150 when both subsystems are weighed about equally, and (c) at iteration 350 when the closed-loop subsystem dominates. From the spatial trajectory (top row), the exerted perturbations do not greatly affect the smoothness of the trajectory. From the spatial error plots (bottom row), as soon as a perturbation occurs, the spatial error can be quickly reduced to a normal level within a few tens of iterations. The final spatial error and the number of iterations spent are not substantially affected, either. This result suggests robustness of the neural architecture against unexpected external perturbations, regardless of when they occur.

5.6.5 Sensitivity to Timing Parameters

Internal timing of a neural architecture refers here to the times at which its neural components or connections are enabled or disabled. In other words, internal timing controls and coordinates the sequences of activations/deactivations of the components of the architecture, and thus controls the flow of neural activity to achieve certain computations. The complexity of computing feasible timing, as well as the precision requirements for executing the timing, directly contributes to the control overhead incurred by a timing control mechanism (not neurally modeled in this study). It is therefore preferable that internal timing of an architecture be made simple and forgiving, which means that the architecture can tolerate inaccurate timing.

The internal timing of the architecture is fairly simple, containing only three timing parameters: t^{assoc}, t^{out}, and t^{filter}. The value of t^{assoc} defines the time when the associative connections between the spatial map and the joint angle map, as well as those between the spatial difference map and the joint angle difference map, are enabled. The value of t^{out} defines the time when the outputs ($\mathbf{\Theta}^o$ and $\mathbf{\Delta\Theta}^c$) are generated. The value of t^{filter} defines the length of the time window for the temporal average filters. This subsection studies how sensitive the spatial error is to the values of the timing parameters. Forgiving internal timing should accept a relatively wide range of timing parameter values without the spatial error significantly increasing.

In this experiment, the values of t^{assoc}, t^{out}, and t^{filter} are varied while the initial arm position and the spatial target are kept the same as used in Figure 5.15. The baseline values are: $t^{assoc} = 50$, $t^{out} = 130$, and $t^{filter} = 30$. The effects of varying the

three parameters are shown in Figure 5.23, where each data point is the average of 10 independent simulations with randomly initialized architectures. The spatial error fluctuates but stays essentially at the same level while changing t^{assoc} over a rather wide range. The value of t^{out} also has little effect on the spatial error when it is greater than or equal to 80, which is t^{filter} time steps after t^{assoc}. Generating output earlier than this time of course results in poor accuracy either because the neural activity has not been propagated to the joint angle map and the joint angle difference map, or because the temporal average filters have not seen t^{filter} activity patterns. The effects of t^{filter} indicate that as long as the temporal average filter takes an average over more than one activity pattern, the spatial error becomes steady. In summary, the architecture is quite tolerant of changes in control parameter values. The only restrictions about the internal timing of the architecture are: (1) $t^{out} \geq t^{assoc} + t^{filter}$ and (2) $t^{filter} > 1$, which are quite forgiving. More importantly, these timing parameters do not depend on the length or the onset of individual limit cycles, and therefore online limit cycle detection is not necessary.

5.7 Physical Robot Implementation Results

For validation purposes and for assessing the applicability of the neural architecture beyond the simulated arm in SMILE, its use in controlling the movements of a physical Baxter robot's arm is studied. The goal here is very limited: just to demonstrate that the neural architecture described and trained using a simulated robot as above, could be ported to and run effectively on a physical robot *without any further training.*

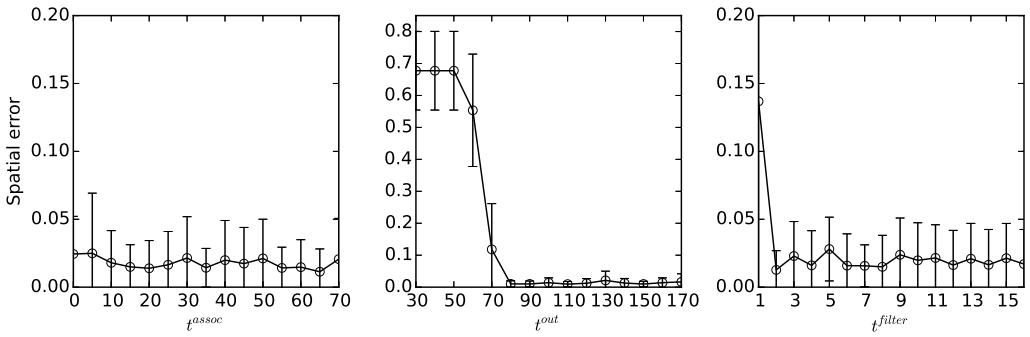

Figure 5.23: Effects of timing parameter values t^{assoc}, t^{out}, and t^{filter} on spatial error. Each error bar indicates one standard deviation over the 10 independent simulations. Note the different scales on the vertical axes. The t^{filter} range shown in the right-most figure is up to 16 (instead of 30) because further increasing t^{filter} does not lead to substantial changes of spatial error.

Two shoulder joints, S0 (roll) and S1 (pitch), and an elbow joint, E1 (pitch), of the robot's left arm are respectively mapped to $\theta_1, \theta_2, \theta_3$ of the neural architecture, while other joints are fixed at 0, forming a 3-DOF arm as schematized in Figure 4.7 and studied in the simulations above. A commodity workstation is used to host both the neural architecture and Robot Operating System (ROS) (Quigley et al., 2009), which serves as a communication medium between the neural control architecture and the robot. Through ROS, the neural architecture obtains joint angles and issues joint velocity commands at a target polling rate of 60 Hz. While the robot is equipped with cameras, since the interest in this study is not machine vision, these cameras are not used to determine spatial locations. Instead, the target location is specified and input manually, and the manipulator location is computed based on joint angles (reported by the robot's sensors) using the Denavit-Hartenberg method (Hartenberg and Denavit, 1965, p.435). The neural architecture is trained using data generated by the ideal arm model (Section 4.3.1), and is then used to control the robotic arm

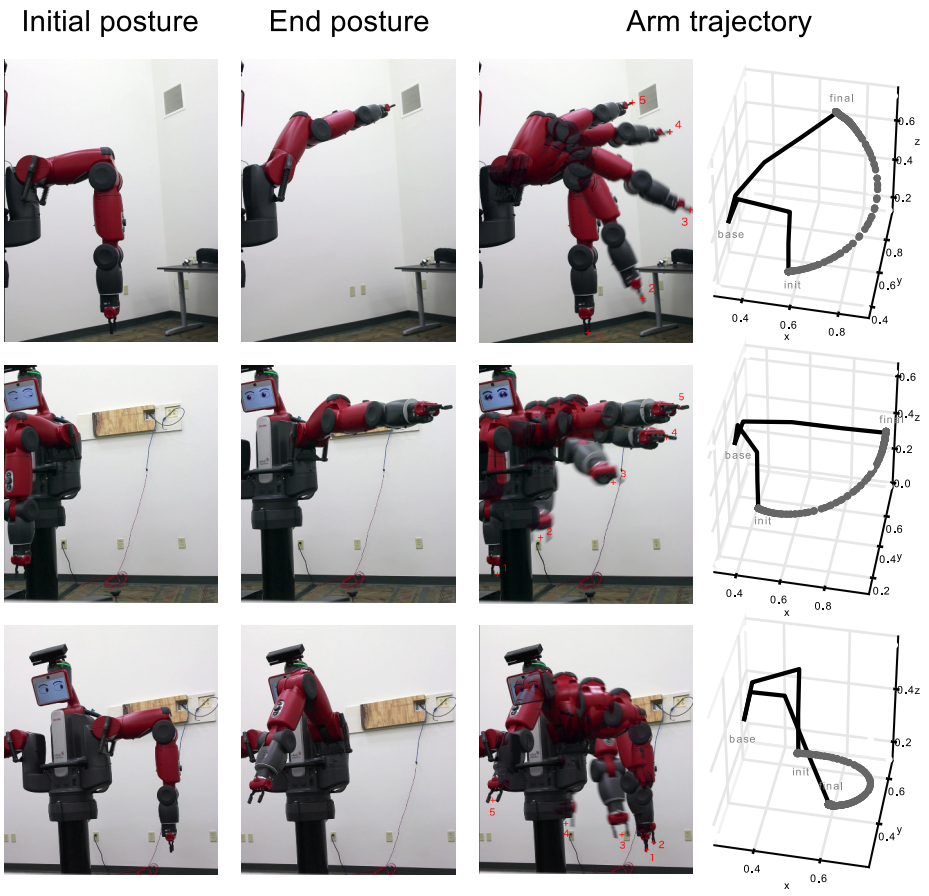

Figure 5.24: Three trajectories of a physical robotic arm. Each row shows the results of a different pair of initial arm position and spatial target. The first two columns show the initial and final arm positions of the robot. The third column shows superimposed images illustrating the trajectory of the robot arm. The right-most column plots the manipulator locations using joint angle sensor history recorded from the robot in a fashion like the earlier figures in this paper.

without further training on the robot itself. Unlike the arm model, the mechanical components and sensors of the robotic arm may introduce errors and latencies in measuring the joint angles and in executing joint velocity commands.

Figure 5.24 shows results of the robot arm performing three reaching movements. The initial arm postures and the spatial targets of the movements are distinct for each movement. The first two movements are center-out reaching, where the arm

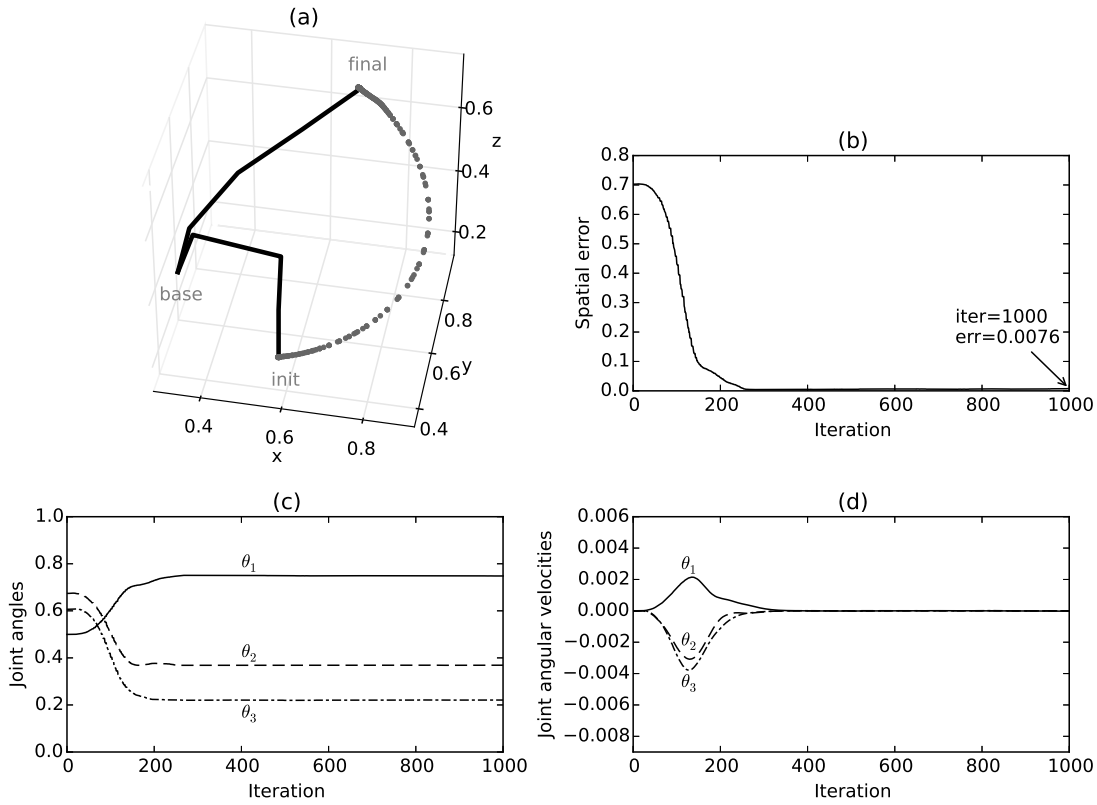

Figure 5.25: Arm trajectory of the robot. The initial arm position and the spatial target are the same as those used in Figure 5.15 as well as in Figure 5.24a. Figures (a)–(d) are again arranged in the same way as in Figure 5.15. Data are recorded from the joint angle sensors of the robot.

starts at positions close to the torso and reaches out to an above-shoulder target and a shoulder-height target. The third movement starts from one side of the torso and ends in the front. The trajectories of manipulator locations show smooth curved paths from the starting points to the targets. The final spatial errors for the three movements are 0.008 (1.3 cm), 0.002 (0.3 cm), and 0.001 (0.1 cm).* Since the first two movements do not satisfy the stopping criteria $\epsilon = 0.001$, they both take the maximum of 1000 iterations. The third movement takes 899 iterations. While this performance is not

*Since these spatial errors are computed based on the joint angles, their accuracy depends on the accuracy of the joint sensors.

as good as running on the simulated ideal arm model, it is generally acceptable for typical robotic applications, and could potentially be improved with further training on the physical robot.

Figure 5.25 shows the detailed joint angle and velocity trace of the first arm movement, which uses the same initial position and the same spatial target as the experiments shown in Figure 5.15. The trajectory is quite qualitatively similar to Figure 5.15, showing a sigmoid-shaped joint angle trace and a single-peaked, close to bell-shaped, velocity profile, although slightly not as smooth. In summary, these results indicate that the neural architecture, *despite being trained with an ideal arm model*, can successfully operate a Baxter robot's arm without further training.

5.8 Summary and Discussion

In this chapter, a neural architecture was created for integrated open-loop and closed-loop arm reaching movements based on limit cycle SOMs. These SOMs adopt limit cycle dynamics of sparsely-coded multi-focal activation states, something reminiscent of rhythmically oscillating biological cortical activity in a limited sense. This type of activity, however, is much harder to harness compared with a static representation. With such oscillatory activity, reaching and maintaining a fixed spatial location with an arm is also challenging, not to mention the potential risk that the architecture may need a much more complicated control mechanism to time each neural component properly.

The neural architecture proposed here contains four maps representing spatial

coordinates, spatial differences, joint angles, and joint angle differences, which together realize an integrated open-loop and closed-loop system. A three-stage learning process was introduced to train the maps and connections between them, using both unsupervised and supervised methods. It was found that in spite of the oscillatory nature of the map layers' activity patterns, well-formed maps self-organized during learning, just as they do with more standard non-oscillatory SOMs. The results indicate that the neural architecture is able to generalize and produce commands that control an arm to reach and maintain spatial targets reasonably fast and with low spatial errors, while maintaining smooth and reasonably natural arm trajectories.

In the architecture, outputs of the open-loop and closed-loop subsystems are weighted such that their relative influence is progressively shifted from the open-loop to the closed-loop subsystem during a movement. This is reminiscent of the human motor control system that combines two control modes: feedforward and feedback (Jordan and Wolpert, 1999; Kawato et al., 1987; Sainburg et al., 1999). The feedforward mode is particularly critical in the early stages (up to ∼50–100 ms) of a movement since the sensory feedback from the periphery cannot be fully processed and thus employed by the supra-spinal control system to regulate the movement (Kurtzer, 2015). Conversely, the closed-loop mode is largely employed during the later stages of movement where the sensory feedback has time to be processed, and this allows for movement regulation by supra-spinal control centers for dealing with perturbations and/or accuracy constraints on the final position (beyond ∼50–100 ms) (Kurtzer, 2015). It must be noted that although computational neural models combining both feedforward (open-loop) and feedback (closed-loop) control have been proposed in the

past, these do not employ oscillatory SOMs as in this study.

Examination of the neural architecture in more detail characterized many desirable aspects of its behavior. It was shown that when either the open-loop or the closed-loop subsystem operated alone, its performance decreased in either case. Further, the impact of individual maps on the overall performance was assessed by applying activity noise to the maps or permanently disabling map nodes. Arm joints were also abruptly perturbed to see how the architecture reacts. In general, the architecture proved to be very robust in that it maintained reasonably good performance with all of these different types of internal or external interference, and even when one of the two subsystems was shut down. Internal timing complexity is another concern for an architecture based on dynamic activity. The neural architecture depends on only three timing parameters, and their values can be fixed quite arbitrarily and do not need to correspond to the length/onset of individual limit cycles. Consequently, although the architecture is based on limit cycle activity, explicit detection of limit cycles is not necessary for it to successfully control an arm movement. Finally, when the architecture was ported to a physical robot, it was able to control the robot arm reasonably well without further training, in spite of the inevitable mechanical latency and imprecision this produced.

Although the created architecture is by no means a model of any part of the brain, the results of lesions in the open-loop or closed-loop subsystem are somewhat comparable to neurophysiological observations. For example, it has been argued that biases in biological "internal models", which purportedly guide movements without sensory feedback, result in overshooting or undershooting of the movements (Bhanpuri

et al., 2014; Manto, 2009). This roughly corresponds to the situation where the open-loop subsystem in the architecture was partially damaged, and this resulted in a clear undershooting early in the trajectory. On the other hand, the closed-loop subsystem plays a role similar to that of the primate cerebellum, which is related to integrating feedback sensory information. For example, cooling of the interpositus nucleus in monkey cerebellum, which is related to proprioceptive sensory feedback, results in tremor (Manto, 2009; Vilis and Hore, 1980), and human "intention tremor" (occurs when an extremity approaches the endpoint of a visually guided movement) is strongly related to cerebellar disease (Ropper and Samuels, 2009). In the results reported above, lesions in the closed-loop subsystem led to a tremor as the manipulator approaches the target, generating something reminiscent of intention tremor seen in humans. Such post-damage results are particularly compelling support that the limit cycle SOM architecture captures important aspects of biological motor control functionality, in that they were neither intentionally programmed into the architecture or even anticipated by the author prior to examining the experimental results. Of course, the architecture is currently limited in that it does not handle constraints such as the presence of obstacles or posture requirements; this could be an important issue for future work.

— Chapter 6 —

Discussion and Conclusion

This chapter concludes this dissertation by first summarizing the work reported, followed by current limitations and possible future work. Finally, the contributions that this work made are highlighted.

6.1 Summary

Many past neural architectures that can be structurally related to multi-region brain models tend to use a group of unconnected nodes (i.e., a "layer") to loosely represent a cortical region. On the other hand, neuroanatomical structures of a brain region generally contain topographical lateral connections among cortical columns. From this perspective, the limit cycle SOMs studied in this work have great potential to replace unconnected layers in neural architectures, since they provide a computationally efficient abstraction of topographical lateral connections using one-step competitive activation and learning, and they have once and again been proven to be both computationally useful and able to account for various biological cortical phenomena. Unfortunately, to date the use of SOMs of any kind in creating neural architectures is quite limited due to limitations of conventional SOMs.

A primary limitation that this dissertation addresses is the past use of static single-winner representations in conventional SOMs. This means that external stimuli are represented by a SOM using a single fixed activity state usually containing a single winner. This traditional approach not only encodes information in an inefficient fashion, but also results in generally less robust systems. A reason for the latter is that using a static representation ignores ongoing neural dynamics that potentially result in changing activity states. In fact, as an enormous amount of neuro-physiological data indicates, biological neural systems are unlikely to operate based solely on static or fixed states. They are highly rhythmic and oscillatory, a quality that is rarely accounted for in past work on brain-inspired neural architectures involving SOMs. Additionally, many past architectures perform computation only for the duration that inputs are present. This is in contrast to biological nervous systems in which sustained neural responses to transient stimuli are a common phenomenon, and this sustained activity is likely to serve as short-term (working) memory that supports ongoing neurocomputations without persistence of the original stimuli. Therefore, to address these issues, the research done in this dissertation work has focused on building robust multi-region neural architectures using SOMs based on limit cycle activity that is reminiscent of ongoing oscillatory activity of the brain.

As a first step, a conventional SOM model was augmented in Chapter 3 to exhibit multi-focal activity dynamics by introducing a multi-winners-take-all process, locally recurrent connections, and activation/training continuation. A systematic search of parameter values uncovered a parameter regime in which limit cycle activity attractors reliably emerged in ongoing activity dynamics of the SOM that exhibits cyclically

repeating, multi-focal activity patterns. These limit cycle attractors were learned through self-organization, and were shown to be able to encode both static and temporal sequence inputs, the features of which can be either binary of real-valued. Compared with static representations and other types of attractors, it was found that limit cycle attractors were the most stable (i.e., perturbation-resistant) representations in a SOM, especially when each map node was not recurrently connected to itself. It was also found that each learned limit cycle is more unique than other types of representations, making it easier for subsequent processing to distinguish different limit cycles. Not only did each limit cycle become more distinct from one another, but the similarity or distance between limit cycles was found to preserve the similarity between their corresponding inputs. This included inputs that were never used to train the limit cycle SOM, indicating that the SOM generalized to new stimuli, a quality that is crucial to any neurocomputational models. Equally crucial is the formation of topographic maps in limit cycle SOMs. It was found that in spite of using different activation and learning rules from conventional SOMs, topographic maps can be formed robustly in a limit cycle SOM. While the resulting map is certainly not as smooth as in conventional SOMs, its quasi-repetitive patterns are reminiscent of biological cortical maps such as those in cat visual cortex.

Having identified and analyzed limit cycle representations in individual SOMs, the next step taken was to learn associations between different SOMs to retrieve those limit cycle representations, since association is a common principle that is needed for many neural architectures to work. Chapter 4 studied three situations where associations were learned. In the first situation, a static pattern in one SOM

was shown to successfully retrieve a whole limit cycle in another SOM, even in the presence of noise. Retrieval of limit cycles was more robust than those of other types of representations. In the second situation, a limit cycle in one SOM was shown to be successfully retrieved using a limit cycle in another SOM through associative connections. While the first two situations were studied in the context of image-to-phoneme-sequence mapping using relatively naive learning rules, the third situation dealt with an open-loop arm control problem that requires associating limit cycles for continuous spaces. With enhanced learning rules, associative connections were shown to generalize to new inputs in the third situation. A common property in all three situations is that, despite continually changing activity in SOMs, exact timing is not important for learning or activating limit cycle associations. For example, training data were typically prepared by sampling activity at arbitrary time steps without considering the boundaries of limit cycle attractors, and sometimes even activity states outside of a limit cycle were used for training (first situation). Similarly, times to activate learned associations are quite free, and were fixed arbitrarily in the experiments with little regards to limit cycle timings. This eliminates the need to explicitly detect limit cycles when learning/activating associations, making a neural architecture temporally robust without the need for high-precision timing control.

Finally, a neural architecture based on limit cycle SOMs was created for a practical robotic application in Chapter 5. The goal was to control a robotic arm to reach a *fixed* spatial location through a smooth trajectory and *hold it there*, in spite of oscillatory activity in the neural architecture. This is an attempt to address the issue of whether oscillatory multi-SOM architectures can solve general non-oscillating tasks,

something that would be expected because biological nervous systems apparently can. A neural architecture containing four limit cycle SOMs was created for this purpose. The architecture consists of an open-loop and a closed-loop subsystems. Whereas open-loop outputs are weighted more in early stages of an arm movement, closed-loop outputs progressively take over in later stages and fine tune the manipulator location to reach a spatial target. The results showed that the spatial error was quite low and the arm trajectory was smooth. It was also found that the architecture is quite robust. In response to a variety of ways to disrupt/damage it, the architecture's performance was impaired in some cases but not completely destroyed. For example, when gate timings was varied, the overall performance was not significantly affected. To further validate its practicality, this neural architecture was ported to control a physical robotic arm, and it worked quite successfully *without further training* with the physical robot. More interestingly, some biologically observed behaviors could be recreated with the architecture, although the architecture was not intentionally built to be neuro-anatomically accurate or programmed to exhibit those behaviors. For example, when the closed-loop subsystem was partially damaged, something reminiscent of intention tremor in human patients occurred. In summary, all of these results suggest that neural architectures based on limit cycle SOMs have the potential to be practical and robust, and to be able to account for more biological phenomena than conventional SOMs do at both the neural activity level and the behavior level.

6.2 Limitations and Future Work

The first limitation is that, while the issue of generalization has been addressed throughout this dissertation, consideration of this issue was largely limited in static data. How and whether a limit cycle SOM generalizes to new temporal sequence inputs appear to be a much harder problem, since the set of all possible temporal sequences forms a much larger space than with static data. For example, temporal sequence inputs may be of different lengths. Each pattern in a temporal sequence may also appear in many other sequences, and it may also occur multiple times in each sequence. Although some past studies on SOMs partially addressed this issue, they generally use single-winner static representations. Therefore, it would be very useful in the future to determine how limit cycle activity generalizes to new temporal sequences. The phoneme sequence data used in this research is apparently insufficient. To study generalization, a much larger training set with real-valued features might be used, such as spatial trajectories of arm movements.

Another limitation is that, although limit cycles in a SOM were self-organized to encode external stimuli, they were not self-organized to encode limit cycles in other SOMs in the same architecture (although associations were learned between limit cycles). It is therefore unknown how a limit cycle SOM can be used to represent activity in other limit cycle SOMs. In past studies, single-winner static SOMs have been arranged in a structure where a "higher-level" SOM takes input from one or multiple "lower-level" SOMs. This seems more difficult for limit cycle SOMs, since each SOM's activity becomes a temporal sequence of multi-focal patterns. My hypothesis is that,

assuming limit cycle SOMs generalize to new temporal sequence inputs, higher-level limit cycle SOMs can develop more abstract representations than lower-level SOMs do, and this eventually will help accomplish more complex input pattern recognition.

Finally, while the robotic arm control neural architecture developed in this work (Chapter 5) was found to be accurate, fast, and robust, a limitation is that it does not deal with posture constraints or obstacle avoidance. For example, in a pick-and-place task where an object on a tabletop is to be picked up, the manipulator of a robotic arm must not point upwards. In the implementation done in this research, there is no way to specify this constraint. Further, the limit-cycle SOM architecture currently only controls a 3-DOF arm. How it would work if scaled up to control a full 7-DOF arm, or even to coordinate two arms at the same time, is unknown. Finally, how oscillatory movements may be generated is not clear either. Intuitively this appears to be relatively achievable given the oscillatory internal activity associated with limit cycle SOMs, and that there are many past central pattern generator neural models that might be modified and adapted to this purpose. This would be useful to explore since the results could be applied to robotic tasks such as hammering a nail, tightening a screw, etc. In general, these future goals will not only make the neural controller more practical, but may also shed significant light on how oscillatory activity can be used for more general purpose neurocomputation.

6.3 Summary of Contributions

Past SOMs use single-winner static representations to encode information while largely ignoring ongoing neural dynamics. Even in the rare exceptions where post-stimuli neural activity does persist in a SOM, it is typically a behavior built into individual map nodes (e.g., using phase oscillators), and it is unclear how this continuing activity could be used for neurocomputations. This work for the first time studies SOMs that represent information using self-organized limit cycle dynamics of activity states. Following this idea, specific research questions raised include: How and under what conditions (e.g., learning rules and parameter values) can limit cycle attractors emerge through self-organization using generic non-oscillating map nodes? Do topographic maps like those in conventional SOMs still form? How can these limit cycle attractors be used to encode information? What are their properties, such as their abilities to resist perturbations, to become more distinguishable, to represent input spaces well, and to generalize to new stimuli? Can limit cycle representations be associated to form a basis for further neurocomputations? Can an intrinsically oscillatory neural architecture composed of limit cycle SOMs be built for a practical fixed-point application? If so, what are the properties of the architecture, such as how robust it is?

Along the way to seeking answers to these questions, the following contributions were made to the field:

- Developing limit cycle SOMs. This first contribution is showing that information can indeed be encoded using limit cycle attractors of activity states in SOMs.

This includes developing learning rules and systematically exploring parameter regime for reliably acquiring limit cycle attractors, as well as studying their properties to serve as representations such as stability, uniqueness, and generalization. Different types of inputs are investigated, including both binary and real-valued features, and both static and temporal sequence inputs. Map formation is reliably observed under all of these conditions.

- Learning associations for retrieving limit cycle representations. Since learning associations is the foundation of many neurocomputations, this second contribution is the exploration of ways to associate a limit cycle with another SOM's (static or limit cycle) activity, such that the limit cycle can be retrieved without the original afferent stimuli that it encodes being present. This is achieved by considering different connectivity between SOMs (single-route and dual-route) and different learning rules to train associative connections. The learned associations were shown to generalize well, and timings to train/activate the associations were shown to be forgiving.

- Building a limit-cycle SOM neural architecture for a practical robotic application. The third contribution is the construction and examination of a multi-SOM neural architecture using limit cycle activity for a general, fixed-point task: stable arm reaching. The challenge was to create an architecture that is able to control an arm through a smooth trajectory to arrive at and remain at a fixed location with high spatial accuracy, in spite of its continually oscillating activity. It was found that this could be achieved and that the architecture based on

limit cycle activity was quite robust in a variety of ways, and that sometimes its behaviors are reminiscent of those of humans. The neural architecture trained on a simulated arm was successfully ported to a physical robot without further training.

While this work was originally inspired by brain oscillations, it is currently unclear to what extent the brain operates based on limit cycle attractors. However, the limit-cycle SOM architectures studied in this work clearly showed that robust neurocomputations can be accomplished using non-fixed-point activity states, and that it is possible to simultaneously account for more biological phenomena than past SOM architectures, including neural oscillations, multi-focal activity patterns, quasi-repetitive map formation, and persistent neural responses to transient stimuli.

Bibliography

Abbott, A. (2013). Solving the brain. *Nature*, 499(7458):272–274.

Angulo, V. and Torras, C. (2008). Learning inverse kinematics: Reduced sampling through decomposition into virtual robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(6):1571–1577.

Bação, F., Lobo, V., and Painho, M. (2005). The self-organizing map, the Geo-SOM, and relevant variants for geosciences. *Computers & Geosciences*, 31(2):155–163.

Barreto, G., Araújo, A., and Ritter, H. (2003). Self-organizing feature maps for modeling and control of robotic manipulators. *Journal of Intelligent and Robotic Systems*, 36(4):407–450.

Bednar, J. and Miikkulainen, R. (2000). Tilt aftereffects in a self-organizing model of the primary visual cortex. *Neural Computation*, 12(7):1721–1740.

Bengio, Y. and Lee, H., editors (2015). *Deep Learning of Representations. Neural Networks*, 64.

Bhanpuri, N., Okamura, A., and Bastian, A. (2014). Predicting and correcting ataxia using a model of cerebellar function. *Brain*, 137(7):1931–1944.

Bi, G.-Q. and Poo, M.-M. (1998). Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of Neuroscience*, 18(24):10464–10472.

Bouvier, G., Desdouits, N., Ferber, M., Blondel, A., and Nilges, M. (2015). An automatic tool to analyze and cluster macromolecular conformations based on self-organizing maps. *Bioinformatics*, 31(9):1490–1492.

Bullock, D., Grossberg, S., and Guenther, F. (1993). A self-organizing neural model of motor equivalent reaching and tool use by a multijoint arm. *Journal of Cognitive Neuroscience*, 5(4):408–435.

Buzsaki, G. (2006). *Rhythms of the Brain*. Oxford University Press.

Carpinteiro, O. (1999). A hierarchical self-organizing map model for sequence recognition. *Neural Processing Letters*, 9(3):209–220.

Chang, F.-J., Chang, L.-C., Kao, H.-S., and Wu, G.-R. (2010). Assessing the effort of meteorological variables for evaporation estimation by self-organizing map neural network. *Journal of Hydrology*, 384(1–2):118–129.

Chappell, G. and Taylor, J. (1993). The temporal Kohønen map. *Neural Networks*, 6(3):441–445.

Chattopadhyay, M., Dan, P. K., and Mazumdar, S. (2014). Comparison of visualization of optimal clustering using self-organizing map and growing hierarchical self-organizing map in cellular manufacturing system. *Applied Soft Computing*, 22:528–543.

Chaumette, F. and Hutchinson, S. (2006). Visual servo control. I. Basic approaches. *IEEE Robotics Automation Magazine*, 13(4):82–90.

Chen, N., Ribeiro, B., Vieira, A., and Chen, A. (2013). Clustering and visualization of bankruptcy trajectory using self-organizing map. *Expert Systems with Applications*, 40(1):385–393.

Chen, Y. and Reggia, J. (1996). Alignment of coexisting cortical maps in a motor control model. *Neural Computation*, 8(4):731–755.

Chow, T. W. S. and Rahman, M. K. M. (2009). Multilayer SOM with tree-structured data for efficient document retrieval and plagiarism detection. *IEEE Transactions on Neural Networks*, 20(9):1385 –1402.

Colomé, A. and Torras, C. (2015). Closed-loop inverse kinematics for redundant robots: Comparative assessment and two enhancements. *IEEE/ASME Transactions on Mechatronics*, 20(2):944–955.

Coltheart, M., Rastle, K., Perry, C., Langdon, R., and Ziegler, J. (2001). DRC: A dual route cascaded model of visual word recognition and reading aloud. *Psychological Review*, 108(1):204–256.

Deboeck, G. and Kohonen, T. (2013). *Visual Explorations in Finance: with Self-Organizing Maps*. Springer.

Delgado, S., Morán, F., Mora, A., Merelo, J., and Briones, C. (2015). A novel representation of genomic sequences for taxonomic clustering and visualization by means of self-organizing maps. *Bioinformatics*, 31(5):736–744.

Eliasmith, C., Stewart, T., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., and Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179 – 211.

Euliano, N. and Principe, J. (1999). A spatio-temporal memory based on SOMs with activity diffusion. In Oja, E. and Kaski, S., editors, *Kohonen Maps*, pages 253–266. Elsevier.

Fankhauser, N. and Mäser, P. (2005). Identification of GPI anchor attachment signals by a Kohonen self-organizing map. *Bioinformatics*, 21(9):1846–1852.

Feldman, D. (2012). The spike-timing dependence of plasticity. *Neuron*, 75(4):556–571.

Fell, J. and Axmacher, N. (2011). The role of phase synchronization in memory processes. *Nature Reviews Neuroscience*, 12(2):105–118.

Finnerty, G., Shadlen, M., Jazayeri, M., Nobre, A., and Buonomano, D. (2015). Time in cortical circuits. *The Journal of Neuroscience*, 35(41):13912–13916.

Fisher, R. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188.

Flash, T., Meirovitch, Y., and Barliya, A. (2013). Models of human movement: Trajectory planning and inverse kinematics studies. *Robotics and Autonomous Systems*, 61(4):330–339.

Furukawa, T. (2009). SOM of SOMs. *Neural Networks*, 22(4):463–478.

Garis, H., Shuo, C., Goertzel, B., and Ruiting, L. (2010). A world survey of artificial brain projects, part I: Large-scale brain simulations. *Neurocomputing*, 74(1–3):3–29.

Gaskett, C. and Cheng, G. (2003). Online learning of a motor map for humanoid robot reaching. In *International Conference on Computational Intelligence, Robotics and Autonomous Systems*.

Gentili, R., Han, C., Schweighofer, N., and Papaxanthis, C. (2010). Motor learning without doing: Trial-by-trial improvement in motor performance during mental training. *Journal of Neurophysiology*, 104(2):774–783.

Gentili, R., Oh, H., Huang, D.-W., Katz, G., Miller, R., and Reggia, J. (2015). A neural architecture for performing actual and mentally simulated movements during self-intended and observed bimanual arm reaching movements. *International Journal of Social Robotics*, 7(3):371–392.

Gorricha, J. and Lobo, V. (2012). Improvements on the visualization of clusters in geo-referenced data using self-organizing maps. *Computers & Geosciences*, 43:177–186.

Grajski, K. and Merzenich, M. (1990). Hebb-type dynamics is sufficient to account for the inverse magnification rule in cortical somatotopy. *Neural Computation*, 2(1):71–84.

Guenther, F. and Barreca, D. (1997). Neural models for flexible control of redundant systems. In Morasso, P. and Sanguineti, V., editors, *Self-organization, Computational Maps, and Motor Control*, pages 383–421. North-Holland.

Guimarães, G. (2000). Temporal knowledge discovery for multivariate time series with enhanced self-organizing maps. In *International Joint Conference on Neural Networks*, volume 6, pages 165–170.

Guimarães, G., Lobo, V., and Moura-Pires, F. (2003). A taxonomy of self-organizing maps for temporal sequence processing. *Intelligent Data Analysis*, 7(4):269–290.

Gustafsson, L. and Paplinski, A. (2006). Bimodal integration of phonemes and letters: An application of multimodal self-organizing networks. In *International Joint Conference on Neural Networks*, pages 312–318.

Hagenbuchner, M., Sperduti, A., and Tsoi, A. (2003). A self-organizing map for adaptive processing of structured data. *IEEE Transactions on Neural Networks*, 14(3):491–505.

Hammer, B., Micheli, A., Neubauer, N., Sperduti, A., and Strickert, M. (2005). Self organizing maps for time series. In *Workshop on Self-Organizing Maps*, pages 115–122.

Hartenberg, R. and Denavit, J. (1965). *Kinematic synthesis of linkages*. McGraw-Hill.

Henriques, R., Lobo, V., and Bação, F. (2012). Spatial clustering using hierarchical SOM. In Johnsson, M., editor, *Applications of Self-Organizing Maps*, chapter 12. InTech.

Horio, K. and Yamakawa, T. (2001). Feedback self-organizing map and its applicaiton to spatio-temporal pattern classification. *International Journal of Computational Intelligence and Applications*, 1(1):1–18.

Hsu, C.-C. and Lin, S.-H. (2012). Visualized analysis of mixed numeric and categorical data via extended self-organizing map. *IEEE Transactions on Neural Networks and Learning Systems*, 23(1):72–86.

Hua, H. (2016). Image and geometry processing with oriented and scalable map. *Neural Networks*, 77:1–6.

Huang, D.-W., Katz, G., Langsfeld, J., Gentili, R., and Reggia, J. (2015a). A virtual demonstrator environment for robot imitation learning. In *IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*.

Huang, D.-W., Katz, G., Langsfeld, J., Oh, H., Gentili, R., and Reggia, J. (2015b). An object-centric paradigm for robot programming by demonstration. In Schmorrow, D. and Fidopiastis, C., editors, *Foundations of Augmented Cognition*, pages 745–756. Springer.

Huang, J. and Hagiwara, M. (1997). A multi-winners self-organizing neural network. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 2499–2504.

Hutchinson, S., Hager, G., and Corke, P. (1996). A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670.

Iwasaki, T. and Furukawa, T. (2016). Tensor SOM and tensor GTM: Nonlinear tensor analysis by topographic mappings. *Neural Networks*, 77:107–125.

James, D. and Miikkulainen, R. (1995). SARDNET: A self-organizing feature map for sequences. In *Conference on Neural Information Processing Systems*, volume 7, pages 577–584.

Johnsson, M. and Balkenius, C. (2008). Recognizing texture and hardness by touch. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 482–487.

Johnsson, M., Balkenius, C., and Hesslow, G. (2011). Multimodal system based on self-organizing maps. In Madani, K., Correia, A., Rosa, A., and Filipe, J., editors, *Computational Intelligence*, volume 343 of *Studies in Computational Intelligence*, pages 251–263. Springer.

Jordan, M. and Wolpert, D. (1999). Computational motor control. In Gazzaniga, M., editor, *The Cognitive Neurosciences*. MIT Press.

Kaipainen, M. and Ilmonen, T. (2003). Period detection and representation by recurrent oscillatory self-organizing map. *Neurocomputing*, 55(3–4):699–710.

Kajić, I., Schillaci, G., Bodiroža, S., and Hafner, V. (2014). Learning hand-eye coordination for a humanoid robot using SOMs. In *ACM/IEEE International Conference on Human-Robot Interaction*, pages 192–193.

Kalteh, A., Hjorth, P., and Berndtsson, R. (2008). Review of the self-organizing map (SOM) approach in water resources: Analysis, modelling and application. *Environmental Modelling & Software*, 23(7):835–845.

Kangas, J. (1990). Time-delayed self-organizing maps. In *International Joint Conference on Neural Networks*, volume 2, pages 331–336.

Kaski, S., Kangas, J., and Kohonen, T. (1998). Bibliography of self-organizing map (SOM) papers: 1981-1997. *Neural Computing Surveys*, 1(3–4):1–176.

Kawato, M., Furukawa, K., and Suzuki, R. (1987). A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, 57(3):169–185.

Kayacik, H., Zincir-Heywood, A., and Heywood, M. (2003). On the capability of an SOM based intrusion detection system. In *International Joint Conference on Neural Networks*, volume 3, pages 1808–1813.

Khouzam, B. and Frezza-Buet, H. (2013). Distributed recurrent self-organization for tracking the state of non-stationary partially observable dynamical systems. *Biologically Inspired Cognitive Architectures*, 3:87–104.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69.

Kohonen, T. (1988). The 'neural' phonetic typewriter. *Computer*, 21(3):11–22.

Kohonen, T. (1991). The hypermap architecture. In *International Conference on Artificial Neural Networks*, volume 2, pages 1357–1360.

Kohonen, T. (1997). Exploration of very large databases by self-organizing maps. In *International Conference on Neural Networks*, volume 1, pages PL1–PL6.

Kohonen, T. (2001). *Self-organizing maps*, volume 30 of *Springer Series in Information Sciences*. Springer.

Kohonen, T. (2013). Essentials of the self-organizing map. *Neural Networks*, 37:52–65.

Kohonen, T., Oja, E., Simula, O., Visa, A., and Kangas, J. (1996). Engineering applications of the self-organizing map. *Proceedings of the IEEE*, 84(10):1358–1384.

Koikkalainen, P. and Horppu, I. (2007). Handling missing data with the tree-structured self-organizing map. In *International Joint Conference on Neural Networks*, pages 2289–2294.

Koskela, T., Varsta, M., Heikkonen, J., and Kaski, K. (1998). Temporal sequence processing using recurrent SOM. In *International Conference on Knowledge-Based Intelligent Electronic Systems*, volume 1, pages 290–297.

Kumar, P. and Behera, L. (2010). Visual servoing of redundant manipulator with Jacobian matrix estimation using self-organizing map. *Robotics and Autonomous Systems*, 58(8):978–990.

Kurtzer, I. (2015). Long-latency reflexes account for limb biomechanics through several supraspinal pathways. *Frontiers in Integrative Neuroscience*, 8:99.

Lallee, S. and Dominey, P. (2013). Multi-modal convergence maps: From body schema and self-representation to mental imagery. *Adaptive Behavior*, 21(4):274–285.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Leinonen, L., Hiltunen, T., Torkkola, K., and Kangas, J. (1993). Self-organized acoustic feature map in detection of fricative-vowel coarticulation. *Journal of the Acoustical Society of America*, 93(6):3468–3474.

Li, P., Zhao, X., and MacWhinney, B. (2007). Dynamic self-organization and early lexical development in children. *Cognitive Science*, 31(4):581–612.

Lichodzijewski, P., Nur Zincir-Heywood, A., and Heywood, M. (2002). Host-based intrusion detection using self-organizing maps. In *International Joint Conference on Neural Networks*, volume 2, pages 1714–1719.

Liu, N., Wang, J., and Gong, Y. (2015). Deep self-organizing map for visual classification. In *International Joint Conference on Neural Networks*.

Liu, Y. and Weisberg, R. (2011). A review of self-organizing map applications in meteorology and oceanography. In Mwasiagi, J., editor, *Self-Organizing Maps - Applications and Novel Algorithm Design*, chapter 13, pages 253–272. InTech.

Louis, P., Seret, A., and Baesens, B. (2013). Financial efficiency and social impact of microfinance institutions using self-organizing maps. *World Development*, 46:197–210.

Malsburg, C. (1973). Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14(2):85–100.

Manto, M. (2009). Mechanisms of human cerebellar dysmetria: experimental evidence and current conceptual bases. *Journal of NeuroEngineering and Rehabilitation*, 6:10.

Manukyan, N., Eppstein, M., and Rizzo, D. (2012). Data-driven cluster reinforcement and visualization in sparsely-matched self-organizing maps. *IEEE Transactions on Neural Networks and Learning Systems*, 23(5):846–852.

Markram, H., Lübke, J., Frotscher, M., and Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275(5297):213–215.

Martinetz, T., Ritter, H., and Schulten, K. (1990). Three-dimensional neural net for learning visuomotor coordination of a robot arm. *IEEE Transactions on Neural Networks*, 1(1):131–136.

McQueen, T., Hopgood, A., Tepper, J., and Allen, T. (2004). A recurrent self-organizing map for temporal sequence processing. In Lotfi, A. and Garibaldi, J., editors, *Applications and Science in Soft Computing*, volume 24 of *Advances in Intelligent and Soft Computing*, pages 3–8. Springer.

Ménard, O. and Frezza-Buet, H. (2005). Model of multi-modal cortical processing: Coherent learning in self-organizing modules. *Neural Networks*, 18(5-6):646–655.

Miikkulainen, R., Bednar, J., Choe, Y., and Sirosh, J. (2005). *Computational maps in the visual cortex*. Springer.

Mohan, V., Morasso, P., Sandini, G., and Kasderidis, S. (2013). Inference through embodied simulation in cognitive robots. *Cognitive Computation*, 5(3):355–382.

Mohebi, E. and Bagirov, A. (2014). A convolutional recursive modified self organizing map for handwritten digits recognition. *Neural Networks*, 60:104–118.

Monner, D. and Reggia, J. (2009). An unsupervised learning method for representing simple sentences. In *International Joint Conference on Neural Networks*, pages 2133–2140.

Monner, D. and Reggia, J. (2012). Emergent latent symbol systems in recurrent neural networks. *Connection Science*, 24(4):193–225.

Morse, A., de Greeff, J., Belpeame, T., and Cangelosi, A. (2010). Epigenetic robotics architecture (ERA). *IEEE Transactions on Autonomous Mental Development*, 2(4):325–339.

Niedermeyer, E. and Silva, F. (2005). *Electroencephalography: Basic Principles, Clinical Applications, and Related Fields*. Lippincott Williams & Wilkins.

Nori, F., Natale, L., Sandini, G., and Metta, G. (2007). Autonomous learning of 3D reaching in a humanoid robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1142–1147.

Oja, M., Kaski, S., and Kohonen, T. (2003). Bibliography of self-organizing map (SOM) papers: 1998-2001 addendum. *Neural Computing Surveys*, 3(1):1–156.

Petreska, B. and Billard, A. (2006). A neurocomputational model of an imitation deficit following brain lesion. In *International Conference on Artificial Neural Networks*, volume 4131 of *Lecture Notes in Computer Science*, pages 770–779. Springer.

Pilly, P. and Grossberg, S. (2012). How do spatial learning and memory occur in the brain? Coordinated learning of entorhinal grid cells and hippocampal place cells. *Journal of Cognitive Neuroscience*, 24(5):1031–1054.

Pilly, P. and Grossberg, S. (2013). Spiking neurons in a hierarchical self-organizing map model can learn to develop spatial and temporal properties of entorhinal grid cells and hippocampal place cells. *PLoS ONE*, 8(4):e60599.

Pöllä, M., Honkela, T., and Kohonen, T. (2009). Bibliography of self-organizing map (SOM) papers: 2002-2005 addendum. http://lib.tkk.fi/Reports/2009/isbn9789522482532.pdf.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. (2009). ROS: An open-source robot operating system. In *IEEE International Conference on Robotics and Automation, Workshop on Open Source Software*.

Rauber, A., Merkl, D., and Dittenbach, M. (2002). The growing hierarchical self-organizing map: Exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, 13(6):1331–1341.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *International Conference on Neural Networks*, volume 1, pages 586–591.

Ropper, A. and Samuels, M. (2009). *Adams and Victor's Principles of Neurology*. McGraw-Hill.

Rumbell, T., Denham, S., and Wennekers, T. (2014). A spiking self-organizing map combining stdp, oscillations, and continuous learning. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):894–907.

Sainburg, R., Ghez, C., and Kalakanis, D. (1999). Intersegmental dynamics are controlled by sequential anticipatory, error correction, and postural mechanisms. *Journal of Neurophysiology*, 81(3):1045–1056.

Salhi, M., Arous, N., and Ellouze, N. (2009). Principal temporal extensions of SOM: Overview. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 2(3):121–143.

Sarlin, P. (2013). Self-organizing time map: An abstraction of temporal multivariate patterns. *Neurocomputing*, 99:496–508.

Saxon, J. and Mukerjee, A. (1990). Learning the motion map of a robot arm with neural networks. In *International Joint Conference on Neural Networks*, volume 2, pages 777–782.

Schulz, R. and Reggia, J. (2004). Temporally asymmetric learning supports sequence processing in multi-winner self-organizing maps. *Neural Computation*, 16(3):535–561.

Schulz, R. and Reggia, J. (2005). Mirror symmetric topographic maps can arise from activity-dependent synaptic changes. *Neural Computation*, 17(5):1059–1083.

Shanmuganathan, S., Sallis, P., and Buckeridge, J. (2006). Self-organising map methods in integrated modelling of environmental and economic systems. *Environmental Modelling & Software*, 21(9):1247 – 1256.

Strickert, M. and Hammer, B. (2005). Merge SOM for temporal data. *Neurocomputing*, 64:39–71.

Sutton, G., Reggia, J., Armentrout, S., and D'Autrechy, C. (1994). Cortical map reorganization as a competitive process. *Neural Computation*, 6(1):1–13.

Swindale, N., Shoham, D., Grinvald, A., Bonhoeffer, T., and Hubener, M. (2000). Visual cortex maps are optimized for uniform coverage. *Nature Neuroscience*, 3(8):822–826.

Sylvester, J. and Reggia, J. (2009). Plasticity-induced symmetry relationships between adjacent self-organizing topographic maps. *Neural Computation*, 21(12):3429–3443.

Székely, G., Rizzo, M., and Bakirov, N. (2007). Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769–2794.

Tarek, B., Najet, A., and Noureddine, E. (2014). Hierarchical speech recognition system using MFCC feature extraction and dynamic spiking RSOM. In *IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*.

Ultsch, A. (1993). Self-organizing neural networks for visualisation and classification. In Opitz, O., Lausen, B., and Klar, R., editors, *Information and Classification*, Studies in Classification, Data Analysis and Knowledge Organization, pages 307–313. Springer.

Varstal, M., Millán, J., and Heikkonen, J. (1997). A recurrent self-organizing map for temporal sequence processing. In *International Conference on Artificial Neural Networks*, pages 421–426.

Vilis, T. and Hore, J. (1980). Central neural mechanisms contributing to cerebellar tremor produced by limb perturbations. *Journal of Neurophysiology*, 43(2):279–291.

Voegtlin, T. (2002). Recursive self-organizing maps. *Neural Networks*, 15(8–9):979–991.

Wakuya, H. and Terada, A. (2009). Temporal signal processing by feedback SOM: An application to on-line character recognition task. In *International Conference on Neural Information Processing, Part II*, volume 5864 of *Lecture Notes in Computer Science*, pages 865–873. Springer.

Walter, J. and Ritter, H. (1996). Rapid learning with parametrized self-organizing maps. *Neurocomputing*, 12(2–3):131–153.

Wan, W. and Fraser, D. (1999). Multisource data fusion with multiple self-organizing maps. *IEEE Transactions on Geoscience and Remote Sensing*, 37(3):1344–1349.

Weems, S. and Reggia, J. (2006). Simulating single word processing in the classic aphasia syndromes based on the Wernicke–Lichtheim–Geschwind theory. *Brain and Language*, 98(3):291–309.

Wiemer, J. (2003). The time-organized map algorithm: Extending the self-organizing map to spatiotemporal signals. *Neural Computation*, 15(5):1143–1171.

Winder, R. and Reggia, J. (2012). The role of working memory in an urban pursuit scenario. In Adami, C., Bryson, D., Ofria, C., and Pennock, R., editors, *Artificial Life*, volume 13, pages 291–298. MIT Press.

Xie, X. and Seung, H. (2003). Equivalence of backpropagation and contrastive Hebbian learning in a layered network. *Neural Computation*, 15(2):441–454.

Yoshikawa, Y., Koga, J., Asada, M., and Hosoda, K. (2003). Primary vowel imitation between agents with different articulation parameters by parrot-like teaching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 149–154.

Zhang, L., Tao, H., Holt, C., Harris, W., and Poo, M.-M. (1998). A critical window for cooperation and competition among developing retinotectal synapses. *Nature*, 395(6697):37–44.

Zhao, Z., Zhang, X., and Fang, Y. (2015). Stacked multilayer self-organizing map for background modeling. *IEEE Transactions on Image Processing*, 24(9):2841–2850.