A Computational Model for Episodic Memory Inspired by the Brain

by

Adam Peter Trischler

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of University of Toronto Institute for Aerospace Studies
University of Toronto

© Copyright 2016 by Adam Peter Trischler
Abstract

A Computational Model for Episodic Memory Inspired by the Brain

Adam Peter Trischler
Doctor of Philosophy
Graduate Department of University of Toronto Institute for Aerospace Studies
University of Toronto
2016

Memory is a pillar of intelligence, and to think like us, it may be that artificial systems must remember like us. This dissertation introduces a computational model for episodic memory that is inspired by functions of the hippocampus and its interaction with the cortex. The model has two distinct components: a recurrent neural network in which experienced episodes are represented by attracting dynamical trajectories, and a deep autoencoder that both compresses high-dimensional sensory inputs and reconstructs sensory experiences from lower-dimensional stored trajectories. The model is founded on neuroscientific research which suggests that the hippocampus acts as an associative memory system, and that hippocampal traces are unfolded in the cortex. In order to realize associative memory function, a new algorithm for the construction of recurrent neural networks has been developed. This algorithm allows recurrent networks to be trained to approximate, arbitrarily well, any prescribed dynamical system. Additionally, we present a method by which arbitrary dynamical systems of attractors can be constructed by smoothing collections of vector fields. Together, these two methods provide a mathematically rigorous, end-to-end pipeline for the construction of complex attractor networks. The power of this pipeline is demonstrated with a host of examples. Our episodic memory model is demonstrated through the storage and retrieval of movie-like visual episodes. These episodes are retrieved automatically from partial inputs that represent related or analogous experiences.
Dedication

For my Father
who is nowhere
but everywhere
Acknowledgements

First I must thank my advisor, Gabriele D’Eleuterio, for inspiring me with his imagination, for freeing me to pursue the problems that interest me, for binding me with his rigor, and for teaching me what research, in its purest form, is all about.

I thank my Doctoral Examination Committee members, Professor Craig Steeves, Professor Tim Barfoot, and Professor Jose Luis Perez Velazquez, for their valuable advice and constructive ideas.

I am indebted to so many others: To Lisa, Petar, Michael, William, Daniel, and all of Inside Out Ultimate, for keeping me sane; to Rebecca, for her love; and of course, to my mother, who has supported this seemingly endless education of mine (both tangibly and intangibly) since she taught her baby boy to read.
## Contents

1 **Introduction and motivation** .................................................. 1
   1.1 On the remembrance of things past ...................................... 1
   1.2 Modelling the hippocampus .............................................. 3
       1.2.1 Hippocampal function ........................................... 3
       1.2.2 Dynamical systems and state-space sculpting ................. 4
       1.2.3 Approximation by recurrent neural network .................. 5
   1.3 Modelling the cortex .................................................. 6
       1.3.1 Cortical function ................................................. 6
       1.3.2 Autoencoders .................................................... 7
   1.4 Foundational decisions .............................................. 8
   1.5 Overview ..................................................................... 9

2 **Literature review and background** ..................................... 12
   2.1 Modelling neurons and networks ..................................... 12
   2.2 Dynamical systems and their construction ......................... 17
       2.2.1 A primer .......................................................... 17
       2.2.2 Chaotic dynamics ............................................... 19
       2.2.3 The inverse problem ............................................ 21
       2.2.4 Sverdlove’s solution ............................................ 22
       2.2.5 Regularization .................................................. 24
   2.3 Training recurrent neural networks .................................. 27
       2.3.1 Backpropagation through time ................................ 28
       2.3.2 Real-time recurrent learning ................................ 29
4.10 A note on network potentials and catastrophic forgetting . . . . . . . . . . 108

5 The episodic memory model ......................................................... 111
  5.1 Overview ................................................................. 111
  5.2 Cortical model ......................................................... 111
  5.3 Cortico-hippocampal interaction ......................................... 114
  5.4 Storage and retrieval examples ........................................... 115
  5.5 Conclusion .............................................................. 121

6 Discussion and Future Work ...................................................... 123
  6.1 Dynamical systems: sculpting and exploring .............................. 123
  6.2 RNN synthesis: limitations, extensions, and biological plausibility . . 125
  6.3 The episodic model ..................................................... 133
  6.4 Principles of neural organization ......................................... 135

7 Conclusion .............................................................................. 137
  7.1 Overview ................................................................. 137
  7.2 Contributions to the field .................................................. 140
  7.3 Closing thoughts .......................................................... 141

Bibliography .............................................................................. 143
List of Figures

1.1 Salvador Dali’s “The Persistence of Memory.” ........................................... 2
1.2 Schematic of the episodic memory model. ............................................... 11

2.1 Schematic of a typical neuron demonstrating electrical flow along the axon, through the synapse, to the dendrites [73]. ................................. 13
2.2 Voltage as a function of time in the Hodgkin-Huxley model. Spiking frequency increases with input voltage. ................................. 14
2.3 Plot of the logistic sigmoid function with steepness $\gamma = 0.5$. ......... 17
2.4 The strange attractor of the Lorenz system. ........................................... 20

3.1 Graph of the rescaled logistic sigmoid function, $\gamma = 0.05$. ............. 52
3.2 Action of the sigmoid function on points in Euclidean space, which drives points to vertices of the hypercube. ................................. 54
3.3 Schematic of the Hopfield network’s state space when two diametrically opposed memory vectors are embedded. ......................... 59
3.4 Sampled basins of two memory vectors in a 3-neuron Hopfield network. . 60
3.5 The combined actions of $\sigma$ and $W$ generate Hopfield attractor dynamics. 62
3.6 Plot of attractor basins and separatrix (black) for three memories embedded in a Hopfield network. ............................................. 63
3.7 Vector field and several orbits for the regularized system of (3.6). .......... 66
3.8 Vector field and several orbits for the regularized system of (3.5). .......... 69

4.1 Schematic of the three stages of the RNN synthesis algorithm. The general vector field on the left may be a product of regularization. ............. 78
4.2 Vector field and orbits of a system with two fixed points. \( m = 7, d_\# = 800, \)
\( \text{MSE} = 8.1 \times 10^{-6}, e_{\text{max}} = 0.0075, e_{\text{orb}} = 0.033, e_L = 1.6 \times 10^9, T = 80, \)
\( t_\# = 1000. \) ................................................................. 88

4.3 Vector field and orbits of a system with four fixed points. \( m = 9, d_\# = 1600, \)
\( \text{MSE} = 2.2 \times 10^{-5}, e_{\text{max}} = 0.011, e_{\text{orb}} = 0.044, T = 80, t_\# = 1000. \) 89

4.4 Vector field and orbits of a system with three limit cycles. \( m = 100, \)
\( d_\# = 10000, \text{MSE} = 0.0081, e_{\text{max}} = 0.31, e_{\text{orb}} = 0.0033, T = 80, t_\# = 5000. \) 90

4.5 Vector field and orbits for the Van der Pol system. \( m = 10, d_\# = 1600, \)
\( \text{MSE} = 9.2 \times 10^{-5}, e_{\text{max}} = 0.031, e_{\text{orb}} = 0.022, T = 80, t_\# = 10000. \) 91

4.6 Vector field and orbits for the forced Van der Pol system. \( m = 20 + 4, \)
\( d_\# = 2400, \text{MSE} = 2.0 \times 10^{-9}, e_{\text{max}} = 1.5 \times 10^{-4}, e_{\text{orb}} = 0.0085, T = 400, \)
\( t_\# = 10000. \) ................................................................. 92

4.7 Vector field and orbits for the forced Duffing equation. \( m = 10 + 4, \)
\( d_\# = 1600, \text{MSE} = 4.0 \times 10^{-10}, e_{\text{max}} = 4.4 \times 10^{-5}, e_{\text{orb}} = 3.2 \times 10^{-4}, \)
\( T = 80, t_\# = 5000. \) ................................................................. 93

4.8 Vector field and orbits of a system with one limit cycle and one fixed point.
\( m = 60, d_\# = 18000, \text{MSE} = 0.0017, e_{\text{max}} = 0.16, e_{\text{orb}} = 0.068, T = 80, \)
\( t_\# = 5000. \) ................................................................. 94

4.9 Orbit of the chaotic Rössler system with RNN approximation. \( m = 40, \)
\( d_\# = 39304, \text{MSE} = 3.1 \times 10^{-5}, e_{\text{max}} = 0.055, e_{\text{orb}} = 3.6 \times 10^{-4}, T = 80, \)
\( t_\# = 2500. \) ................................................................. 95

4.10 Orbit of the chaotic Lorenz system with RNN approximation. \( m = 60, \)
\( d_\# = 91125, \text{MSE} = 0.0025, e_{\text{max}} = 0.6395, e_{\text{orb}} = 0.0067, T = 80, \)
\( t_\# = 10000. \) ................................................................. 96

4.11 Structure of a reservoir computer, with recurrently connected reservoir
and feedforward connected output unit(s). The reservoir is analogous to
the hidden units in our networks. Figure adapted from [89]. ............. 100

4.12 Evolution of \( \delta(t) \) over the course of an orbit from Figure 4.2. ............. 103

4.13 Gradient potential for the system depicted in Figure 4.2. ............. 106

4.14 Hamiltonian potential for the system depicted in Figure 4.2. ............. 106
4.15 Gradient potential for the system depicted in Figure 4.8. Note that figure has been rotated for better viewing. ........................................... 107

4.16 Hamiltonian potential for the system depicted in Figure 4.8. Note that figure has been rotated. ........................................... 107

5.1 a) A restricted Boltzmann machine. b) Two stacked RBMs forming an autoencoder. ........................................... 113

5.2 Schematic of the episodic memory model. Images are reconstructed by the autoencoder from an attracting trajectory embedded in the RNN. .... 115

5.3 Sequential frames from an animation of running motion. ................. 115

5.4 Codes in $\mathbb{R}^2$ and the images they reconstruct when fed back through a trained autoencoder. A fitted Fourier series is superimposed. ............ 117

5.5 Synthesized vector field with integrated orbit (green) and an orbit from a trained RNN (blue). ........................................... 118

5.6 Initial image input and sequential frames reconstructed by the autoencoder-RNN memory system. ........................................... 118

5.7 Orbits integrated from a trained RNN, along with attractor-basin separatrix and superimposed frame reconstructions. The RNN remembers two unique motion episodes shown in Figure 5.8 .... 120

5.8 Input images and reconstructed frames of running and walking episodes. Frames are reconstructed by an autoencoder-RNN memory system from the elliptical limit cycles in Figure 5.7. ................. 120

6.1 A sequence depicting RNN performance after incremental training runs on a fixed-point attractor system. The most recently trained trajectory is indicated in each subfigure by the red “T”. The sequence proceeds right-to-left, top-to-bottom. ........................................... 128

6.2 A sequence depicting RNN performance after incremental training runs on a limit-cycle attractor system. The most recently trained trajectory is indicated in each subfigure by the red “T”. The sequence proceeds top-to-bottom. ........................................... 129
6.3 A sequence depicting RNN trajectories after incremental training runs on a system with two fixed points. In the top subfigure only trajectories in the left half-plane have been presented to the RNN. In the bottom subfigure, trajectories from the right half-plane along with fifteen pseudotrajectories (plotted in magenta) have been presented for training.
Chapter 1

Introduction and motivation

A man’s real possession is his memory. In nothing else is he rich, in nothing else is he poor.

— Alexander O. Smith

1.1 On the remembrance of things past

The prospect of understanding and replicating the brain is at once daunting and endlessly promising. The human brain is, quite simply, the most complex system in the known universe. After more than 50 years of research, artificial intelligence systems still pale in comparison to their biological analogues (or inspirations) but they are catching up. Aircraft fly themselves, Google knows more or less what we’re searching for, and many of us speak naturally to assistants waiting in our pockets. This is only the beginning.

Memory is a pillar of intelligence. As the cognitive scientist Douglas Hofstadter observes, everyday thought is in general not problem-solving but rather a stroll through long-term memory. Most of the time we are unaware of this unceasing, bubbling recall, but every so often memory strikes us: For Proust, the singular taste of a madeleine dipped in tea summoned up a remembrance of things past that filled volumes.

We use analogies to label and understand situations (“that’s the pot calling the kettle black!”), and it is our ability to recognize similarities to the past, to map remembered, sometimes dismembered solutions onto analogous problems, that contributes so often to
Chapter 1. Introduction and motivation

Figure 1.1: Salvador Dali’s “The Persistence of Memory.”

the leaps and bounds of progress. To quote from Hawkins’ On Intelligence [45]:

Making predictions by analogy to the past is something you do continually while awake. This occurs along a continuum, ranging from simple everyday acts of perception occurring in sensory regions of the cortex to difficult, rare acts of genius occurring at the highest levels in the cortex. At a fundamental level, everyday acts of perception are similar to the rare flights of brilliance. It is just that the everyday acts are so common we don’t notice them.

Memory and analogy equip us to prosper in our uncertain but regular world.

Robotic memory, as manifest in digital stacks on silicon chips, is accurate, fast, and reliable to a degree that puts biological memory to shame. But robotic systems still cannot solve the open-ended, ill-posed problems at which we are so adept. Perhaps this is why: perhaps in its very inaccuracy, its slippery synthesis of disparate elements, its tendency to distort and decay (evoked brilliantly in Dali’s “The Persistence of Memory”, Figure 1.1), lies biological memory’s secret strength.

The ultimate vision of this work is to foster intelligent robotic life. It is our conjecture that to think like us, robots must remember like us. To that end, we develop a
computational model for memory that is inspired by the brain. The model is focussed, in particular, on episodic memory.

Episodic memory is the memory of autobiographical events, the collection of past experiences from a particular time and place, having some temporal extent and sequence [100]. It allows us mentally to travel back in time. It is what we see (or touch or taste) in our mind’s eye when we replay an event from the past.

As we will see in Chapter 2, existing biologically inspired episodic memory models lack the ability to store and reconstruct these movie-like episodes, which may be very high-dimensional. The model we develop herein, conversely, can record and retrieve image sequences composing temporally extended, animated memories. The model is based on the function of the hippocampus and its interaction with the cortex.

1.2 Modelling the hippocampus

1.2.1 Hippocampal function

The hippocampus is a seahorse-shaped component of the limbic system that lies beneath the cerebral cortex. There is a general consensus that it plays a critical role in the encoding and retrieval of episodic memory [82]. This view dates back to evidence from the well-known psychiatric patient H.M. [81], and is supported by modern neurophysiological [86, 17] and imaging data [80].

Study of the hippocampus has led many to theorize that this brain structure acts as an associative memory system [65, 75, 96]. From a computational standpoint, associative memory systems perform pattern completion. Stored patterns are reconstructed from cues containing incomplete information about them [82, 10], and this process is what we might call recollection: the triggering of a memory from the past by a similar event in the present.

In our model, we will mimic this associative function with a hippocampal analogue. Region CA3 of the hippocampus is believed to be the particular locus of associative function; it is a region characterized by highly recurrent connectivity between neurons
We therefore embody the model’s hippocampal analogue as a continuous-time recurrent neural network (RNN). RNNs represent a large class of computational model designed to represent the brain abstractly. As such they consist of numerous artificial neurons (represented as mathematical functions) connected in some graph. The feature that distinguishes RNNs from feedforward neural networks (more widely used in machine learning, to date) is the existence of closed cycles in the connection topology. As a consequence of these cycles, RNNs may exhibit self-sustained dynamics in the absence of any input [60]. Mathematically, RNNs are dynamical systems.

1.2.2 Dynamical systems and state-space sculpting

It is no surprise, given its purported role in memory, that the hippocampus functions as a dynamical system, because dynamical systems are well suited to the modelling of episodes that extend through time. This is because they describe the time dependence of a point in some geometrical space (called the state space). Dynamical systems are represented typically (and herein) as a set of ordinary differential equations (ODEs) plus initial conditions, with the ODEs defining the time-evolution of the system from initial conditions. The path that the system follows through the state space is called an orbit or a trajectory.

Dynamical systems have in fact become a mainstay of computational neuroscience, underlying a variety of models for information processing and memory function in the brain [18], [55], [4]. Those used most often in memory modelling contain attractors, which are points or sets of points in the state space toward which the system converges through time. Associative memory function in particular can be manifest in a dynamical system with attractors. Pattern completion is achieved when a given pattern is an attractor and incomplete cues are points within that attractor’s basin. In this case the system evolves from the cues to the full pattern. Attractor states can be used thus to represent specific memories [7].

As an internal record of the infinitely varied and varying external world, the ideal dynamical system for memory should possess a state space that can be structured arbitrarily. That is, it should be possible to render any point or set of points an attractor, and
embed within the state space prescribed orbits. As a stepping stone towards the memory model we have developed a framework for synthesizing such arbitrary, multiatractor systems. We call it *state-space sculpting.*

A dynamical system’s compositional ODEs define a vector field. State-space sculpting uses a mathematical technique called *regularization* to combine a collection of discontinuous vector fields into a single continuous system. In this manner, individual attractor systems can be joined together to form a prescribed multiatractor system—that is, a system that stores multiple memories. Dynamical systems so designed form the foundation of our hippocampal model. To exist in plausible hippocampal form, they must be recast as recurrent neural networks.

### 1.2.3 Approximation by recurrent neural network

Our goal in developing the hippocampal analogue is the manifestation of arbitrary dynamical systems as RNNs. Although recurrent neural networks are typically written as a set of ordinary differential equations, the network form induces certain constraints on the operations that variables may undergo. This means that, in general, a given dynamical system is not a recurrent neural network (although every recurrent neural network is a dynamical system).

Nevertheless, it was proved by Funahashi and Nakamura [27] that any dynamical system can be approximated to arbitrary accuracy by a recurrent neural network. Funahashi and Nakamura’s proof was not constructive, but we have harnessed it to develop a training algorithm for RNNs. The algorithm involves training a three-layer feedforward network to approximate the vector field function of a prescribed dynamical system; then the feedforward network is transformed into recurrent form by restructuring certain weight matrices. The recurrent network so designed replicates the original system’s dynamics—in particular the type and location of attractors in the state space.

It is these trained RNNs that supply the memory model’s hippocampal analogue. A recurrent network is made to store specific, sculpted state-space orbits and attractors that encode episodic memories.
1.3 Modelling the cortex

1.3.1 Cortical function

The cerebral cortex is the brain’s outer layer of neural tissue, commonly described as comprising three parts: the sensory, motor, and association areas. These are connected to various subcortical structures, including the hippocampus. The cortex plays a key role in memory, attention, perceptual awareness, thought, language, and consciousness. Here we focus on its contribution to episodic memory.

Imaging studies [72, 108] suggest that memory recall involves a constructive process [11]. More specifically, it is theorized [82] that the seed of an episode, consisting of a relatively small quantity of information, is encoded in the hippocampus during memorization (this encoding is called the episodic memory trace); during recall, an episode’s trace becomes active and reinstates the episode in cortical circuits and those cortical circuits reconstruct the episode via mental simulations.

In our model, this function will be realized by a cortical analogue whose primary task is to reconstruct an episode from its memory trace in the hippocampal unit. In this sense the cortical analogue runs a type of decompression: It unfolds a trajectory in the hippocampal state space into a sequence of remembered sensory patterns (sights, sounds, etc.).

Such decompression at the point of memory reconstruction implies compression at the point of memory formation; i.e., the stored memory trace must be distilled from an original, experienced episode. This distillation, too, is a putative function of cortico-hippocampal interactions in the brain. According to theory ([82] and references therein), experiences are first construed mentally as a stream of events or situations, with these construals expressed as transient patterns in cortical circuits. Patterns are projected in turn to the entorhinal cortex (the main interface between cortex and hippocampus), thence propagating in a loop around the hippocampal system. There they trigger changes in neuronal connectivity, constituting the formation of an episodic trace.

With this in mind, our memory model’s cortical analogue has been developed to subsume the dual functions of compression for memory storage and decompression for
memory reconstruction. It is embodied as a deep autoencoder.

1.3.2 Autoencoders

Deep autoencoders are stochastic feedforward neural networks used primarily for dimensionality reduction [47], that is, the determination of compact representations of high-dimensional data. Such representations are often referred to as codes. However, autoencoders can also be run in reverse, in the so-called generative mode, to reconstruct high-dimensional data from compact codes. This is because, in order to learn efficient encodings, autoencoders are trained to reconstruct their own inputs.

The deep autoencoders used in our model are made by stacking restricted Boltzmann machines (RBMs) [47]. These are two-layer stochastic neural networks that learn probability distributions over their input sets. Inputs are represented at one neuron layer while “hidden features” in the input data are represented at the other. Through training, the network effectively discovers which hidden features enable it to reconstruct the inputs. A stack of decreasingly sized RBMs iteratively compresses data to a small set of hidden features that constitutes a code.

Restricted Boltzmann machines are not the only model that can be used for autoencoding. They have been selected because they perform well and because they are, in essence, neural networks.

As highlighted in Chapter 2, our overarching memory model is motivated by known brain physiology and neural processing (although it is highly abstracted). We utilize both the compressive and generative functions of the deep autoencoder to mimic the corresponding functions of the cortex: compression of episode data to codes and regeneration of episodes from codes. The cortical autoencoder works in concert with the hippocampal analogue, feeding it signals and receiving signals from it as follows. A stream of inputs, constituting an episode, is compressed by the autoencoder to a set of points in a low-dimensional code space; the code space acts also as the state space of the hippocampal RNN; an orbit through the set of code points, in the proper temporal order, constitutes an episodic memory trace; the hippocampal RNN is trained to store this orbit as an attractor. This process constitutes the formation of a memory. Memory retrieval begins
with the dynamical evolution of the hippocampal RNN; this evolution converges upon and follows the stored memory trace since it is attracting in the state space; the cortical autoencoder takes in the reinstated memory trace as a sequence of codes through time, and from it reconstructs (approximately) the original stream of inputs making up the episode. This summarizes the overall function of the memory model.

1.4 Foundational decisions

Underlying our episodic memory model are two key decisions. First, we will treat memory traces as deterministic rather than stochastic objects. Likewise, the memory retrieval process, in this work, is not tantamount to sampling from some distribution. This decision is a simplifying one, and allows us to focus on deterministic rather than stochastic dynamical systems, but it is also, largely, philosophical: we believe that the goal of a memory system should be to record deterministically.

Of course, the idea that sensory inputs emerge from some external distribution is very sensible, and memories are noisy — different people have different recollections of the same event, and the recollections of individuals change over time. Our model will not entirely ignore these realities. Stochasticity enters our system via the deep autoencoder, which is a stochastic network. It models uncertainty and noise in the sensory inputs and their reconstructions, and this uncertainty flows into the hippocampal representation during the encoding phase.

This decision defines the choice of RNN used to model the hippocampus. Elman/Jordan networks and long short-term memory (LSTM) networks have achieved impressive results in sequence modelling for various domains such as natural language processing [49, 88, 67]; however, they are trained on vast swathes of sequence data presumed to emerge from some underlying distribution. They model the probability of an output state conditioned on an observed state and a previous latent state. In contrast, our aim in the present work is a deterministic dynamical system that, given an input, converges deterministically on the associated attractor state. Future work should focus on the role of stochasticity and noise in the hippocampus.
The second key decision we make is to focus on continuous-time dynamical systems, and continuous-time RNNs for their approximation. This is in contrast to discrete-time systems, modeled by finite-difference equations rather than ordinary differential equations. Our motivation here is partly to explore an area that has received significantly less treatment in the literature: as will be seen in Chapter 2, existing techniques for RNN training focus almost exclusively on discrete-time networks. It also must be said that, although a translation can generally be made between discrete-time and continuous-time systems, there are inherent dynamical differences. For instance, chaotic dynamics cannot exist in a continuous system with fewer than three dimensions, while one-dimensional discrete-time systems like the logistic map readily exhibit chaos. Working in continuous time avails us of all the work in functional analysis, which we can turn upon our systems to gain theoretical insight. Indeed, we will found our approach to RNN training on a convergence result for continuous-time systems. Finally, we note that real-world systems, like the brain, are generally continuous (although the brain also makes use of discrete spiking dynamics).

1.5 Overview

The subject of this dissertation is a computational model for human-like episodic memory. The model is inspired by the function of the hippocampus and its interactions with the cortex. It comprises two neural components: a hippocampal analogue, manifest as a recurrent neural network, and a cortical analogue, manifest as a deep autoencoder. Together, these components encode incoming experiences into episodic memory traces, store those traces as associative orbits in a dynamical system, and remember stored experiences through dynamical evolution and episode reconstruction.

The model is not intended to mimic microscopic brain function with high fidelity. It is mesoscopic in nature, meant to tackle the problem of episodic memory from a dynamical-systems perspective. Because dynamical systems are fairly well understood and modelled with rigorous mathematics, it is our hope that this approach will lead to a better understanding of memory mechanisms in the brain. In particular, the recurrent
networks synthesized to store prescribed memory traces may, by their weight-matrix structure, illuminate and predict principles of neural organization in the brain.

The primary motivation in developing this model is the creation of more intelligent, more capable robotic systems. By the dissertation’s conclusion, we will have explicated, and instantiated as software, a system of coupled artificial neural networks capable of remembering visual episodes from similar cues. This comprises an episodic memory that can be utilized in robotic applications.

The main contributions to the field that this work entails are summarized below, and will be detailed throughout the dissertation.

- A framework, state-space sculpting, has been developed to construct arbitrary, prescribed dynamical systems. This involves use of a technique called regularization to combine discontinuous vector fields together and to delineate attractor basins in the combined fields.

- An algorithm has been developed to synthesize recurrent neural networks that replicate arbitrary dynamical systems. To our knowledge, this algorithm gives state of the art performance for continuous-time dynamical-system approximation in terms of accuracy with respect to number of neurons required.

- A model of human-like episodic memory was developed, capable of remembering animated visual episodes. This model combines a recurrent neural network with a deep autoencoder.

The neural architecture of our episodic memory model is depicted in Figure 1.2, which shows the feedforward autoencoder used to compress and regenerate episodes and the recurrent network that stores and recalls memory traces.

The dissertation is organized as follows. In Chapter 2, we review the literature on episodic memory and existing theoretical and computational models, as well as provide background information for various concepts discussed further on. Chapter 3 presents the mathematics of state-space sculpting with some examples, and analyze the well known Hopfield network from a state-space sculpting perspective. The mathematics for synthesis of recurrent neural networks from feedforward networks, and the training methodology
for replicating dynamical systems with RNNs, are presented in Chapter 4. Chapter 4 also demonstrates the capabilities of the RNN synthesis algorithm through several examples. In Chapter 5 we explain and construct the episodic memory model, combining a synthesized RNN (the hippocampal analogue) with a trained autoencoder (the cortical analogue). In this system we store multiple visual episodes and demonstrate remembrance of the same. Chapter 6 provides a discussion of the model, focusing on learning mechanisms and extensions of the model. We conclude the thesis in Chapter 7.
Chapter 2

Literature review and background

2.1 Modelling neurons and networks

Neurons are the core component of the central nervous system and the brain. They are electrically excitable cells that process and transmit information through electrical and chemical signals. They consist of a cell body, an axon, and dendrites. Neurons connect to each other to form networks, in which the axon of one cell sends signals to the dendrites of others. Signals must cross gaps known as synapses. See Figure 2.1. Each neuron has a single axon and multiple dendrites, but the axon may make connections to the dendrites of many other cells.

Metabolically driven ion pumps maintain a voltage gradient across a neuron’s cell membrane. When the voltage changes by an amount greater than the neuron’s threshold, an all-or-nothing response called an action potential is generated. An electrical pulse propagates along the axon, through charge-carrying ions, to its terminus at a synapse; chemicals called neurotransmitters are then released into the synapse where they bind with dendritic receptors; this binding opens ion channels within the post-synaptic neuron, leading to a voltage change therein. Depending on the strength of this excitation, the target cell may go on to fire an action potential of its own. Excitation strength depends on properties of the pre- and postsynaptic neurons, including the quantity of neurotransmitter available for release at the signal cell and the number of receptors on the target cell. These properties change over time so that connections between neurons,
Figure 2.1: Schematic of a typical neuron demonstrating electrical flow along the axon, through the synapse, to the dendrites [73].

and the signals sent across them, grow stronger or weaker. Roughly speaking, this process of synaptic-strength modification is how learning takes place.

It is from this seemingly simple, evolving intercellular communication that all the complexity of human behaviour arises.

Biologically plausible neuron modelling began with Hodgkin and Huxley’s work on the squid giant axon [50]. The conductance-based model they derived from experiment describes how action potentials are initiated and propagated. It consists of an electrical circuit model and attendant set of four nonlinear differential equations. In the circuit, the cell membrane is represented as a capacitance, voltage-gated and leaky ion channels are represented by conductances, the gradients driving the ion flows are represented as batteries, and ion pumps are represented by a current source. The voltage response of the model to a current injection can be seen in Figure 2.2, highlighting the periodic firing of action potentials. Note that the frequency of firing increases as does the input voltage. It should also be clear from the figure why action potentials are often referred to as “spikes.”
Hodgkin and Huxley’s differential system was distilled to two dimensions by FitzHugh [22], by abstracting away some of the physiologically faithful system parameters. His model is classified as a relaxation oscillator: If the external stimulus exceeds a threshold value then the system generates a spike of activity before relaxing to its rest state. An equivalent circuit for the model was developed by Nagumo [70].

Nagumo and Sato [69] later discretized the model as a single finite-difference equation that encompassed the rich behaviour of the original, continuous, 4-dimensional system. Since that time numerous refinements and extensions have been made. Some models aim for physiological realism while others aim for simplicity of simulation and analysis (of course with minimal sacrifice to phenomenological realism).

One of the most commonly used neuron models in modern, large-scale brain simulations is the leaky integrate-and-fire (LIF) neuron [18]. This falls into the latter category. A neuron is modelled as a leaky integrator of its input current, $I(t)$, according to

$$C \frac{dV}{dt} = - \frac{V(t)}{R} + I(t),$$

where $V(t)$ represents the membrane potential at time $t$, $C$ is the membrane capacitance, and $R$ is the membrane resistance. The potential $V(t)$ is the quantity of interest.
Equation (2.1) describes a resistor-capacitor (RC) circuit. Ion leakage flows through the resistor while integration of the current is due to the capacitor. As can be inferred from the form of (2.1), spiking events are not explicitly modelled in the LIF. Rather, when the membrane potential reaches a certain threshold value $V_{th}$ it is instantly reset to a resting value $V_r$. The leaky integration process then restarts after a refractory period of $\tau_r$ seconds. Input current must exceed the threshold $I_{th} = V_{th}/R_{th}$ for the cell to fire a spike. The cell’s firing frequency is computed according to

$$f(I) = \begin{cases} 0, & I \leq I_{th}, \\ (\tau_r - RC \log(1 - \frac{V_{th}}{I_{th}}))^{-1}, & I > I_{th}. \end{cases}$$

In many models, particularly within the field of recurrent-network modelling, the concept of discrete spiking events is abstracted away entirely. In these cases, neuron state variables are taken to represent firing frequencies as in (2.2), or mean membrane voltages, rather than instantaneous voltages as in (2.1). It has been thought, since the pioneering work of Adrian [2, 3], that a neuron’s firing rate encodes most of the information it communicates. Adrian [2] showed that the firing rate of stretch receptor neurons in muscles is related to the force applied to the muscle, and so inferred that frequency encodes stimulus strength. This frequency approach has been applied successfully to neural measurement, modelling, and simulation for the last 80 years.

Of course, this approach neglects any information that may be encoded in the exact timing of spikes, and indeed, experimental evidence exists to suggest the approach may be too simplistic to account for all of the brain’s phenomena [54, 95, 74]. Nevertheless, nonspiking models remain a cornerstone of neural network research because they account for an important spectrum of observed neural function. The “additive model” of Grossberg [32, 33] was drawn from psychological data on the emergence of learning in living animals adapting autonomously in real time. It is a system of nonlinear differential equations designed to model how brain functions control behaviour, and has found widespread use in biological network research.

The additive model is so named because the terms that determine the rate of change
of neuron potentials are summed. In its simplest form, the model can be written as

$$\dot{s}(t) = -Ps(t) + W\sigma(s(t)) + u(t). \quad (2.3)$$

Here\(^1\), \(s(t) \in n^+m\mathbb{R}\) is the vector of internal neuron states (representing mean potentials) at time \(t\), \(P \in n^+m\mathbb{R}^{n+m}\) is a diagonal matrix of passive decay constants, \(W \in n^+m\mathbb{R}^{n+m}\) is the matrix of connection strengths, \(\sigma\) is the neuron activation function, and \(u(t) \in n^+m\mathbb{R}\) is the vector of external inputs from outside the network.

The decay terms cause neural activity to die out in the absence of stimulation from outside the network or from connected neurons within. The activation function is used to impart the all-or-nothing effect to signals incoming from connected neurons: typically, the nonlinear logistic sigmoid

$$\sigma(x) = \frac{1}{1 + \exp(-x/\gamma)} \quad (2.4)$$

is used, which, approximately, maps inputs \(x < 0\) to 0 and inputs \(x > 0\) to 1. The parameter \(\gamma\) controls the steepness of the transition, and the function’s form for \(\gamma = 0.5\) is plotted in Figure 2.3. The matrix \(W\) is referred to as the weight matrix. The weights it collects scale incoming signals, thereby modelling the strength of the synapse between each pair of neurons. Element \(w_{ij}\) scales the signal from neuron \(j\) into neuron \(i\). Elements may be positive, producing excitatory signals, or negative, producing inhibitory signals.

The additive model has considerable power, having found application in computational vision, learning, reinforcement learning, language processing, sensory-motor control, and associative memory [31, 36, 37, 39, 40, 101]. It was further proved capable, by Funahashi and Nakamura, of approximating to arbitrary accuracy any autonomous dynamical system [27]. It is this particular model that we use to embody our memory system’s recurrent neural networks.

---

\(^1\)In the notation used, \(^n\mathbb{R}\) refers to the vector space of \(n \times 1\) real column matrices, \(\mathbb{R}^m\) the vector space of \(1 \times m\) real row matrices, and \(^n\mathbb{R}^m\) the vector space of \(n \times m\) real matrices. Note that if \(A \in ^n\mathbb{R}^m\) then \(A\) is also a map, \(A : m\mathbb{R} \rightarrow n\mathbb{R}\).
2.2 Dynamical systems and their construction

2.2.1 A primer

A dynamical system is a mathematical concept in which a deterministic rule, called the evolution rule or update rule, describes how a point in a geometrical space depends on time. The swinging of a pendulum, the flow of electric charge in a circuit (or a neuron), and the population numbers of interacting predator-prey species are just a few examples of systems that can be modelled within this framework. Dynamical systems concepts have their origins in Newtonian mechanics.

The evolution rule is an implicit relation specifying how the system state changes over a short time interval, possibly only an infinitesimal instant (the relation is either a difference equation for discrete time or differential equation for continuous time). To determine the state for all future times requires iterating the relation. The iteration procedure is referred to as solving the system or integrating the system. Given an initial point it is possible to determine all its future positions by solving/integrating; the resultant collection of points is called a trajectory or orbit.

Technically speaking [91], a differentiable dynamical system on an $n$-dimensional manifold $M$ is a differentiable action of the real numbers on $M$, in particular, a $C^1$ (i.e., con-
tinuous) map $\phi : M \times \mathbb{R} \to M$ with the property that $\phi(\phi(x, t), s) = \phi(x, t + s)$, $x \in \mathbb{R}^n$. These maps occur as the flows associated with vector fields on $M$; typically the map $\phi$ itself is referred to as the flow. The orbit through a point $x$ is the curve $\phi(x, \mathbb{R})$. This curve is the solution to the initial-value problem at $x$ and there are well-known theorems on the existence and uniqueness of such solutions (see [48]).

An orbit is closed if it corresponds to a periodic solution, i.e., if $\phi(x, t) = \phi(x, t + T)$ for some period $T$. A closed orbit is a limit cycle if it is isolated from all other closed orbits. A critical point (or fixed point) is a point $x$ where the vector field vanishes, so that $\phi(x, \mathbb{R}) = \{x\}$. A critical orbit is either a critical point or a limit cycle. A critical orbit is stable if it has an arbitrarily small neighborhood in which all other orbits approach it as $t \to \infty$ without leaving the neighborhood. In particular, this is the definition of Lyapunov stability. If the previous condition is true as $t \to -\infty$ then the orbit is unstable. An orbit is said to be orbitally stable if the trajectory through any point sufficiently near the orbit stays near it for all time. Otherwise the orbit is called singular. The neighborhood within which all orbits approach a stable orbit is called the orbit’s basin of attraction.

There are various “types” of critical point: sinks, which are stable; sources, which are unstable; saddles, which are stable in some directions (the stable manifolds) and unstable in others (unstable manifolds); and centers, which are surrounded by a family of closed orbits. The different cases may be distinguished by considering the eigenvalues of the Jacobian matrix of $\phi$. A critical point is degenerate if the Jacobian is singular at that point.

There may exist boundaries in a system’s state space that separate orbits with different properties, sometimes, for example, lying on the interface between two basins of attraction. These boundaries are called separatrices. A separatrix $S$ is an invariant manifold, i.e., if $\phi(x, t) \in S$ then $\phi(x, t + \tau) \in S \forall \tau \in \mathbb{R}$. For example, in the equation of motion for a pendulum a separatrix exists at the angular momentum where libration becomes rotation.

We say that two dynamical systems are topologically equivalent if there is a homeomorphism of the phase space (or state space), $M$, which preserves orbits. A dynamical system is structurally stable if it has a neighborhood in the appropriate space of all flows
in which every other system is topologically equivalent to it. Loosely speaking, this means that perturbing the map $\phi$ does not change the topological qualities of its orbits—for instance, the types of its critical orbits.

Dynamical systems are now a mainstay of computational neuroscience. They underlie a variety of models for information processing and memory function in the brain [18], [55], [4], and persistent activity in biological neural networks is posited to result from dynamical attractors in neural state space [7].

2.2.2 Chaotic dynamics

It turns out that simple nonlinear dynamical systems can exhibit completely unpredictable behavior that appears to be random, in spite of the fact that their evolution rules are deterministic. This seemingly random but fundamentally deterministic behavior is called chaos, a property that is difficult to define precisely owing to various interpretations. The hallmark of chaos is sensitivity to initial conditions, whereby initially close points in state space evolve quickly (typically exponentially) to mutually distant points. This phenomenon is known colloquially as “the butterfly effect” and accounts for a chaotic system’s unpredictability.

There are three properties that generally characterize a chaotic dynamical system [105]:

(i) A dense set of points with periodic orbits (a subset $A$ of set $X$ is dense if its set closure $\overline{A} = X$),

(ii) Sensitivity to the system’s initial condition, and

(iii) Topological transitivity; a function $f$ is topologically transitive if, given any two intervals $U$ and $V$, there is some positive integer $k$ such that $f^k(U) \cap V \neq \emptyset$.

Perhaps the earliest discovery of chaos was made by Henri Poincaré, who, while studying the famous three-body problem in the 1880s, found that there exist orbits that are nonperiodic and yet not forever increasing nor approaching a fixed point. Indeed, chaos adds a new type of attractor to the dynamical systems repertoire: the strange attractor. Strange attractors, unlike fixed points and limit cycles, possess great detail
Chapter 2. Literature review and background

and complexity, usually exhibiting a fractal structure. They are sets on which chaotic behaviour takes place and which attract initial conditions from some neighborhood; more precisely, they are attracting sets with zero measure and fractal dimension [106]. The well known Lorenz attractor is plotted in Figure 2.4 to provide an example.

![Figure 2.4: The strange attractor of the Lorenz system.](image)

We discuss chaos here because various researchers have postulated that chaos may be important to memory dynamics [25, 98]. For instance, based on electroencephalography studies of the mammalian olfactory system, Freeman [25] conjectures that distinct remembered stimuli are represented by distinct attractors where the shift from one to another is a transition to chaos. Likewise Tsuda [98] suggests that chaotic behaviour may act as the neural correlate of a temporal episode in a process consisting of attractor “snapshots” linked in coded order by intervening chaotic motion.

Because of the potential importance of chaos to memory, we will demonstrate that our hippocampal analogue is capable of exhibiting chaotic behaviour and strange attractors.
2.2.3 The inverse problem

The inverse problem for dynamical systems is the construction of a system of equations given prescribed dynamical properties of their solutions [91]. The prescription may take the form of a schematic diagram of the flow in state space or a list of analytically defined critical orbits. For our purposes, solution of the inverse problem enables the construction of systems containing arbitrary prescribed memory traces.

The first attack on the inverse problem was made by Forster [23], who showed how to construct a system in the plane ($\mathbb{R}^2$) with a single critical point of prescribed type (within certain restrictions). Gil’derman [29] later generalized a similar result for piecewise linear systems to $n$ dimensions. Markus [61] showed that any topological type of flow in the plane with no critical points and a countable number of isolated separatrices can be realized by a system of differential equations, and Ganiev [28] showed how to construct $C^\infty$ equations for structurally stable configurations in the plane having all their critical orbits inside a limit cycle.

Sverdlove [91] considered a more general case, solving the inverse problem allowing for an arbitrary finite number of critical points and of periodic orbits. He begins by constructing a family of equation systems whose solutions all have the prescribed local critical behavior. Then he selects from this family the subfamily of systems whose critical orbits have the proper global relations to each other. Solution of the local problem is achieved by extending the work of Al’mukhamedov [5] and Valeeva [102] whose constructions work only in certain special cases. Sverdlove modifies these constructions, obtaining a solution to the global inverse problem in the plane for systems of structurally stable topological type.

Using Sverdlove’s method we can embed prescribed stable fixed points and limit cycles within a dynamical system. If we use that system as the basis for a hippocampal recurrent neural network, then the stable fixed points and limit cycles function like episodic memory traces, as we demonstrate in this work. Next we summarize the important mathematical results that underlie Sverdlove’s solution [91].
2.2.4 Sverdlove’s solution

First we tackle the construction of limit cycles. Suppose we have a set of $m$ simple, disjoint closed curves in $\mathbb{R}^2$

$$C_i : f_i(x, y) = 0, \ i = 1, \ldots, m, \quad (2.5)$$

where the functions $f_i$ are sufficiently smooth, have no critical points on the $C_i$ and are chosen so that $f_i$ is positive outside $C_i$ and negative inside. The goal is to find functions $P(x, y)$ and $Q(x, y)$ defining a planar vector field such that the system

$$\frac{dx}{dt} = P(x, y), \quad \frac{dy}{dt} = Q(x, y) \quad (2.6)$$

has the curves (2.5) and no other as its limit cycles. To deal with all the curves at once, the function

$$F(x, y) = \prod_{i=1}^{m} f_i(x, y) \quad (2.7)$$

is formed so that the union of the curves is the set $Z = F^{-1}(0)$. Since the $f$’s have no critical points on $C_i$, $F$ has none either. For the curves to be limit cycles, $Z$ must be an invariant set of the flow (2.6). Thus we require $(dF/dt)(x, y) = 0$ for $(x, y) \in Z$ moving along an orbit. Using the shorthand $F_x \triangleq \partial F/\partial t$ and likewise for $F_y$, it can be shown that $dF/dt = PF_x + QF_y = 0$ whenever $F = 0$ if and only if $P$ and $Q$ take the form

$$P = AF_y + B_1F, \quad Q = -AF_x + B_2F, \quad (2.8)$$

where $A, B_1, B_2$ are arbitrary functions with $A \neq 0$. This renders the $C_i$ closed orbits of (2.6) but we must choose $A, B_1, B_2$ so that they are limit cycles. Al’mukhamedov showed that the choice $A(x, y) \triangleq 1, \ B_1 = G(x, y)F_x, \ B_2 = G(x, y)F_y$ would achieve this, yielding the system

$$P = F_y + FGF_x, \quad Q = -F_x + FGF_y. \quad (2.9)$$
In particular, setting $G \triangleq 1$ renders the family (2.5) as the only limit cycles of the flow (2.9) [5]. However, it can be shown that all these limit cycles are unstable.

It was Sverdlove who proved that the stability of each limit cycle could be prescribed. Along any orbit of (2.9) it is the case that

$$\frac{dF}{dt} = FG(F_x^2 + F_y^2).$$ (2.10)

Then it is easy to prove the following result using the characteristic index [91] of each $C_i$:

**Lemma 1** (Sverdlove [91]). (a) If $G < 0$ in a neighborhood of $C_i$, then $C_i$ is a stable limit cycle of (2.9). (b) If $G$ changes sign going across $C_i$, then $C_i$ is a semi-stable limit cycle of (2.9).

Here, *semi-stable* means that trajectories flow towards the limit cycle on one side and away from it on the other. This lemma is in service of the following theorem:

**Theorem 1** (Sverdlove [91]). Suppose we prescribe in advance the stability-type for each $C_i$. Then there exists a function $G(x,y)$ such that each $C_i$ has the desired type and there are no other limit cycles of the system.

The theorem’s proof details the construction of the required $G$ and thereby equips us to build dynamical systems with desired limit cycles. The direction of flow on these cycles has not yet been defined, but Sverdlove further proves, with a system of the form

$$\frac{dx}{dt} = G^*F_y + FG_Fx, \quad \frac{dy}{dt} = -G^*F_x + FG_Fy$$ (2.11)

where now $A(x,y) = G^*(x,y)$, that

**Theorem 2** (Sverdlove [91]). There exists a function $G^*(x,y)$ such that the system (2.11) has a prescribed direction of flow on each limit cycle $C_i$.

Now we elucidate the prescription of critical (fixed) points. Supposing that $(x_0, y_0)$ is a critical point of (2.9), then $P(x_0, y_0) = Q(x_0, y_0) = 0$. This leads to the system of
homogeneous equations

\[ F_y + FGF_x = 0, \quad -F_x + FGF_y = 0. \] (2.12)

Sverdlove’s next important result is the following:

**Theorem 3** (Sverdlove [91]). The set of critical points of (2.9) is the same as the set of critical points of \( F \). In particular, a source or sink of (2.9) is a maximum or minimum of \( F \), a saddle point of (2.9) is a saddle point of \( F \), and degenerate critical points also coincide.

Given this information, if \((x_0, y_0)\) is to be a critical point then we simply include in \( F \) a factor \( f_{m+1}(x, y) \) that is 0 at \((x_0, y_0)\) and positive elsewhere. Any number of such factors can be put into \( F \). Furthermore, a factor of \( G \) may be used, as in the limit-cycle case, to render \((x_0, y_0)\) a source or sink as desired. This is formalized in the following.

**Theorem 4** (Sverdlove [91]). The function \( F \) can be redefined so that the system (2.9) will have sources and sinks at prescribed points.

Finally, a theorem of Valeeva [102] confirms that saddle points and centers may be obtained at prescribed points as well. So concludes our brief summary of the solution of the inverse problem in the plane.

### 2.2.5 Regularization

Sverdlove’s inverse solution is restricted to the plane \( \mathbb{R}^2 \). However, in synthesizing complicated memory traces and the state spaces that contain them, it may be necessary to work in higher dimensions. Furthermore, it is often difficult in practice to construct the functions \( G \) and \( G^* \) appropriately, and to find factors of \( F \) for critical points. Both of these issues become increasingly limiting as more memory traces are stored in a state space.

As such, we seek alternative, more flexible methods for the synthesis of vector fields. One approach we have taken is based on uniting sets of discontinuous vector fields. It was
inspired by investigations into the Hopfield network’s state space, wherein stable critical points are embedded in locally attracting regions delineated by separatrices. It is often straightforward to construct differential systems containing single, isolated critical points and limit cycles, even in higher dimensions. Thus, a more complicated multiattractor system can be synthesized by combining individual single-attractor systems. The issue is how to combine them appropriately, since individual fields will in general be mutually discontinuous where they interface. One method that can be used to unite discontinuous vector fields into a continuous whole is called regularization. Its exposition is the subject of this section.

Sets of discontinuous vector fields have been well studied in the past. Much of the pioneering early work was done by Filippov [21]; however, his motivation was a means to understand and model systems that exhibit discontinuities, such as mechanical assemblies with fluctuating contact or electrical circuits with switching. Regularization was developed by Sotomayor and Teixeira [85]. The first results, like the solution of Sverdlove, were restricted to $\mathbb{R}^2$, but the method was later generalized to higher dimensions [59]. Here we will relate the original results for the sake of simplicity.

Following Sotomayor and Teixeira [85], let $M$ be a 2-dimensional manifold and $f : M \to \mathbb{R}$ be a $C^\infty$ function. We define a boundary $S = f^{-1}(0)$ which divides $M$ into two disjoint regions. Boundary $S$ is assumed to have a single connected component in such a way that $M \setminus S$ has two connected components denoted by $M^+ = f^{-1}(0, \infty)$ and $M^- = f^{-1}(-\infty, 0)$. Let $\chi^r$ denote the space of $C^r$ vector fields on $M$ and let $\Omega = \Omega(M, f)$ be the space of vector fields $Z$ on $M$ defined by

$$Z(q) = \begin{cases} X(q) & \text{if } f(q) > 0 \\ Y(q) & \text{if } f(q) < 0 \end{cases}$$

where $X, Y \in \chi^r$. Thus $Z$ is a discontinuous combination of the fields $X$ and $Y$. We write $Z = (X,Y)$ to highlight that relationship.

The smoothing effect of regularization, which renders the combined field continuous, is obtained using a transition function. This is a $C^\infty$ function $\varphi : \mathbb{R} \to \mathbb{R}$ such that $\varphi(t) = 0$ if $t \leq -1$, $\varphi(t) = 1$ if $t \geq 1$ and $\varphi'(t) > 0$ if $t \in (-1, 1)$. 
Chapter 2. Literature review and background

Definition 1 (Sotomayor and Teixeira [85]). A $\varphi_\epsilon$-regularization of $Z = (X, Y) \in \Omega$ is the one parameter family of vector fields $Z_\epsilon$ in $\chi^r$ given by

$$Z_\epsilon(q) = [1 - \varphi_\epsilon(f(q))]Y(q) + \varphi_\epsilon(f(q))X(q)$$

where $\varphi_\epsilon(t) = \varphi(t/\epsilon)$.

In their paper, Sotomayor and Teixeira give conditions on $Z = (X, Y)$ which determine the global phase portrait of the regularization and guarantee the structural stability of that regularization for any transition function and small $\epsilon$. This is based on the characterization of the class $\Sigma^r$ of structurally stable vector fields on smooth submanifolds of $M$ due to Andronov-Pontryagin. Let $N$ be a 2-dimensional submanifold of $M$ with boundary $\partial N$ and let $X_N$ be the restriction of a vector field $X$ in $\chi^r$ to $N$.

Definition 2 (Sotomayor and Teixeira [85]). We call $\Sigma^r$ the class of all vector fields $X \in \chi^r$ which satisfies the following conditions:

1. all singular points and periodic orbits of $X_N$ are hyperbolic and contained in the interior of $N$;

2. any tangency between a trajectory of $X$ and $\partial N$ is quadratic;

3. $X'$ does not have saddle or tangency connections.

Definition 3 (Sotomayor and Teixeira [85]). Let $G$ be the class of all vector fields $Z = (X, Y) \in \Omega$ which satisfy the following conditions:

1. $X$ and $Y$ are in $\Sigma^r(M^+)$ and $\Sigma^r(M^-)$ respectively;

2. all $S$-singularities and $S$-periodic orbits of $Z$ are elementary;

3. No simple orbit of $Z$ connects a fold point of $Z$ and a critical point of $F(Z)$ or two fold points of $Z$;

4. $Z$ does not have $S$-saddle connections;

5. The graphs of $Z$ are either simple or elementary.
In the definition above, $S$-periodic orbits are periodic orbits on the boundary curve $S$; likewise for $S$-singularities and $S$-saddle connections. The definitions of simple and elementary graphs are beyond the scope of this summary.

The main result of Sotomayor and Teixeira’s article is this:

**Theorem 5** (Sotomayor and Teixeira [85]). Let $Z = (X,Y)$ be in $G$. Then there exists a positive number $\epsilon_0$ such that for any $\epsilon < \epsilon_0$, $Z_\epsilon$ is in $\Sigma'(M)$.

Thus, the regularized vector field $Z_\epsilon$ will also be characterized by hyperbolic fixed/periodic points only, quadratic tangency with its boundary, and will contain no saddle connections. All this means that the regularized field is continuous and also structurally stable (therefore tolerant to perturbations). Both of these properties will prove vital in the development of our memory framework, particularly in the training of the hippocampal analogue.

### 2.3 Training recurrent neural networks

Our goal in constructing a recurrent neural network for the model’s hippocampal analogue is a network we can mould and manipulate to exhibit any prescribed behaviour (or set thereof). We synthesize a dynamical system with the prescribed behaviour set, and then we must cast that system as a RNN somehow.

In fact, Funahashi and Nakamura proved that the additive network of (2.3) can approximate any sufficiently smooth dynamical system [27]. Their article’s first theorem states that any finite-time trajectory of a given $n$-dimensional dynamical system can be approximately realized to arbitrary accuracy by an $(n + m)$-unit continuous-time recurrent neural network. We will expound upon this result and its proof in more detail in a later chapter, for it forms the foundation of our RNN synthesis procedure. Unfortunately, the proof is not constructive; it does not tell us how to find the particular RNN that achieves the desired approximation. However, and as we will show, standard machine learning techniques and some matrix manipulation can be used to train the desired network.
Recurrent neural network training is a difficult problem. Several techniques for the replication of a given dynamical system are known in the literature, of varying complexity and power.

### 2.3.1 Backpropagation through time

Backpropagation through time (BPTT) is a method that adapts the standard backpropagation algorithm, as applied to feedforward networks that do not evolve in time, to recurrent, temporally evolving networks [107, 60]. It is probably the most widely used method for RNN training (with various modifications, such as stochastic sample selection [13] and the use of regularizing terms such as momentum [57]). The standard feedforward algorithm presupposes that connections between neuron units induce a cycle-free ordering of those units, and therefore cannot be transferred directly to RNNs. The solution is to “unfold” the network in time: Identical copies of the RNN are stacked in layers, and connections within the network are redirected to obtain connections between subsequent copies. Each layer represents the same network at a different step in time. The result of this unfolding is a feedforward network amenable to backpropagation. Network weights are identical in and between all copies.

BPTT is a supervised learning technique, meaning that a set of input data and the desired corresponding outputs are provided (this is often referred to as the teacher or training data). In this case the training data consists of a single pair of input-output time series, respectively

\[
\mathbf{u}(t) = [u_k(t)], \quad \mathbf{d}(t) = [d_l(t)],
\]

where \(k = 1, \ldots, K\) indexes the inputs, \(l = 1, \ldots, L\) indexes the outputs, and \(t = 1, \ldots, T\) indexes the discrete timesteps. Note that the output time series \(\mathbf{d}(t)\) represents a prescribed dynamical trajectory through state space. The goal is that when the synthesized RNN is driven by \(\mathbf{u}(t)\), its output units will yield the trajectory \(\mathbf{d}(t)\).

The forward pass of one training epoch involves updating the stacked network starting from the first copy (at layer or time \(t = 1\)) and working upwards. The meaning of \(t\) has
changed from a training instance in the conventional feedforward case to “time.” At each
copy and time $t$, input $u(t)$ is read in and then the internal network state $s(t) \in \mathbb{R}^n$ is
computed from $u(t)$ and $s(t-1)$. Supposing that a subset of the network’s units compute
the output, and representing these by $o(t) \in \mathbb{R}^L$, the error to be minimized is

$$E = \sum_{t=1}^{T} ||d(t) - o(t)||^2.$$  \hspace{1cm} (2.14)

Without going into too much detail, the backpropagation algorithm minimizes the error
(2.14) through a process of gradient descent. Its central task is the computation of
the error gradients $\partial E/\partial w_{ij}$. These are used to modify the network weights such that
the modified network produces the desired output with low error. Unfortunately, standard
and stochastic gradient-descent approaches to RNN training are notorious for the
issue of vanishing/exploding gradients [49]: As gradient information is propagated back
through the layers (representing timesteps) of the network, it tends to shrink or expand
exponentially depending on the magnitude of the largest eigenvalue of the network’s re-
peated weight matrix. In the literature it is generally agreed that memory spans over 20
timesteps are quite difficult to achieve using BPTT [60], although various workarounds
exist, as we shall discuss shortly. Note that time is necessarily discrete in BPTT and
the networks it functions upon, since timesteps must be associated to discrete network
copies.

2.3.2 Real-time recurrent learning

Real-time recurrent learning (RTRL) is another gradient-descent technique that com-
putes a RNN’s exact error gradient, in terms of the weights, at every discrete timestep.
Perhaps the earliest description was given by Williams and Zipser [109]. The technique is
very straightforward: The effect of a weight change on the network dynamics is obtained
simply by differentiating the network equations by the weights. This yields a new $n$-
dimensional discrete-time linear dynamical system with time-varying coefficients where
Chapter 2. Literature review and background

30

the vector

\[
\left( \frac{\partial s_1}{\partial w_{ij}}, \ldots, \frac{\partial s_n}{\partial w_{ij}} \right)
\]

(2.15)

is the dynamical variable. We can compute (2.15) forward in time by iterating the equation for the exact error gradients simultaneously with the network dynamics, yielding the required weight adjustment. Although RTRL is mathematically transparent, the computational cost for each update step is \(O(n^4)\). This renders the scheme practicable only for very small networks on the order of about ten neurons [60].

2.3.3 The extended Kalman filter

Some of the best results in RNN training for system identificatin are obtained using the extended Kalman filter (EKF). This is a state estimation technique for nonlinear systems derived by linearizing the original Kalman filter about the current state estimate. The EKF’s superior performance owes in part to the fact that it is a second-order gradient descent algorithm; this means it uses curvature information from the squared error surface. The pioneering work in using the EKF for RNN training was done by Feldkamp [20]. To apply the filter to the task of estimating optimal weights of a recurrent neural network, the weights are interpreted as the state of a dynamical system. The output \(o(t)\) of an RNN at timestep \(t\) is a function \(h_t\) of the weights \(W(t)\) and the inputs (if they exist) up to \(t\):

\[o(t) = h_t(W(t), u(0), \ldots, u(t)).\]

It is often assumed that the network update contains some process noise which is added to the weights in the form of Gaussian uncorrelated noise \(q(t)\). Thus the state update step for the dynamics of the RNN is given by

\[W(t + 1) = W(t) + q(t)\]

\[o(t) = h_t(W(t)).\]

The state dynamics of this system are static except for shifts induced by noise. The training task takes the form of estimating the state \(W(t)\) from an initial guess and
a sequence of desired outputs $d(0), \ldots, d(t)$ which act as training data. Note that this output sequence comes in the form of a solution trajectory from the prescribed dynamical system as before. The simplified state leads to some simplifications of the classical EKF recursions: the state update step becomes unnecessary and the measurement update becomes

$$K(t) = P(t)H^T(t)[H(t)P(t)H^T(t)]^{-1}$$

$$W(t + 1) = W(t) + K(t)\xi(t)$$

$$P(t + 1) = P(t) - K(t)H(t)P(t) + Q(t),$$

where $K(t)$ is the Kalman gain, $P(t)$ is the conditional error covariance matrix, $H(t)$ is the Jacobian

$$H(t) = \frac{\partial h_t(W)}{\partial W}$$

with respect to the state variables (weights) evaluated at the previous state estimate, $\xi(t) = d(t) - h_t(W(t))$ is the error between observed output and output calculated from the state estimate, and $Q(t)$ is the covariance matrix of the process noise. Obtaining the derivatives in $H(t)$ of the network outputs with respect to the weights is the major challenge with EKF techniques. These derivatives can be computed exactly using RTRL or approximately using BPTT.

The EKF presented here is discrete in time. A continuous-time EKF exists, but its application to the RNN training problem requires continuous-time derivatives for the Jacobian. Unfortunately, BPTT and RTRL do not supply these.

2.3.4 Prediction training and phase-space learning

Tsung and Cottrell [99] developed a vector-field approach to training RNNs that is quite similar to ours from a high-level perspective. Their phase-space learning consists of: (1) embedding a temporal signal to recover its phase-space structure (trajectory), (2) generating local approximations of the underlying vector field about the given trajectory, and (3) approximation of the vector field with a feedforward network. Note that this
method, just like ours, transforms the recurrent network problem into a feedforward one.

In phase-space learning, feedforward networks are produced using prediction training [58]. In that scheme, a feedforward network is given a time window as inputs, which consist of a set of previous points on the trajectory to be learned. Then the inputs are shifted and the network is trained on the next point. The time window essentially manifests a delay-space embedding that reconstructs the phase space of the underlying system. This is an application of Takens’ embedding theorem [92].

Once the feedforward prediction network has learned, it can be treated as recurrent by iterating on its own predictions. Thus, the network is discrete in time.

Although the high-level vector field and feedforward perspective is shared between our methodology and phase-space learning, the networks constructed by the two methods differ fundamentally. The single albeit significant difference of continuous versus discrete time requires an entirely different mathematical approach. Our RNNs are constructed by algebraically manipulating the feedforward network structures into continuous-time recurrent networks. Phase-space learning networks are simply feedforward networks that iterate, in discrete time, on their own outputs.

2.3.5 Reservoir Computing

The other popular and successful technique for RNN training is the concept of reservoir computing (RC). In reservoir computers, a large, randomly recurrently connected sea of neuron units, referred to as the reservoir, feeds forward to a set of output units and is driven by an input signal. The reservoir functions as a dynamical system and must exhibit what is called the echo state property. This relates asymptotic properties of the reservoir dynamics to the driving signal. Intuitively speaking, the echo state principle means that the reservoir asymptotically eliminates any information from initial conditions [60]. Reservoir units feature sigmoidal activation functions while output units feature linear activation functions.

The interesting characteristic of RCs is that the output units may be made to approximate a prescribed dynamical orbit by training only the output weights that feed them (typically by a linear regression process); the random recurrent connections within
the reservoir are fixed. Training of the output weights is again supervised.

Reservoir computers have achieved excellent performance in reproducing prescribed temporal patterns, both periodic and chaotic. However, the reproduction of multiple trained patterns (representing a multiatractor system) from a single network typically requires a switching effect induced by a time-varying input signal; our model does not require such a signal. Furthermore, the randomly connected dynamical reservoir at the core of reservoir computers is, to date, not well understood from a theoretical standpoint [60]. For instance, significant research is currently focused on optimizing a reservoir for a particular task in some automated fashion, rather than by the current method of manual search. It is our hope that the memory model under development herein will help to inform neuroscience regarding principles of neural organization. As such, we sought a mathematical network model that was more theoretically transparent.

2.3.6 Hessian-free optimization and momentum

Backpropagation through time, because it uses only gradient information, is a first-order optimization method. The vanishing/exploding gradients problem can be circumvented by using curvature information, i.e., with second-order optimization techniques like Newton’s method [62]. Unfortunately, exact second-order methods like Newton’s require computation and inversion of the Hessian matrix (of second derivatives), a task that is exceedingly computationally expensive and practically intractable for large neural networks. The answer to this problem is typically to use approximations to the Hessian, such as low-rank, easily invertible matrices. An alternative approach that has achieved great success in recent RNN training is the Hessian-free algorithm of Martens [62, 63]. In Hessian-free optimization, the network objective function is modeled using a local quadratic approximation,

\[ f(\theta + p) \approx f(\theta) + \nabla f(\theta)^T p + \frac{1}{2} p^T B p, \]

where \( \theta \) is the vector of network parameters and \( p \) is the parameter update. The matrix \( B \) quantifies the curvature; in Newton’s method \( B = H \), the Hessian, while approximate
methods use approximations. In Hessian-free, $B$ is never explicitly computed; rather, its matrix-vector products are computed approximately using the conjugate gradient method (first-order).

As mentioned, the Hessian-free method achieves excellent results in RNN training, although later work suggests that similar results could be achieved simply by augmenting stochastic gradient descent techniques with momentum [90]. In momentum techniques, a velocity vector is accumulated over directions of persistent reduction in the objective function, across iterations. The classical momentum update equations are given by

$$v_{t+1} = \mu v_t - \epsilon \nabla f(\theta)$$
$$\theta_{t+1} = \theta_t + v_{t+1},$$

where $\epsilon > 0$ is the learning rate and $\mu \in [0, 1]$ is the momentum coefficient.

Like the methods listed previously, Hessian-free optimization applies only to discrete-time RNNs. Momentum can be applied to any form of gradient descent, whether gradients are backpropagated through time or through feedforward layers.

### 2.3.7 Critiquing the state of the art

The techniques just discussed all have similar drawbacks. BPTT and RTRL typically train on single trajectories sampled from dynamical systems rather than the vector fields that define them. This is natural because those methods are used primarily for system identification; that is, building a model of an unknown system from which one has some measurements. However, the issue is that single trajectories cannot contain complete information about more complex multiattractor systems. This is especially obvious in systems with disjoint attractor basins. Unfortunately methods do not exist to unify systems trained by BPTT or RTRL methods on separate trajectories. Furthermore, long term memory effects often cannot be captured using these training methods because of the vanishing/exploding gradients problem [49]. This issue was resolved with the introduction of the Long short-term memory (LSTM) network, which features multiplicative gates and a linear unit whose gradient is the identity (so that its value can be remembered over
many timesteps) [49]. However, the LSTM network and its variants, (e.g. the gated recurrent unit [15]) operate in discrete time.

Training on multiple trajectories using the EKF or phase-space learning turns out to be a simple matter. For the EKF, this is because, in the RNN case, trajectories are measurements of the state (weights). Thus, multiple trajectories taken from the same system provide multiple measurements of the same underlying state. As long as distinct trajectories are sampled at identical timesteps, we can stack these measurements and augment the algorithm. While generally superior in performance to BPTT and RTRL, the EKF shares their restriction to discrete-time systems because those methods are required for the computation of derivatives. Pearlmutter developed an algorithm for the determination of continuous-time derivatives in RNNs [76] but it relies on a discrete approximation of the network activity, its computational complexity rules out all but the smallest networks, and its performance was not found to be suitably accurate for our purposes.

The only real drawback for phase-space learning and Hessian-free optimization is that they operate in discrete time. However, it is an explicit goal of this research to realize continuous-time dynamical systems as continuous-time RNNs (as modeled by ODEs).

We have already discussed the present drawbacks of reservoir computing.

### 2.3.8 Our training methodology

Our RNN synthesis algorithm functions on the continuous-time networks that are more faithfully representative of the brain and it totally circumvents the difficulty of disjoint attractor basins and distinct dynamical regimes that cannot be captured in single trajectories. One or both of these two issues plague all the methods cited above. Furthermore, our algorithm uses standard backpropagation on shallow feedforward networks, a process that is computationally cheaper and well studied. Because the trained networks are shallow, the vanishing/exploding gradients problem is avoided completely. As a side note, networks generated by our procedure share a structural relation to reservoir computers, and this correspondence might be used to illuminate the dynamical reservoir of those networks and perhaps provide a means for reservoir optimization in future.
Our approach to RNN training is closely related to that used in the Neural Engineering Framework (NEF) of Eliasmith and Anderson [18], which we will discuss in Section 2.4.2. In fact, our method can be considered a special case of the NEF training procedure, and the networks it generates a special class of NEF networks. We will highlight the restrictions we make on our networks, constrained with those available to the NEF, after detailing our procedure in Chapter 4.

### 2.4 Memory models

#### 2.4.1 Associative memory

Associative memory systems are brain-inspired mechanisms capable of storing a set of patterns; the systems can reconstruct stored patterns from cues containing incomplete information about them [10]. This process is what we might call recollection: the triggering of a past memory by a similar current situation. Associative memory systems are often referred to as “content-addressable” memories since a pattern’s content is what serves to locate or trigger it.

The well known Hopfield network [52] is one of the earliest examples of associative memory systems. Its dynamics are governed by an energy function. Network activity relaxes to local minima of this energy function as the neuron activation rule iterates, and prescribed patterns may be set as local minima by a learning rule.

The neurons in a Hopfield net may be binary threshold units or continuous sigmoidal units. In the former case, neurons take on values of either 1 or −1 with the value determined by whether a unit’s input exceeds its threshold. The activation (or update) rule is a difference equation that operates as follows. With $s_i(t + 1)$ the state of unit $i$ at the next timestep,

$$s_i(t + 1) = \begin{cases} 1, & \text{if } \sum_j w_{ij}s_j(t) \geq \theta_i, \\ -1, & \text{otherwise}. \end{cases}$$

(2.16)

Here $s_j(t)$ is the state of neuron $j$ at timestep $t$ and $\theta_i$ is the threshold of neuron $i$. In the latter continuous case neurons may take on any value in the range $[-1, 1]$, with the
update rule being given by a rescaled logistic sigmoid:

\[
\tilde{\sigma}(x) = \frac{2}{1 + \exp(-x/\gamma)} - 1.
\] (2.17)

It is important to note that the rule itself is discrete in time (as opposed to continuous in the case of a differential equation). As such, neuron activations may be performed asynchronously, where each unit is updated one-at-a-time, usually in random order, or synchronously, where all units are updated simultaneously.

The strength of the connection from unit \(j\) to unit \(i\) is given by weight \(w_{ij}\). The possible weight values are typically restricted such that:

\[
w_{ii} = 0, \forall i
\]
\[
w_{ij} = w_{ji}, \forall i, j.
\]

Symmetric weights guarantee that the energy function decreases monotonically as the activation function iterates; if nonsymmetric weights are used then periodic or chaotic behaviour may result [56].

A scalar value referred to as the “energy,” \(E\), is associated with each state of the Hopfield network. It is defined as

\[
E(t) = -\frac{1}{2} \sum_{i,j} w_{ij} s_i(t) s_j(t) + \sum_i \theta_i s_i(t).
\] (2.18)

When units update asynchronously in random order, the quantity \(E(t)\) so defined is guaranteed to either decrease or remain the same. Under the action of repeated updates the network will converge to a local minimum of the energy function.

Binary patterns can be set as local minima of the energy function using a Hebbian learning rule. Introduced in 1949 by Donald Hebb, the Hebbian learning paradigm explains the process of “associative learning,” in which simultaneous activation of two neurons leads to an increase in the synaptic strength between them [46]. This phenomenon is often summarized pithily as “Neurons that fire together wire together.” Hebbian learning
is implemented in the Hopfield network by setting the weights according to

$$w_{ij} = \frac{1}{K} \sum_{k=1}^{K} \mu_{i}^{(k)} \mu_{j}^{(k)},$$

(2.19)

where $\mu_{i}^{(k)}$ represents bit $i$ from pattern $k$ and there are $K$ patterns in total to be stored.

The Hopfield network is rudimentary as a model of human memory. It accounts for the associative effect of pattern retrieval from similar or incomplete cues, but only fixed points can be stored in its state space. This limits memories to static patterns; episodes are not possible. Furthermore, there is no control over basins of attraction in the Hopfield network. We will demonstrate in the next chapter that the basin geometry is completely defined by the placement of fixed points. What this means is that we cannot prescribe what regions of the state space (and what other patterns) are associated to a given memory. It is also the case that unprescribed local minima can exist in the network’s energy function, and thus the network may converge to false memories from certain cues.

In the decades since Hopfield’s work [52], the field of associative memory systems has advanced substantially. A statistical neurodynamical method was proposed by Amari and Maginu [6] that explained dynamical behaviours in the recalling process of associative networks and provided bounds on their memory capacity. Amari and Maginu specifically studied transient behaviour in associative networks as their activity converged on spurious memory patterns, revealing how such patterns become attractors, as well as the peculiar attractor-basin shapes that result for prescribed stored patterns. Their method applies in the limit as the number of neurons tends to infinity.

A pseudoinverse method was developed by Nara et al. [71] for embedding cyclic memories in recurrent neural networks, whereby a chain of several stored patterns is recollected in sequence. This is analogous to storing a limit cycle within the network and represents an advancement from the static patterns of the original Hopfield network. Again, patterns are limited to binary vectors. Using this method, it is not possible to prescribe the paths of transition between stored patterns and the resulting pattern flow is not continuous in any meaningful sense. The authors further demonstrate that by reducing the connectivity (the number of neurons each neuron is connected to), a transition from
limit cycle behaviour to chaos results. In the chaotic regime, there is complex wandering of the network state that visits the basins of multiple or all stored patterns.

A memory network based on [52] that employed chaotic neurons was constructed by Adachi and Aihara [1] to model biological memory dynamics more accurately. That network model is defined by the difference equations

\[ x_i(t+1) = \sigma[\eta_i(t+1) + \zeta_i(t+1)], \]  
\[ \eta_i(t+1) = k_f \eta_i(t) + \sum_{j=1}^{N} w_{ij} x_j(t), \]  
\[ \zeta_i(t+1) = k_r \zeta_i(t) - \alpha x_i(t) + a_i. \]  

(2.20)  
(2.21)  
(2.22)

In (2.20), \( x_i(t) \in (0,1) \) is the continuously-valued neuron output at discrete time \( t \), \( \sigma \) is, again, the sigmoidal activation function, and \( \eta_i(t) \) and \( \zeta_i(t) \) are internal state variables.

State variables \( \eta_i \) track internal network signals, namely feedback and the input from synaptically connected neurons. In (2.21), \( k_f \in (0,1) \) is a feedback decay parameter and \( w_{ij} \) gives the strength of the synaptic connection from the \( j \)th neuron to the \( i \)th neuron.

Variables \( \zeta_i \) track external signals arising from inhibition and stimulation. Each neuron \( i \) is connected to an inhibitory partner that mimics its output with strength \(-\alpha\), while temporally-constant stimulation is induced by \( a_i \).

As in the Hopfield model, the Hebbian learning rule (2.19) is used to embed binary patterns as attractors in the state space. However, the network defined by (2.20)-(2.22) exhibits chaos for certain tunings of the parameters. In this regime, the network's behaviour is more complicated than gradient descent to stored fixed point attractors. In particular, the decay parameters and inhibitory signal build up as the network state rests near or on an attractor, eventually forcing the state away. The network then wanders in state space, passing near or through other attractors. The authors characterize this behaviour as an itinerant search through state space. Stored memory patterns are recalled aperiodically.

More recently, cellular neural networks (CNNs)—a special class of RNN—have been investigated for their applications to associative memory. Various methods exist for
storing memory patterns as stable equilibria in CNNs (see [16] for a review), often relying on specially tuned piecewise continuous activation functions unique to each neuron [10].

In contrast to the networks cited, our associative model (the hippocampal analogue) employs a comparatively simple, canonical RNN formulation with a single activation function; it also offers greater design control over the system dynamics in terms of the variety of attractor-types that can be embedded, the convergence properties of these attractors, and the geometry of their basins. Finally, it relies on a geometric view of state space that we believe is intuitive and transparent.

2.4.2 Brain-inspired models

In this section, we present a review of recent neural-network models of memory. These range from general attractor networks, abstracted somewhat from brain physiology, to variously accurate models of memory formation in the hippocampus, to a full-blown model of several interconnected brain structures that includes the hippocampus.

The Neural Engineering Framework

Underlying his very-large-scale brain simulation [19], Eliasmith developed a unified approach to building and controlling spiking attractor networks. The very capable neural engineering framework (NEF) of Eliasmith and Anderson [18] can be used to construct biologically plausible neural networks exhibiting three classes of attractor behaviour relevant for biological systems: static (fixed), cyclic (periodic), and strange (chaotic). This framework is quite similar in spirit and in some technical details to our method for constructing recurrent neural networks exhibiting arbitrary prescribed dynamics. Importantly, the NEF enables the construction of continuous-time networks.

Three principles form the foundation of the NEF, and these are as follows: (i) neural representations are defined by the combination of nonlinear encoding (neural spiking) and weighted linear decoding over populations of neurons over time; (ii) transformations of neural representations are functions of the variables represented by neural populations and are determined using a weighted linear decoding; (iii) neural dynamics are
characterized by considering neural representations as control theoretic state variables.

In this framework, the activities of a population of neurons encode some input vector according to the nonlinear function describing the neurons’ response. Given this encoding, the original stimulus vector can be approximately recovered by decoding the population activities; this is accomplished by taking the inner product of the activities with a set of decoding vectors. Decoding vectors are determined by a least-squares method. Similarly, approximate transformations of the original stimulus vectors can be defined using a modified set of decoding vectors, referred to as transformational decoders. Transformational decoders may also be computed using a least-squares method, from the definitions of the functions they are meant to represent. In more detail, neuron activities $a_i$ encoding some vector $\mathbf{q}$ are given by

$$a_i(\mathbf{q}) = G_i[\alpha_i(\mathbf{q} \cdot \tilde{\phi}_i) + J_i^{\text{bias}}],$$

where $G_i$ is the nonlinear activation function, $\alpha_i$ is a gain and conversion factor, $J_i^{\text{bias}}$ is a current signal that accounts for background activity, and $\tilde{\phi}_i$ is the linear encoding weight. A transformation $F(\mathbf{q})$ on the variable $\mathbf{q}$ can be performed according to

$$\hat{F}(\mathbf{q}) = \sum_i a_i(\mathbf{q})\phi_i^{F(\mathbf{q})},$$

where $\phi_i^{F(\mathbf{q})}$ is a linear decoding weight and $\hat{F}$ is an approximation of the desired transformation.

The NEF is highly flexible. Both the nonlinear activation function $G_i$ and the transfer functions of its neurons are generic. For example, the activation function can be continuous (sigmoidal) or spiking (LIF), etc., and the neurons may be made to act as biologically plausible low-pass filters having transfer function $h(s) = \frac{1}{1 + s\tau}$. In the low-pass filter case, the function to be approximated becomes $\mathbf{y} = \tau F(\mathbf{q}) + \mathbf{q}$ rather than $\mathbf{y} = F(\mathbf{q})$.

To build desired attractor networks using the NEF, the vector variables represented by a neural population are viewed as control theoretic state variables. The desired dynamics then define the standard matrices in the control formulation. The linear encoders $\tilde{\phi}_i$ are
set randomly, then the set of (mean squared) optimal linear decoders \( \phi_i^{F(q)} \) is computed in the standard way by minimizing the mean squared error. This means that, contrary to the typical backpropagation used in training neural networks, error is evaluated only at the output, based on the linear decoders; it is not propagated back to assign blame to the randomly-set linear encoders. In some ways this is advantageous: because errors need not be propagated through the activation function \( G_i \), this function need not be differentiable. This of course allows for a wider range of activation function.

As mentioned, there are strong connections between the NEF and our RNN training technique, although the two methods spring from formalisms. We take as our starting point a theorem from the continuous-time RNN literature that is not employed by the NEF authors. Through it, we arrive at a procedure and a class of networks which can be viewed as a special case of the NEF. However, by starting from this theorem and tracing a different route to the end procedure, we show that the special case of networks we utilize is governed by theoretical bounds on its performance. We also provide an analysis that yields helpful insights into certain parameters of network construction that is not present in NEF work. In particular, we will use our networks’ simple form to develop an algebraic approximation for their hidden units. This facilitates network analysis such as the computation of network potential functions, and makes explicit that the networks simulate smaller-dimensional systems.

It is important to note, though, that our framework by no means obviates the NEF. The NEF, like our method, enables the construction of RNNs that realize arbitrary dynamical systems. We will make the mathematical connections between the two approaches explicit after detailing our RNN framework in Chapter 4.

Finally, we point out that the NEF seems to deal with multiattractor systems by interconnecting multiple attractor networks trained individually, whereas our framework can be used to build multiattractor vector fields and then simulate them within a single trained network. The significance of this difference is debatable.
The hippocampal model of Samsonovitch and Ascoli

Samsonovich and Ascoli developed a model that casts episodic memory retrieval in the same framework as physical navigation, basing both these processes in the hippocampus [79].

It is now widely accepted that in rodents and primates, particular hippocampal cells, called *place cells*, selectively fire at a high rate when the animal is located in a particular spatial domain—this domain is known as the place field of the place cell. The system of place fields and connections between place cells forms an internal cognitive map of an environment.

Likewise, there is substantial evidence going back to the psychology patient H.M. that the hippocampus is responsible for the formation of long-term autobiographical (episodic) memory. More recently, McClelland *et al.* [64] proposed their complementary learning systems theory. Its hypothesis is that the hippocampus provides a mechanism for rapid acquisition of new information from the environment, and that following the initial acquisition the hippocampus ‘teaches’ this new information to the neocortex for long-term storage. The teaching process, which may occur during sleep, must overcome the problem of catastrophic forgetting whereby old memories are effectively overwritten by new ones (more on this later). According to this theory, the memory of an episode is retrieved in the neocortex by reactivating a hippocampal pointer to the episode.

Samsonovich and Ascoli posit that the hippocampus’s navigational capabilities have been appropriated for memory retrieval. They make the assumption that an episodic memory trace may not be retrievable without some information about the original context of the episode, this context not being associated with the hippocampal pointer. They do not provide support for this assumption beyond a thought experiment (and these are always disturbingly circular when it comes to cognition), but they do cite neuroscientific evidence that the hippocampus is involved in the process of context reconstruction. Their main thesis is that the hippocampus reconstructs a particular episode’s context through a search for a nearly-shortest path within the space of remembered contexts. Specifically, if one thinks of a set of episodic memories and their connections as an abstract graph,
then a rule that ensures retrieval of episode X, given an initial state of sensory input Y, is to follow a connected path from Y to X. This is where the navigation comes in.

Their model hippocampal network operates as follows. It consists of CA3 and CA1 units, each representing groups of neurons. Each small neighborhood in the environment has a unique representation in the system being associated with a particular CA3 unit and a particular CA1 unit. The environment, which in their model may represent a literal physical environment or the space of contexts, is assumed to be explored extensively in order to build up a set of place cell connections. This set is built up as follows. Each visit, during exploration, to a particular location results in a strong activation of the corresponding CA3 and CA1 model units, and at various arbitrary moments during exploration the system pauses and the current location is taken as a potential future goal. At this point, the activity mode changes to reactivation of recently active CA3 place cells. Furthermore, CA1 cells representing the current location are also reactivated, so CA3 place cells become associated with the selected CA1 cell whose place field represents a potential future goal—the strength of the associations is proportional to the recency of a place-cell firing during exploration. After the conclusion of such an associative learning event, the system resumes random exploration of the environment, until another arbitrary potential goal is selected, and the associative process repeats. Thus, if the exploration phase continues until a preselected goal location is reached, the CA3 model units acquire weighted connections to the CA1 representation of the goal. The network of place cells then provides a capacity for finding a short path to this goal location from any given point in the environment: At each location, several randomly generated possible local moves are explored, and the move that produces the strongest excitation of the place cell associated with the goal is selected and performed. This process is repeated until the goal is reached. As mentioned, the authors believe that the hippocampus reconstructs an episode’s context through a search for a nearly shortest path, and this is how that path is approximated.

Samsonovitch and Ascoli present simulation results for a two-dimensional maze navigation problem. After random exploration of the environment the model is tasked with reaching a preselected goal. The typical resultant trajectory steered clear of dead ends,
did not include loops, and followed a path with length on the same order as the shortest possible path (i.e., the paths were nearly optimal).

Also presented is a simulation in which the hippocampal network learns a path of transitions between attractors in a paired, randomly connected attractor network. As this network evolves through time, certain transitions occur between stored attractors in a fashion similar to that observed in Adachi and Aihara’s chaotic network, and the attractors are modelled as locations within the hippocampal network and the transitions as paths between locations. This is claimed to be a proof-by-example of their context reconstruction concept.

Hattori’s complementary-learning-systems model

Hattori [43] also developed a memory framework based on the complementary learning systems theory, although this models both the hippocampus and the neocortex explicitly. The hippocampus is represented by a chaotic recurrent neural network akin to that of Adachi and Aihara, while the neocortex is represented by a multilayer feedforward network. Hattori’s specific aim was a model that overcame the problem of catastrophic forgetting. Catastrophic forgetting is the process by which old information tends to be destroyed by the storage of new information in neural networks.

Hattori’s hippocampal network uses Hebbian learning on input patterns for the formation of attractors. This proceeds according to the same rule used by the Hopfield network; however, the rule is modified with a term that induces a gradual forgetting of old patterns over time. The rule is given below, where $\gamma \in [0,1]$ represents the forgetting term and $\mu_i^{(k)}$ represents the $i$th element of memory pattern $k$.

$$w_{ij}(t + 1) = \gamma w_{ij}(t) + \mu_i^{(k)} \mu_j^{(k)}.$$  \hspace{1cm} (2.23)

Once patterns have been stored in the network, these patterns are extracted and passed on to the neocortical network by giving the hippocampus a random input and observing the chaotic sequence of recalled patterns. As was shown by Adachi and Aihara, stored patterns are recalled for long sequence periods, relative to arbitrary or transition patterns.
The extracted patterns are then learned by the multilayer network by supervised backpropagation. Catastrophic forgetting is overcome by training the hippocampal patterns along with a population of pseudopatterns extracted from the neocortical network. The pseudopattern concept was first proposed by Robins [77] and works as follows. The neocortical network is given a random input pattern which is then fed forward to produce an output. Each of these input-output pairs is used as an example for supervised training of the network to preserve what it already knows. The idea is that the pseudopatterns sample the existing underlying function of the network, and reinforce that function with enough training examples (here ‘function’ refers to the mathematical function realized by the feedforward network).

Hattori provides simulation examples demonstrating his dual-network model is able to store and recall correctly eight patterns representing alphabet characters. These eight patterns were learned sequentially in batches of two such that a conventional feedforward network would suffer catastrophic forgetting. Hattori therefore claims superior performance of his model.

Hattori later extended this model to represent hippocampus physiology more accurately [44]. The extended model includes discrete McCulloch-Pitts neuron layers representing the entorhinal cortex and the dentate gyrus that feed into region CA3 of the hippocampus. The CA3 region is modeled with the chaotic network of Adachi and Aihara [1] as before. This more complicated model can store a greater number of patterns than its predecessor.

The associative hippocampal model of Rolls

Rolls [78] developed a quantitative computational theory of the hippocampus as an episodic memory system based on association processes. This theory runs counter to that of McClelland et al., proposing that the CA3 network of the hippocampus acts as an associative memory which enables episodes to be formed and stored there. Subsequently, the extensive recurrent connectivity characteristic of this site allows for the retrieval of a whole representation to be initiated by the activations of some small part of the same representation. In this sense, the theory mimics the operation of a Hopfield
network. Rolls’s hypothesis is that because CA3 operates effectively as a single network, it allows arbitrary associations between inputs originating from different parts of the cerebral cortex. Thereby, CA3 enables rapid, one-trial associations between any spatial location (represented by place cells) and an object or reward. The theory is extended to associations between time and object or reward to implement temporal ordering memory. These can be used to represent episodes of a sort, but not in the sense of a continuous flow as in our memory model.

**Vineyard et al.’s cortical-hippocampal architecture**

Vineyard et al. [103] built a neurophysiologically faithful cortical-hippocampal architecture based on variants of adaptive resonance theory (ART) neural networks [14]. Their large-scale model putatively captures qualitative memory phenomena such as autoassociation and recall, although their publications are light on detail. To give a synopsis of the various components of the system: (i) The modelled entorhinal cortex (EC) facilitates the convergence of multiple sensory streams by uniting dorsal and ventral visual streams; (ii) this combination is then received by a model of the dentate gyrus (DG) region which performs pattern reduction and separation. The DG is represented as a series of winner-take-all fuzzy-ART modules which effectively yield sparse encodings of differing input patterns. Next, (iii) resultant unique representations from DG are autoassociated in a model CA3, which is a process similar to Hebbian learning in the Hopfield network, which binds related memories together. Finally, (iv) the major output regions of the hippocampal loop, a conjoined representation of both CA1 and the subiculum (also instantiated as an ART module), create a temporal sequence linking of CA3 encodings back to the original EC representation. The net result of all this computational effort is that the model learns to associate images of distinct faces when they are shown, individually, with images of the same house. Thus is a set of people associated with a unique location.
Hasselmo’s episodic memory model

Finally, Hasselmo developed a model of episodic memory that is similar to ours in its capacity to encode and retrieve temporally extended episodes [41, 42]. It is based specifically on navigation in the hippocampus, and the episodes it stores are spatial trajectories through an environment. Such trajectories are stored by encoding associations between velocity and location.

Hasselmo’s model is based on the physiology of the rat and several neuron cell types present in the mammalian brain. These include head direction cells in the entorhinal cortex and postsubiculum, grid cells in the entorhinal cortex, and place cells in the hippocampus. Head direction cells show firing when an animal is facing in a narrow range of directions. In the model, this direction information is combined with speed information to determine the animal’s velocity. Shifts in the relative phase of entorhinal grid cells perform the integration of velocity necessary to reconstruct a path from velocity and location information. Hipocampal place cells, which fire selectively when the animal is in a specific location, are used to achieve the read-out of spatial location to be combined with velocity information.

The model encodes a trajectory by sequentially encoding associations between place cells and velocity, which occurs through modification of the synaptic connections between active hippocampal place cells and active head direction cells. Synaptic modification follows the Hebbian learning rule and its characteristic outer product form that has been described earlier. For example, the connections between head direction cells and place cells, $W_{PH}$ are modified according to the equation

$$W_{PH}(t + 1) = \frac{1}{2}(W_{PH}(t) + p(t)^T h(t)),$$

where vector $p(t)$ represents the population of place cells and $h(t)$ the population of head direction cells. Connections between all cell groups are formed analogously. Hebbian associations can also be made between sensory stimulus patterns and certain locations in an analogous way, so that encoded trajectories can be cued and retrieved from the current location of the animal or an environmental stimulus that is sensed at that location.
Chapter 2. Literature review and background

Retrieval activity in the model is given by the following equation:

\[ p(t + \delta t) = W_{PG}g \left( \int_0^t W_{HP}p(\tau) d\tau \right). \]

To summarize: place cell activity is transformed through the matrix \( W_{HP} \) to drive activity in the head direction cells. The grid cell function \( g(t) \) integrates this activity to update the grid cell phase, whereupon grid cell activity is transformed by the matrix \( W_{PG} \) to modify the states of the place cells.

Hasselmo’s model achieves impressive results in terms of the encoding and retrieval of spatial trajectories. It also enables storage of trajectories that differ from the episodic traces of our model in at least one key way: the trajectories can intersect with themselves. This is not possible in our model because trajectories arise from dynamical systems (although a dynamical trajectory in a higher-dimensional or phase space can have intersections in lower-dimensional projections).

Hasselmo’s model is also capable of associating sensory stimuli to points along its episodic trajectories, which is related to the function achieved by our cortical model (the deep autoencoder). However, the model has no capacity for transforming incoming sensory stimuli (as we do via layer-wise depth in dimensionality reduction and reconstruction). This will, in practice, limit the representational power of the model since only raw sensory vectors are stored. These can be very high-dimensional and difficult to store through Hebbian learning processes, as earlier discussion on the Hopfield network made clear.

2.5 Research gaps

The major gaps in current episodic memory research that this work intends to fill are as follows. First: with the exception of Eliasmith’s NEF, existing techniques for RNN training apply to discrete-time networks, or are incapable of accurately approximating the rich multiattractor systems that must form the basis of episodic memory (if it can indeed be modelled as an attractor-based dynamical system). Our training approach will
operate in a similar vein to the NEF, on a restricted class of RNNs.

Second: with the exception of Hasselmo’s model, existing memory models capture some of the associative component of episodic memory but they do not directly store episodic traces in the sense of dynamical trajectories. In the case of Hasselmo’s model, there is limited capacity for the retrieval of high-dimensional, animated episodes that are continuous in time. This is because the learning paradigm is strictly Hebbian and lacks layer-wise depth. Our model overcomes this difficulty with its cortical module.

The present research will yield a dual-network cortical-hippocampal memory model inspired by those of Hattori [43], Rolls [78], and Vineyard *et al.* [103]. The hippocampus is modelled as a continuous-time associative RNN and the cortex by a deep autoencoder. In order to realize the desired function of the hippocampal network, we develop a novel RNN training algorithm. This algorithm enables the synthesis of RNNs that approximate continuous-time multiattractor dynamical systems to within theoretical upper bounds. Our complete episodic memory model enables the storage and retrieval of continuous-time animated episodes based on compressed trajectories in hippocampal state space. These movie-like memories represent experienced episodes from the past.
Chapter 3

State-space sculpting for the synthesis of vector fields

3.1 State-space structure of the Hopfield network

We have briefly discussed the Hopfield network as an early model of associative memory. Now we perform a deeper analysis of its state-space structure, with an emphasis on the embedding of fixed points, their basins of attraction, and the separatrices that divide them. It was this investigation that originally motivated and inspired our approach to synthesizing a RNN’s state space. The analysis is mathematically straightforward but reveals the rigid geometry underlying Hopfield attractor dynamics.

The Hopfield network as given in Chapter 2 is not particularly amenable to analysis. The logistic sigmoid function, $\sigma$, prevents obtaining closed-form solutions for fixed points, attractor basins, and so on. Therefore we will assume that the function’s steepness parameter, $\gamma$, is small, so that the sigmoid can be approximated with a discontinuous step-like function $\tilde{\sigma}$:

$$
\tilde{\sigma}(x) \approx \begin{cases} 
1 & \text{for } x > 0, \\
0 & \text{for } x = 0, \\
-1 & \text{for } x < 0.
\end{cases} \quad (3.1)
$$
A rescaled sigmoid with $\gamma = 0.05$ is graphed in Figure 3.1 to illustrate the validity of this approximation.

Now to the analysis. We will write the network’s update rule using the vector shorthand

$$s_i(t+1) = \tilde{\sigma}(w_i^T s(t)),$$

where $s_i(t+1) \in [-1,1]$ is the continuously-valued output of neuron $i$ at discrete time $t+1$, $s(t) \in [-1,1]^n$ is the vector of all $n$ neuron states at time $t$, $w_i$ represents the $i$th column of the synaptic weight matrix $W = [w_{ij}]$, and we take the inner product of $w_i$ with $s(t)$.

### 3.1.1 Fixed points and separatrices

To begin, we show that embedding a single memory in the Hopfield network generates a fixed point at that location. This first result follows the analysis of Keeler [56]. The Hopfield weight matrix is given by $W = [w_{ij}] = \frac{1}{K} \sum_{k=1}^{K} \mu_i^{(k)} \mu_j^{(k)}$, which in the single-memory case reduces to $W = [\mu_i^{(1)} \mu_j^{(1)}]$. Dropping the time dependence and the pattern indexing for clarity, we have

$$w_i^T s = \sum_{j=1}^{n} \mu_i \mu_j s_j.$$
Chapter 3. State-space sculpting for the synthesis of vector fields

If the network state is equal to the embedded memory then \( s_j = \mu_j \) so we have

\[
\mathbf{w}_i^T \mu = \sum_{j=1}^{n} \mu_i (\mu_j)^2 \nonumber
\]

\[= n\mu_i \]

since \( \mu_j = \pm 1 \). Here \( n \) is the number of neurons. The positive factor \( n \) multiplies \( \mu_i = s_i \) to yield a larger number with the same sign as \( s_i \). When this is fed as an argument to the approximate sigmoid function, as in \( s_i(t + 1) = \tilde{\sigma}(\mathbf{w}_i^T \mathbf{s}(t)) \), the result is clear: \( s_i(t + 1) = s_i(t) \), and we have a fixed point. In particular, we can see in the one-memory case that the stored memory vector \( \mu \) is an eigenvector of the weight matrix \( \mathbf{W} \) with eigenvalue \( n \), since \( n \) is constant.

We take a moment to interpret the action of the sigmoid in a manner that will be salient further on. If we assemble the sigmoid’s arguments (call them \( y_i = \mathbf{w}_i^T \mathbf{s} \)) into a vector \( \mathbf{y} = \mathbf{W}s \), then \( \mathbf{y} \) represents a point in \( n \)-dimensional Euclidean space. Application of the sigmoid at each of the \( n \) neurons pushes this point to the hypercube vertex that lies in the same quadrant as \( \mathbf{y} \). This is shown schematically in Figure 3.2. Note that the hypercube vertices are the points consisting of coordinates \( \{\pm 1\}^n \) since we have rescaled; note also that, rather than “quadrant” we should technically say “2\(^n\)-ant” (a generalization of a quadrant to \( n \)-dimensional space), but we will continue to use “quadrant” for the sake of style. The dashed circle in the figure represents the region in which values are too small for the sigmoid to push them all the way to a vertex. This region is just the origin for the discontinuous approximation (3.1) to \( \tilde{\sigma} \).

Thus the sigmoid can be thought of as dividing \( n \)-dimensional space into \( 2^n \) attractor basins, with the attractors being fixed points at the vertices of the rescaled hypercube. The hyperplanes separating quadrants are invariant under the action of the sigmoid; in dynamical systems terminology we would call them separatrices.
Extending the previous analysis to the case of multiple stored memories, we find that

\[
\mathbf{w}_i^T \mathbf{s} = \frac{1}{K} \sum_{k=1}^{K} \sum_{j=1}^{n} \mu_i^{(k)} \mu_j^{(k)} s_j
\]

\[
= \frac{1}{K} \sum_{k=1}^{K} (n - 2H_{s,\mu^{(k)}}) \mu_i^{(k)}.
\]

Here, \(H_{s,\mu^{(k)}}\) represents the Hamming distance from the network’s current state \(\mathbf{s}\) to the stored memory vector \(\mu^{(k)}\). This expression assumes that the network state is at one of the vertices of the scaled and centred hypercube graph, meaning it consists of all \(\pm 1\)’s (a safe assumption given the steepness of the sigmoid we are using); if this is not the case then the negative factor in the expression above is smaller than the Hamming distance value.

The dominant term in the sum for \(\mathbf{w}_i^T \mathbf{s}\) is given by the memory vector closest to the network’s current state, i.e., with the smallest Hamming distance. This term pushes the argument of the sigmoid in the direction of \(\mu_i^{(k)}\) as in the one-memory case, assuming that \(H_{s,\mu^{(k)}} < \frac{n}{2}\). However, its push may be drowned out by the collective push of the smaller
terms from other memories. This is a possibility even when the network’s current state is identically one of the memory vectors $\mu^{(k)}$. Although this will give the largest possible push $n$ towards $\mu^{(k)}$ since $H_{s,\mu^{(k)}} = 0$, that push could still hypothetically be drowned out by the effect of other stored memories. This means that for a memory $\mu^{(k)}$ to act as a fixed point, there cannot be too many other stored memories, particularly if they are similar to $\mu^{(k)}$. Furthermore, for one of these fixed points to be attracting, the network state must be relatively near to it so that $H_{s,\mu^{(k)}}$ is small. We will properly analyze fixed points’ basins of attraction in the next subsection.

In the more general multiple-memory case the vectors $\mu^{(k)}$ are not necessarily eigen-vectors of $W$. Even if they happen to be fixed points of the system, all this means is that $w_i^T \mu^{(k)} = \alpha_i \mu_i^{(k)}$ where the positive factor $\alpha_i$ is given by

$$\alpha_i = \frac{1}{K(\mu_i^{(k)})} \sum_{p=1}^{K} (n - 2H_{s,\mu^{(p)}})\mu_i^{(p)}.$$  

This term, which would function as an eigenvalue if it were constant, depends on the index $i$ and is in general not identical for all neuronal units, meaning that $\mu^{(k)}$ is in general not an eigenvector.

Now we shall investigate the two-memory case in greater detail, changing our notation slightly for greater clarity. Suppose two memory vectors, now given by $a = (a_1, a_2, ..., a_n)^T \in \mathbb{R}^n$ and $b = (b_1, b_2, ..., b_n)^T \in \mathbb{R}^n$, are stored in the network using the Hebbian rule so that $w_{ij} = \frac{1}{2}(a_ia_j + b_ib_j)$ and in full matrix form, $W = \frac{1}{2}(aa^T + bb^T)$. First we show that the hyperplane equidistant from these memories is invariant. The normal vector for this plane is given by $n = a - b$ and we can deduce that it must pass through the origin if the memory vectors are vertices of the hypercube graph. Now we assume that the current network state lies in this plane, representing the state by $p = (p_1, p_2, ..., p_n)^T$. Hence,

$$Wp = \frac{1}{2}(aa^T + bb^T)p$$

$$= \frac{1}{2}a(a^Tp) + \frac{1}{2}b(b^Tp).$$
However, since $p$ is in the plane we have

$$0 = n^T p$$
$$0 = (a - b)^T p$$
$$0 = a^T p - b^T p$$
$$a^T p = b^T p.$$ 

which when substituted into the expression above gives

$$Wp = \frac{1}{2}(a^T p)(a + b)$$

for the argument of the sigmoid. Looking at this argument on a neuron-by-neuron basis, we have $\frac{1}{2}(a^T p)(a_i + b_i)$. Assuming this value is large enough to be pushed to $\pm 1$ (or is identically $0$) we have several cases enumerated below, wherein $d_i(\cdot, \cdot)$ gives the distance between its arguments measured in dimension $i$, and $s_i$ represents the state of neuron $i$ after activation.

Case 1: $a^T p > 0$, $a_i = b_i \Rightarrow s_i$ is pushed to $a_i = b_i$, $d_i(s_i, a_i) = 0 = d_i(s_i, b_i)$;

Case 2: $a^T p < 0$, $a_i = b_i \Rightarrow s_i$ is pushed to $-a_i = -b_i$, $d_i(s_i, a_i) = 2 = d_i(s_i, b_i)$;

Case 3: $a^T p > 0$, $a_i = -b_i \Rightarrow s_i$ is pushed to $0$, $d_i(s_i, a_i) = 1 = d_i(s_i, b_i)$;

Case 4: $a^T p < 0$, $a_i = -b_i \Rightarrow s_i$ is pushed to $0$, $d_i(s_i, a_i) = 1 = d_i(s_i, b_i)$;

Case 5: $a^T p = 0 \Rightarrow s_i$ is pushed to $0$, $d_i(s_i, a_i) = 1 = d_i(s_i, b_i)$.

In each case the subsequent network state is equidistant from memories $a$ and $b$ and therefore lies in the plane normal to $n$.

What if the argument $Wp \approx 0 \neq 0$, so is not large enough to be pushed to a vertex of the hypercube? In this case, we have $s = \tilde{\sigma}(\frac{1}{2}(a^T p)(a + b))$ and taking the inner product of $s$ with $n$ yields

$$n^T s = (a - b)\tilde{\sigma}(\frac{1}{2}(a^T p)(a + b)),$$
which we cannot evaluate cleanly in this form. In the vicinity of \( y = 0 \), we will approximate the logistic sigmoid \( \tilde{\sigma}(y) \) by a linear function passing through the origin, \( i.e., \) \( \tilde{\sigma}(y) \approx Ly \). At the very least, we can be sure that \( \tilde{\sigma} \) makes positive points more positive and negative points more negative in this vicinity, as long as the steepness is sufficient.

Using our linear approximation, we have

\[
\begin{align*}
n^T s &= (a - b)^T \tilde{\sigma}(\frac{1}{2}(a^T p)(a + b)) \\
&= (a - b)^T \frac{L}{2}(a^T p)(a + b) \\
&= \frac{L}{2}(a^T p)(a^T a - b^T b).
\end{align*}
\]

But since memory states \( a \) and \( b \) are vertices of the hypercube graph centred at 0, they have the same magnitude so that \( a^T a = b^T b \). The result is that \( n^T s = 0 \), meaning \( s \) remains in the plane. Therefore the hyperplane equidistant from the two memory locations is invariant. This plane is a *separatrix* just like those mentioned previously. As we will see in the next subsection, it separates the basins of attraction of memory vectors \( a \) and \( b \).

Next we show the existence of a fixed point in this plane at the midpoint between embedded attractors, \( i.e., \) at \( p = \frac{1}{2}(a + b) \). For the sigmoid’s argument we now have

\[
\begin{align*}
W_{\frac{1}{2}}(a + b) &= \frac{1}{2}(aa^T + bb^T)\frac{1}{2}(a + b) \\
&= \frac{1}{4}a(a^T a + a^T b) + \frac{1}{4}b(b^T b + b^T a) \\
&= (n - H_{a,b})\frac{1}{2}(a + b)
\end{align*}
\]

where \( H_{a,b} \) is the Hamming distance between memories \( a \) and \( b \). If \( H_{a,b} < n \) then after sigmoidal activation we have \( s = (a + b)/2 \), so the midpoint is fixed. If it so happens that \( H_{a,b} = n \) then \( s = 0 \) but \( 0 \) is the midpoint in this case. Note however that \( 0 \) is always fixed under the dynamics.

We can also argue that this fixed point is attracting for points in the plane. Looking back at the set of cases for when \( Wp \) is sufficiently large, we see that the point to which
p is being pushed is precisely the midpoint of a and b. It was just shown that when \( Wp \approx 0 \) the point \( p \) is pushed to some other point in the plane, and we now show that this new point (call it \( p' \)) will be pushed eventually to the midpoint as well. Using the linear approximation to \( \tilde{\sigma} \), \( p' \) is given by

\[
p' = (\tilde{\sigma}(w_1^T p), \ldots, \tilde{\sigma}(w_n^T p))^T
\]

\[
= (L(w_1^T p), \ldots, L(w_n^T p))^T
\]

\[
= \frac{L}{2} (a^T p)(a + b).
\]

To use this point as an argument in the sigmoid once more, we premultiply by \( W \):

\[
Wp' = W\frac{L}{2} (a^T p)(a + b)
\]

\[
= \frac{L}{2} (a^T p)W(a + b)
\]

\[
= \frac{L}{2} (a^T p)\frac{1}{2}(aa^T + bb^T)(a + b)
\]

\[
= L(n - H_{a,b})\frac{1}{2} (a^T p)(a + b)
\]

\[
Wp' = L(n - H_{a,b})Wp.
\]

If \( H_{a,b} < n \) and we assume sufficient steepness of the sigmoid so that \( L(n - H_{a,b}) > 1 \), then the argument of the sigmoid increases in magnitude from one activation to the next. This means that eventually it becomes sufficiently large that the network state is pushed to the midpoint of a and b. If \( H_{a,b} = n \), then the network state is pushed to 0, but again \( H_{a,b} = n \) means 0 is the midpoint of a and b. Finally, we can surmise that if the origin is not the midpoint between attractors, the fixed point there must be repelling.

### 3.1.2 Basins

The invariant hyperplane equidistant from memory vectors a and b separates the Hopfield network’s phase space into two disjoint subsets – one containing a and the other containing b. Under the right circumstances (\( n \) sufficiently large compared to \( K \), and the Hamming distance between memories sufficiently large) these disjoint subsets are
precisely the basins of attraction of the memory vectors they contain. We show this now for the simplest case, two memories in a 2-neuron network, as illustrated in Figure 3.3.

From the previous section we know that $w_i^Ts = \frac{1}{2}(a^Ts)a_i + \frac{1}{2}(b^Ts)b_i$. In particular,

$$a^Ts = \sqrt{2}\cos \theta_{a,s},$$

$$b^Ts = \sqrt{2}\cos \theta_{b,s},$$

since $||a|| = ||b|| = \sqrt{2} > 0$. Furthermore, from the diagram $\theta_{a,s} > \frac{\pi}{2}$ implies $0 > \cos \theta_{a,s}$ and $\theta_{b,s} < \frac{\pi}{2}$ implies $0 < \cos \theta_{b,s}$. In fact, since $a$ and $b$ are diametrically opposed in this case the angle between them is $\pi$ and so $\cos \theta_{a,s} = -\cos \theta_{b,s} = \eta$. This means that $w_i^Ts = -\frac{1}{2}(\eta\sqrt{2})a_i + \frac{1}{2}(\eta\sqrt{2})b_i$, but again since $a$ and $b$ are diametrically opposed, $a_i = -b_i$ and $w_i^Ts = (\eta\sqrt{2})b_i$.

The result is that $s$ is pushed to the quadrant containing $b$ and then the sigmoid function does the rest. Clearly if $s$ lies on the other side of the invariant plane (line in this case) then it will be pushed to $a$, so, the basin of $a$, $\beta_a$, is the set of all points on $a$’s side of the invariant plane, while the basin of $b$, $\beta_b$, is the set of all points on $b$’s side of the invariant plane.

We chose diametrically opposed memory vectors because closer memories in a 2-
neuron network interfere with each other to generate spurious attractor states according to the mechanism detailed earlier. In Figure 3.4 we numerically sample the basins of two memory vectors in a 3-neuron network (plotted in the top corners as spheres) which are not diametrically opposed. This is to give an indication that the basin result generalizes without having to present much tedious math (cf. Figure 3.6 in the next section, which plots basins for three memories in a 3-neuron network).

It should be noted that in general there exist subsets of $\beta_a$ and $\beta_b$ which do not converge to $a$ and $b$, respectively. For instance, in the situation displayed in Figure 3.4 it can be shown that the locus of points where $\theta_{a,s} = \frac{\pi}{2}$ (some of this locus lies within $\beta_a$) does not converge to $a$; this comes from the fact that $a^T s = 0$. However, this locus has Lebesgue measure zero in the Euclidean state space so it is sampled with probability zero. The existence of this locus nevertheless highlights a shortcoming of the Hopfield network’s attractor dynamics.

### 3.1.3 On the action of $\sigma$ and $W$

As we saw earlier, we can interpret the logistic sigmoid function on Euclidean $n$-space as a system of $2^n$ attractors. In particular, $\sigma$ maps all of $\mathbb{R}^n$ except for an open ball centred
at the origin to the vertices of the hypercube graph, $Q^n$:

$$\sigma : \mathbb{R}^n \setminus B(\{0\}, r) \rightarrow Q^n,$$

where the ball’s radius $r$ depends on the steepness parameter. Likewise, our approximation $\tilde{\sigma}$ maps all of $\mathbb{R}^n$ except the origin to the vertices of the scaled and translated hypercube graph centered at the origin, $Q^n_0$:

$$\tilde{\sigma} : \mathbb{R}^n \setminus \{0\} \rightarrow Q^n_0.$$

This action was depicted in Figure 3.2, and clarifies what the function of the linear transformation $W$ should be: $W$ maps all vectors in the basin of a memory to the quadrant containing that memory. That is,

$$W : \{s \mid s \in \beta_{\mu(k)}\} \rightarrow q^n_{\mu(k)},$$

where $q^n_{\mu(k)}$ is the quadrant containing $\mu(k)$. We say this is what the action ‘should be’ because the Hopfield network’s attractor dynamics can fail under various conditions.

Although the memories $\mu^{(k)}$ are not eigenvectors of $W$ in general, what we can guarantee about the action of $W$ on $\mu^{(k)}$ is this: If the memory $\mu^{(k)}$ is a fixed point of the system, then the transformation imposed on $\mu^{(k)}$ by $W$ is within $q^n_{\mu(k)}$. Otherwise the network state would be pushed to another quadrant by the sigmoid.

This interpretation allows us to view the Hopfield network, and indeed any network using sigmoidal neurons, as a series of distinct attractor systems. First, the weight matrix $W$ pushes some set of points (the basin) to the appropriate quadrant, then $\sigma$ pushes points in each quadrant to the corresponding hypercube vertex. The combination of these two dynamical systems is depicted schematically in Figure 3.5, which shows the action of $W$ and $\sigma$ on regions of phase space delimited by the various separatrices.

The division of labour between $W$ and $\sigma$ can simplify the analysis of, for instance, Adachi and Aihara’s chaotic network [1]. Therein, $W$ is trained the same way and performs the exact same function as in the Hopfield net; $\sigma$, which is augmented with
The combined actions of $\sigma$ and $W$ generate Hopfield attractor dynamics.

Inhibitory parameters, can be viewed as nested series of two dynamical systems. The first of these, $W$, injects chaos into the network by pushing points to a new quadrant that depends on the current quadrant and the time history of the network state. The second pushes points to the new quadrant’s hypercube vertex.

### 3.1.4 Summary

The above analysis helps to reveal how the Hopfield network separates embedded attractors. Specifically, a codimension-1 surface separates the basins of 2 embedded attractors, and this surface, a separatrix, contains a fixed point that is attracting for points on it. We have only treated in detail the case of 2 embedded attractors; the situation becomes more complicated as more memories are added, as is depicted in Figure 3.6. There are plotted the basins of three separate attractors embedded in the state space, along with an approximation of the separatrix between them. The separatrix has thickness because of how the plot was generated. Points that did not converge to memory locations within a given (low) number of network timesteps were considered not to lie in the basins. This is necessary because the exact separatrix has Lebesgue measure zero and therefore zero probability of showing up in a random sample.

As can be seen, basins necessarily encroach on one another and pairwise separatrices combine as memories are added. This is because the state space of the Hopfield network
Figure 3.6: Plot of attractor basins and separatrix (black) for three memories embedded in a Hopfield network.

is a compact, bounded subset of \( \mathbb{R}^n \). As a consequence, the basin of each attractor must share a boundary with that of every other attractor. This leads to spurious attractors and basins with holes, as have been described.

The state-space sculpting technique we will now present does not suffer these deficiencies. We can choose from all of \( \mathbb{R}^n \) the space or manifold in which to embed attractors, and we have control over attractor-basin geometry. Furthermore, the method is not limited to fixed-point attractors.

The first phase of state-space sculpting involves synthesizing a prescribed vector field. That process is detailed in the next section. The second phase involves implanting that vector field in a recurrent neural network, a process described in Chapter 4.

### 3.2 Vector-field synthesis

The state-space sculpting method for synthesis of vector fields, which has been published previously \[97\], may be summarized as follows. First the desired dynamics are defined. This typically amounts to a set of attractors (fixed point, limit cycle, strange) and their
locations. Then a corresponding set of continuous vector fields is collected, each of these having one of the defined attractors at one of the defined locations in its state space. This step is usually straightforward because analytical forms for standard fixed points and limit cycles are well known. However, the inverse methods developed by Sverdlove [91] can be used to generate desired fields when their analytical forms are not clear. Next, the boundaries along which these fields will be joined together are defined using smooth functions; these may also be used to delineate attractor basins. Finally, the set of vector fields and boundary functions is unified into a single continuous dynamical system using regularization. We expand on this summary below.

Our goal is to construct state spaces similar to those of the Hopfield net, but without its limitations to fixed point attractors and linear basin boundaries. Regularization, as developed by Sotomayor and Texeira [85], is the tool we use.

In regularization, a discontinuous vector field $F_Z$ is first defined from two continuous fields $F_X$ and $F_Y$ as

$$F_Z(q) = \begin{cases} 
F_X(q), & f(q) > 0, \\
F_Y(q), & f(q) < 0.
\end{cases}$$

where $F_{X,Y,Z} : M \to M$. For now we define $F_Z$ on a 2-manifold $M$ wherein the vector $q$ represents generalized state coordinates. The function $f : M \to \mathbb{R}$ is $C^\infty$ and the boundary $S = f^{-1}(0)$ divides $M$ into two disjoint subsets. Thus $F_X$ acts in the subset defined by $M^+ = f^{-1}(0, \infty)$ and $F_Y$ acts in the subset defined by $M^- = f^{-1}(-\infty, 0)$.

To combine fields $F_X$ and $F_Y$ as a single continuous system, a transition function $\varphi(s)$ is applied to smooth the $F_X$-$F_Y$ interface. This is a $C^\infty$ function $\varphi : \mathbb{R} \to \mathbb{R}$ such that $\varphi(s) = 0$ if $s \leq -1$, $\varphi(s) = 1$ if $s \geq 1$ and $d\varphi(s)/ds > 0$ if $s \in (-1, 1)$. To obtain these properties we define the following:

$$g(s) = \begin{cases} 
eg^\frac{1}{2} & \text{for } s > 0 \\
0 & \text{for } s \leq 0.
\end{cases}$$
Then we let \( h(s) = g(s + 1)g(1 - s) \), and finally define

\[
\varphi(s) = \frac{\int_{-\infty}^{s} h(\tau)d\tau}{\int_{-\infty}^{\infty} h(\tau)d\tau}.
\]

The transition function \( \varphi \) acts like a bump function or a partition of unity, endowing each field with a region of effective influence and smoothing the dynamics at the boundary.

The regularization \( F_\epsilon^Z \) of the fields \( F_X \) and \( F_Y \) is given by

\[
F_\epsilon^Z(q) = \varphi_\epsilon(f(q))F_X(q) + [1 - \varphi_\epsilon(f(q))] F_Y(q)
\]

where \( \varphi_\epsilon(s) = \varphi(s/\epsilon) \). It is proved in [85] that for \( \epsilon \) sufficiently small, if \( F_X \) and \( F_Y \) are \( C^r \) for some \( r \) then the regularization \( F_\epsilon^Z \) is \( C^r \) and structurally stable.

The resulting vector field \( F_\epsilon^Z \) defines a \( C^r \) continuous dynamical system with the combined dynamics of \( F_X \) and \( F_Y \) acting, respectively, in the regions delineated by \( f \). Now, if an attractor is global in the region where its vector field acts, then that region’s boundary determines the basin of attraction. As a consequence, it is possible to sculpt basins by manipulating the submanifold \( S = f^{-1}(0) \), where we have choice of the definition of \( f \).

We illustrate the vector field synthesis method with a simple example. We will build an associative memory system on \( \mathbb{R}^2 \) with two stored patterns given by fixed point attractors. The system is in many ways analogous to the Hopfield network just analyzed. Let us embed one fixed point at \((x_1, y_1) = (-p, 0)\) and a second at \((x_2, y_2) = (p, 0)\).

We define our vector fields accordingly, noting that the fixed point at \((-p, 0)\) should be manifest in \( F_Y \) and the fixed point at \((p, 0)\) in \( F_X \). Using straightforward linear functions for the vector fields, we have

\[
F_Y(q) = F_Y(x, y) = (a(x + p), by) \quad (3.4)
\]

\[
F_X(q) = F_X(x, y) = (a(x - p), by) \quad (3.5)
\]

where setting \( a < 0 \), \( b < 0 \) engenders the prescribed attractor dynamics. A natural choice for the attractor basin of the first fixed point is the half plane \( \{(x, y) \mid x < 0\} \) and for that of the second, \( \{(x, y) \mid x > 0\} \). This implies that the regularization boundary,
which separates $\mathbb{R}^2$ into disjoint sets and where the transition function acts, is given by $S = \{(x, y) \mid x = 0\}$. We write this as $S = f^{-1}(0)$, implying $f(x) = x$.

The regularization $F^\epsilon_Z : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is now constructed according to (3.3) with $\epsilon$ set to unity:

$$F^\epsilon_Z(x, y) = \varphi(\epsilon)(a(x - p), by) + (1 - \varphi(\epsilon))(a(x + p), by).$$

Figure 3.7: Vector field and several orbits for the regularized system of (3.6).

A plot of the regularized vector field is shown in Figure (3.6), along with the regularization boundary, and several orbits starting from random initial conditions. The field is depicted by black arrows, points in the orbits by green markers and the boundary by a red dashed line; orbits are obtained by solving the differential system $(\dot{x}, \dot{y}) = F^\epsilon_Z(x, y)$ by numerical integration with MATLAB.

The two-attractor system we have built has straightforward associative dynamics. Orbits converge to one of the two fixed points depending on their starting locations, with points to the left of $x = 0$ flowing to $(-p, 0)$ and points to the right flowing to $(p, 0)$. The line $x = 0$ acts as the intended separatrix forming the boundary between attractor basins. Note from the plotted vector field that this separatrix contains a saddle node, at $y = 0$, as was seen in the Hopfield network. The node is attracting for points along the line $x = 0$ but repelling otherwise.
More interesting sculpted systems will be presented in later chapters but this example conveys the gist of the synthesis procedure. Of course, we are not limited to fixed point attractors nor linear vector fields nor linear regularization boundaries. This last limitation would imply that only vector fields with linearly separable basins could be synthesized, recalling the shortcoming of the classic perceptron—but as stated, the only restriction on regularization boundaries is that they belong to the class of \( C^\infty \) functions.

Design of complicated dynamical systems may call for regularization of any number of vector fields. This is accomplished recursively. From the full set of vector fields to be combined, a pair of fields that share a boundary is selected and these two are regularized; the newly regularized field, treated as a whole, may then be merged with the other fields at subsequent steps. This process is detailed in the \textit{recursive pairwise} algorithm given below.

\textbf{Algorithm 1.} (Recursive pairwise method)

1. To each of the \( p \) attractors associate an appropriately designed vector field and basin. This yields a set of \( p \) vector fields as well as a set of \( p - 1 \) basin boundary functions \( f_j(q) : M \to \mathbb{R}, j = 1, ..., p - 1 \).

2. From the set of boundary functions, select a \( f_j(q) \).

3. If this boundary intersects another, go to Step 8. Otherwise continue.

4. From the set of vector fields, select the field where \( f_j(q) < 0 \) and label it \( Y_j(q) \); select the field where \( f_j(q) > 0 \) and label it \( X_j(q) \).

5. Generate the regularized field \( Z_j(q) = \varphi(\epsilon f_j(q))X_j(q) + [1 - \varphi(\epsilon f_j(q))]Y_j(q) \). This field inherits a boundary based on the placement of the remaining nonregularized vector fields.

6. Delete vector fields \( Y_j(q), X_j(q) \) from the set of vector fields and boundary function \( f_j(q) \) from the set of boundary functions. Add \( Z_j(q) \) to the set of vector fields, forgetting its label.

7. If the set of boundary functions is empty, stop. Otherwise, go to Step 2.
8. Label the intersecting boundary function \( f_j^{(t)}(q) \).

9. \( f_j(q) \) separates the phase space into \( M^+ \) and \( M^- \). In \( M^+ \) (viewed in isolation), regularize the vector fields that contact \( f_j(q) \) across boundary \( f_j^{(t)}(q) \) (according to Steps 4-7 with \( f_j^{(t)}(q) \)).

10. Do the same in \( M^- \).

11. Now \( M^+ \) and \( M^- \) are both characterized locally by only one (regularized) vector field. Regularize these two fields across boundary \( f_j(q) \) (according to Steps 4-7).


The recursive pairwise algorithm appears somewhat cumbersome when attempting to merge a large number of vector fields. However, it makes adding a single field to an already existing, possibly complicated system quite straightforward. The existing system is treated as a whole rather than as a collection of fields, and it is the whole that is regularized with the additional field. From the point of view of the algorithm, there are just two systems (one new, one old) and one border between them. The ease of adding an attractor to an existing collection in this way makes the technique well suited to a system that learns new memories through experience.

From the forms of \( F_Z^\epsilon(q) \) and \( \varphi_\epsilon(s) \) we can deduce that in each basin, as defined by the appropriate \( f_j(q) \), the dynamics are generally dictated by the vector field for that basin; this is true everywhere except for in a strip of width \( \epsilon \) along the basin boundary. Within this strip, the vector field blends progressively with the field that dominates on the other side of the boundary according to a weighted average; the weights are even along the boundary itself. If two or more boundaries are within a distance of less than \( 2\epsilon \) from each other, then a blending of at least three vector fields occurs. In both cases new attractors or other dynamical effects may be born (for a discussion of these effects see, e.g., [9]).

The situation is further complicated in the case that boundaries intersect; additional steps must be taken during the regularization process (see algorithm), and again the blending of multiple vector fields can produce interesting dynamical effects. To provide a
second relatively simple example: in Figure 3.8 four vector fields containing saddle nodes (marked by blue *’s) have been regularized across two boundaries using the recursive algorithm. Each of these fields is described by $F_S(x, y) = (a(x - p_x), b(y - p_y))$, where $p_x$ and $p_y$ are varied for location and alternately one of $a$, $b$ is positive while the other is negative. The standard $x$- and $y$-axes are used as boundaries. This synthesis gives rise to a heteroclinic channel (in fact a cycle) as the unstable manifold of one saddle leads into the stable manifold of another. This is a key feature of the Winnerless Competition Principle of neural computation [4], which our state-space sculpting framework is able to capture. Note also that a stable centre has been born at the saddles’ centroid where the boundaries intersect, and the stable/unstable manifolds of the saddles produce a walled-off cell in the state space that no trajectory can enter or exit. These are two examples of the unintended dynamical effects mentioned.

To clarify the nature of regularized systems we now provide mathematical expressions for their Jacobians, divergence, and curl, etc. These properties are important in classifying dynamical systems.
### 3.3 Various Expressions

The fixed points of the regularized system, \( \{ \mathbf{q} \mid Z_\epsilon(\mathbf{q}) = 0 \} \), are given by the solution to

\[
F_X(\mathbf{q}) = - \left( \frac{1 - \varphi_\epsilon(f(\mathbf{q})))}{\varphi_\epsilon(f(\mathbf{q})))} \right) F_Y(\mathbf{q})
\]

in the smoothing region (i.e., the region within \( \epsilon \) of the boundary). Outside the smoothing region, where \( \varphi_\epsilon(f(\mathbf{q})) = 0 \) we have the fixed points of \( F_Y \) and where \( \varphi_\epsilon(f(\mathbf{q})) = 1 \) we have the fixed points of \( F_X \) as expected. On the boundary \( f(\mathbf{q}) = 0, \varphi_\epsilon(f(\mathbf{q})) = 0.5 \) for a symmetric transition function as used above and the fixed points are given by the solution to \( X(\mathbf{q}) = -Y(\mathbf{q}) \), namely those locations where the vector fields are exactly opposite.

The Jacobian of a regularized field is more interesting. We write that the Jacobian of field \( F_X \) is \( \mathbf{J}_X = [x_{ij}] \) where \( x_{ij} = \partial X_i / \partial q_j \) and similarly for the Jacobians of fields \( F_Y \) and \( F_Z \). A straightforward computation for \( \mathbf{J}_Z \) shows that

\[
z_{ij} = \varphi \frac{\partial X_i}{\partial q_j} + (1 - \varphi) \frac{\partial Y_i}{\partial q_j} + \frac{\partial \varphi}{\partial f} \frac{\partial f}{\partial q_j} (X_i - Y_i)
\]

\[
= \varphi x_{ij} + (1 - \varphi)y_{ij} + d_{ij},
\]

where we have dropped \( \epsilon \) and \( f \)'s explicit dependence on \( \mathbf{q} \) to simplify the notation. We call the

\[
d_{ij} = \frac{\partial \varphi}{\partial f} \frac{\partial f}{\partial q_j} (X_i - Y_i)
\]

'partial difference' terms. We collect these in a matrix \( \mathbf{D} = [d_{ij}] \) for future use. Each term in the Jacobian \( \mathbf{J}_Z = [z_{ij}] \) is thus a weighted average of the corresponding terms in \( \mathbf{J}_X \) and \( \mathbf{J}_Y \) plus a partial difference term.

The fixed points of a 2-dimensional system are completely characterized by the *trace* and *determinant* of the Jacobian, so we calculate these for the regularized system begin-
With the trace, $\tau_{J_z}$:

$$
\tau_{J_z} = \varphi x_{11} + (1 - \varphi)y_{11} + d_{11} + \varphi x_{22} + (1 - \varphi)y_{22} + d_{22}
$$

$$
= \varphi(x_{11} + x_{22}) + (1 - \varphi)(y_{11} + y_{22}) + (d_{11} + d_{22})
$$

$$
= \varphi\tau_{J_x} + (1 - \varphi)\tau_{J_y} + \tau_D
$$

This expression is fairly straightforward. We see that the trace is a weighted average of the traces of $J_X$ and $J_Y$ plus the trace of the partial difference matrix $D$.

The following quantities will be helpful in expressing the determinant, $\Delta_{J_z}$. We define

$$
\Delta_{xy} = \begin{vmatrix} x_{11} & y_{12} \\ x_{21} & y_{22} \end{vmatrix}
$$

with $\Delta_{yx}$, $\Delta_{dx}$, $\Delta_{xd}$, $\Delta_{dy}$, and $\Delta_{yd}$ defined analogously, i.e., the determinant of a matrix built by concatenating the appropriate columns from $J_X$, $J_Y$, or $D$ as indicated by the subscripts. Now,

$$
\Delta_{J_z} = [(1 - \varphi)y_{11} + \varphi x_{11}][(1 - \varphi)y_{22} + \varphi x_{22}] - [(1 - \varphi)y_{21} + \varphi x_{21}][(1 - \varphi)y_{12} + \varphi x_{12}]
$$

$$
+ d_{11}[(1 - \varphi)y_{22} + \varphi x_{22}] + d_{22}[(1 - \varphi)y_{11} + \varphi x_{11}] - d_{21}[(1 - \varphi)y_{12} + \varphi x_{12}]
$$

$$
- d_{12}[(1 - \varphi)y_{21} + \varphi x_{21}] + d_{11}d_{22} - d_{21}d_{12}
$$

$$
= (1 - \varphi)^2\Delta_{J_y} + \varphi^2\Delta_{J_x} + \varphi(1 - \varphi)(\Delta_{yx} + \Delta_{xy})
$$

$$
+ (1 - \varphi)(\Delta_{dy} + \Delta_{yd}) + \varphi(\Delta_{dx} + \Delta_{xd}) + \Delta_D.
$$

This expression is obviously quite complicated, unfortunately, and does not allow us to make many clear claims about the character of the determinant. Outside of the smoothing region, $\varphi(1 - \varphi)$, $\partial\varphi/\partial f$ and one of $\varphi^2$, $(1 - \varphi)^2$ go to zero, leaving simply the determinant of either $J_X$ or $J_Y$ as expected. At the boundary, however, where $\varphi \approx 0.5$ and thus $\varphi \approx (1 - \varphi)$, the first three terms in this expression are all equally dominant; also $\partial\varphi/\partial f$ is near its maximum, meaning the final three terms could be relatively large as well.
To gain a better understanding of what goes on in the smoothing region without dealing with $\Delta_{J_z}$, we can determine the divergence and curl of $F_Z$ as follows:

$$\nabla \cdot F_Z = \varphi \nabla \cdot F_X + (1 - \varphi) \nabla \cdot F_Y + \nabla \varphi \cdot (F_X - F_Y)$$

$$\nabla \times F_Z = \nabla \times (\varphi F_X + (1 - \varphi) F_Y)$$

$$= \varphi \nabla \times F_X + (1 - \varphi) \nabla \times F_Y + \nabla \varphi \times F_X + \nabla (1 - \varphi) \times F_Y$$

$$= \varphi \nabla \times F_X + (1 - \varphi) \nabla \times F_Y + \nabla \varphi \times (F_X - F_Y)$$

These expressions are much cleaner than that for $\Delta_{J_z}$. They demonstrate that the divergence (curl) is a weighted average of the divergence (curl) of the vector fields $F_X$ and $F_Y$, plus a term with the same structure as the divergence (curl) that involves the gradient of $\varphi$ and the ever-present difference field $F_X - F_Y$.

### 3.4 Nota bene

Before we move on to the next chapter, two points of clarification are in order. First, why regularization? For a system as simple as our first example, we could define two vector fields that met discontinuously at the boundary and the dynamics would be essentially the same; this is because no orbit can ever cross the separatrix. However, other systems might require that an orbit traverse a regularization boundary, in which case any dynamical effects caused by field discontinuity would be undesirable. But more important, the theoretical underpinnings of the next phase of the sculpting procedure require a vector field that is at least $C^1$, and indeed, we have found that our RNN training process (to be detailed in the following chapter) achieves better convergence on smoother data. The second point to note is that the exposition of regularization theory we have given, based on [85], is restricted to 2-dimensional systems. This was mainly for the sake of simplicity; the theory was extended to 3-dimensional systems in [59]. Regularization has not been proved to preserve attractors in higher dimensions if they exist near a regularization boundary, but we will design our dynamical systems such that this issue does not arise.
Chapter 4

Synthesis of recurrent neural networks

4.1 Overview

Funahashi and Nakamura proved that a continuous-time dynamical system can be approximated to arbitrary accuracy by a recurrent neural network [27]. Although the proof of their result was not constructive, we have developed an algorithm that harnesses the standard machine learning technique of backpropagation to generate good approximations of the ideal theoretical network. The result of [27] does not account for dynamical systems with external inputs (e.g., forcing functions) and we have made that extension. We also extend their proof of convergence. Following the mathematical explication, we demonstrate the capabilities of our synthesis algorithm with several examples of dynamical systems realized as RNNs. We then discuss a relation of our synthesized networks to echo state networks. Finally, we analyze the behaviour of the networks’ hidden neurons, developing a useful algebraic function that approximates their states from those of the output neurons, and demonstrate how network potential energy functions may be computed.
Chapter 4. Synthesis of recurrent neural networks

4.2 Theoretical foundations

Theorem 1 of Funahashi and Nakamura [27] states that any finite-time trajectory of a given $n$-dimensional dynamical system can be approximately realized to arbitrary accuracy by an $(n + m)$-unit continuous-time recurrent neural network. This RNN has $n$ output units, whose internal states replicate the $n$-dimensional orbits of the prescribed dynamical system, and $m$ hidden units. The formal distinction between output units and hidden units comes from the network topology as shall be seen shortly.

The particular class of networks discussed here, often referred to as the class of additive neural networks, is described by the following differential equation:

$$\dot{s}(t) = -Ps(t) + W\sigma(s(t)) + u(t),$$

where $s(t) \in \mathbb{R}^{n+m}$ is the vector of internal neuron states, $P \in \mathbb{R}^{n+m \times n+m}$ is the diagonal matrix of neuron time constants, $W \in \mathbb{R}^{n+m \times n+m}$ is the matrix of connection weights, and $u(t) \in \mathbb{R}^{n+m}$ is the vector of neuron inputs.\footnote{We indicate the space of $p \times q$ real matrices as $\mathbb{R}^{p \times q}$ and accordingly $\mathbb{R}^p$ is the space of $p \times 1$ real column matrices.} We divide the neurons into output units, whose activations are given by $\omega(t) \in \mathbb{R}^n$, and hidden units, whose activations are $\eta(t) \in \mathbb{R}^m$, such that

$$s = \begin{pmatrix} \omega \\ \eta \end{pmatrix}. \quad (4.2)$$

The activation function, $\sigma(x) = \col[\sigma(x_i)]$, where the bounded, monotonic, increasing $C^1$-sigmoid,

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad (4.3)$$

is generally used.

All neurons in the recurrent networks used here will share the same time constant, $\tau$, so that $P = \text{diag}[\tau^{-1}]$. In this section, following [27], we focus on RNNs without input ($u(t) = 0$). We will discuss input signals two sections hence.
The dynamical system we wish to approximate with an RNN is given by

\[ \dot{q} = F(q), \quad q(0) \in K, \]  

(4.4)

where \( F : D \to \mathbb{R}^n \) is a \( C^1 \)-mapping, \( D \) is a compact subset of \( \mathbb{R}^n \), and \( K \) is a compact subset of \( D \). We emphasize that \( q(t) \) represents the solution initiated at \( q(0) \).

Theorem 1 of Funahashi & Nakamura [27] guarantees that a recurrent neural network of the form (4.1), with identical time constants and without input, can be constructed such that

\[ \max_{t \in J} \| q(t) - \omega(t) \| < \delta \]  

(4.5)

where \( \delta \) is arbitrarily specified and \( J = [0, T] \) is the interval of time on which the solution \( q(t) \) exists. That is to say, orbits generated from the RNN will trace those of the original dynamical system arbitrarily closely.

The key to this theorem’s proof, and indeed our method, is existence of parameters \( m, A \in \mathbb{R}^m, B \in \mathbb{R}^n \), and \( \theta \in \mathbb{R}^m \) such that

\[ \max_q \| F(q) - A\sigma(Bq + \theta) \| < \frac{L_F}{4(e^{LF} - 1)} \delta, \]  

(4.6)

where \( L_F \) is the Lipschitz constant of \( F \). The goal is to approximate \( F(q) \) by

\[ \tilde{F}(q) \triangleq -\frac{1}{\tau} q + A\sigma(Bq + \theta) \]  

(4.7)

such that the solution to \( \dot{q} = \tilde{F}(q) \) is a very close approximation to the solution of \( \dot{q} = F(q) \). [27] show that

\[ \max_{t \in J} \| q(t) - \tilde{\omega}(t) \| < \frac{\delta}{2}. \]  

(4.8)

where \( \tilde{\omega}(t) \), to harmonize with the upcoming notation, is the solution to the system described by \( \tilde{F} \). Our task, therefore, is to find \( m, A, B, \theta \) and \( \tau \).

Parameters such that (4.6) is satisfied are known to exist by the earlier “fundamental approximation theorem” of [26]. If these parameters can be determined, then it is possible to rewrite the latter vector field in the recurrent form of (4.1).
Before presenting the recurrent neural network to approximate the dynamical system \( \dot{\mathbf{q}} = F(\mathbf{q}) \) we shall need to introduce an intermediate system. Defining \( \tilde{\mathbf{\eta}} \triangleq \mathbf{B}\tilde{\mathbf{\omega}} + \theta \), we have
\[
\dot{\tilde{\mathbf{\eta}}} = \mathbf{B}\dot{\tilde{\mathbf{\omega}}} = -\frac{1}{\tau} \tilde{\mathbf{\eta}} + \mathbf{C}\sigma(\tilde{\mathbf{\eta}}) + \frac{1}{\tau} \theta,
\]
where \( \mathbf{C} = \mathbf{BA} \in m\mathbb{R}^n \). Now set \( \tilde{\mathbf{s}} = \text{col} [\tilde{\mathbf{\omega}}, \tilde{\mathbf{\eta}}] \) and define the mapping \( \tilde{G} : n^+m\mathbb{R} \to n^+m\mathbb{R} \)
by
\[
\tilde{G}(\tilde{\mathbf{s}}) \triangleq -\frac{1}{\tau} \tilde{\mathbf{s}} + \mathbf{W}\sigma(\tilde{\mathbf{s}}) + \frac{1}{\tau} \tilde{\theta}
\]
where
\[
\mathbf{W} = \begin{pmatrix} \mathbf{O} & \mathbf{A} \\ \mathbf{O} & \mathbf{C} \end{pmatrix} \in n^+m\mathbb{R}^{n+m}, \quad \tilde{\theta} = \begin{pmatrix} \mathbf{0} \\ \theta \end{pmatrix} \in n^+m\mathbb{R}.
\]
(4.11)
Then the first \( n \) components of the solution of
\[
\dot{\tilde{\mathbf{s}}} = \tilde{G}(\tilde{\mathbf{s}}), \quad \tilde{\mathbf{s}}(0) = \begin{pmatrix} \tilde{\mathbf{q}}(0) \\ \mathbf{B}\tilde{\mathbf{q}}(0) + \theta \end{pmatrix}
\]
form the solution of \( \dot{\mathbf{q}} = \tilde{F}(\mathbf{q}) \).

Finally, we define the mapping \( G : n^+m\mathbb{R} \to n^+m\mathbb{R} \) as
\[
G(\mathbf{s}) \triangleq -\frac{1}{\tau} \mathbf{s} + \mathbf{W}\sigma(\mathbf{s})
\]
(4.12)
where, as before, \( \mathbf{s} = \text{col} [\mathbf{\omega}, \mathbf{\eta}] \). The dynamical system defined by \( G \) can be realized by a recurrent neural network with \( \mathbf{\omega} \) as the internal states of \( n \) output units and \( \mathbf{\eta} \) as the internal states of \( m \) hidden units. Note that, in (4.11) for \( \mathbf{W} \), the block \( \mathbf{A} \) represents connections from the hidden units to the output units; the block \( \mathbf{C} \) represents connections between hidden units, and it is here that recurrent loops lie.

As mentioned, the parameter \( \tau \) must be selected to satisfy certain conditions which we now give. These are [27]
\[
\frac{\|\mathbf{q}\|}{\tau} < \frac{\delta L_F}{4(e^{LF\tau} - 1)}, \quad \frac{1}{\tau} < \frac{L_{\tilde{G}}}{2}, \quad \frac{\|\theta\|}{\tau} < \frac{\delta L_{\tilde{G}}}{4(e^{L_{\tilde{G}}\tau} - 1)}, \quad (4.13)
\]
where $L_{\tilde{G}}$ is the Lipschitz constant of $\tilde{G}$. Clearly, $\tau$ satisfies all conditions (4.13) when it is sufficiently large.

It is proved in [27] that if $\tau$ satisfies (4.13), then for $\omega(0) = \tilde{\omega}(0) = q(0)$ and $\eta(0) = \tilde{\eta}(0) = Bq(0) + \theta$,

$$\max_{t \in J} \| \tilde{s}(t) - s(t) \| \leq \frac{\delta}{2}$$

and hence

$$\max_{t \in J} \| \tilde{\omega}(t) - \omega(t) \| \leq \frac{\delta}{2}. \quad (4.14)$$

Combining (4.8) and (4.14) yields the desired result of (4.5), telling us that the RNN system $\dot{s} = G(s)$ approximates $\dot{q} = F(q)$ arbitrarily closely.

The maximum discrepancy between the vector fields $A\sigma(Bq + \theta)$—call this $F_{NN}(q)$—and $F(q)$, i.e.,

$$E_{\max} = \max_q \| F(q) - A\sigma(Bq + \theta) \| \quad (4.15)$$

establishes an upper bound on the accuracy of the RNN’s approximation. Assuming $\tau$ is sufficiently large,

$$\max_{t \in J} \| q(t) - \omega(t) \| \leq \frac{2E_{\max}}{L_F} (e^{L_F T} - 1) = E_L. \quad (4.16)$$

As mentioned, we know from the fundamental approximation theorem [26] that the parameters to satisfy any desired accuracy exist—it is now a matter of finding them.

### 4.3 Network training and solution

Parameters that make $E_{\max}$ small can be found algorithmically. The key insight is that the approximating vector field $F_{NN}(q)$ takes the form of a three-layer feedforward neural network. This network has $n$ input units, $m$ sigmoidal hidden units, and $n$ linear output units. As such, it is amenable to standard machine learning techniques. Facilitating this training approach is the mathematical expression for the vector field $F(q)$ defining the desired system. This expression can be sampled to arbitrary resolution to provide training data. Specifically, a set of training points is sampled from a region of interest
Figure 4.1: Schematic of the three stages of the RNN synthesis algorithm. The general vector field on the left may be a product of regularization.

\[ D \] in the dynamical system’s state space. These points act as the feedforward network input; evaluating \( F(q) \) on the set of input points yields the target set of feedforward outputs.

We have implemented a machine-learning approach with excellent results. Using Levenberg-Marquardt backpropagation (the standard training algorithm in MATLAB’s neural network toolbox), we train a three-layer feedforward neural network on data sampled from vector field \( F(q) \). Training yields \( A, B, \) and \( \theta \) such that \( E_{\text{max}} \) is small. These feedforward networks are then transformed into continuous-time recurrent networks as described, that is, the RNN is given by \( G \) as defined in (4.12). Solution of the RNN as a dynamical system \( \dot{s} = G(s) \) is based on the initial condition

\[
s(0) = \begin{pmatrix} \omega(0) \\ \eta(0) \end{pmatrix}, \quad \omega(0) = q(0), \quad \eta(0) = Bq(0) + \theta.
\] (4.17)

The condition on the hidden units is an affine transformation of the condition on the output units, which is identical to the condition on the original dynamical system. The recurrent network so defined closely reproduces the original system’s dynamics, as results presented in the next section demonstrate. In practice, we find we can do much better than the bound in (4.16) which increases exponentially with time.
4.4 Systems with external forcing

We have thus far concentrated on the simulation of autonomous dynamical systems. However, another important class of problem to consider is the system with external forcing. The present method can be extended to enable the construction of RNNs to simulate driven dynamical systems.

Driven systems are generally described by nonautonomous ODEs, namely,
\[
\dot{q} = F(q) + f(t).
\]  

(4.18)

where \(f(t)\) accounts for any external force. It is certainly possible to drive a network constructed with our algorithm by some external signal \(f(t)\). This is just a matter of connecting the driving signal additively to the system’s output units and integrating as usual. However, we want to show that it is possible for our framework to capture driven systems in their entirety; that is, that a properly trained and constructed recurrent network can generate the solution of the complete system (4.18).

This type of system does not fall under the realm of Funahashi and Nakamura’s theoretical work. However, there are several ways in which external forces can be treated.

First and most directly, a nonautonomous system can be recast into autonomous form by the creation of a new variable representing time, i.e., \(q_{n+1} = t\). The system of ODEs is then augmented by the simple differential relation, \(\dot{q}_{n+1} = 1\). Unfortunately, systems in this form are not amenable to the procedure outlined above. This is because Funahashi & Nakamura’s theorem requires that the state variables lie on some compact subset of \(\mathbb{R}^n\). Time \(t\), which is now a linearly increasing state variable, will always leave any such compact subset.

An alternative approach, which circumvents this theoretical issue, is to recognize that forcing functions can themselves be considered as products of their own dynamics. For example, a sinusoidal forcing term, \(f(t) = a \cos(\omega t + \phi)\), is the result of the system \(\ddot{f} + \omega^2 f = 0\), where the amplitude and phase are determined by the initial conditions. In this case, the original system can be augmented by the forcing dynamics. However, the approach taken here is to build a recurrent neural network from the forcing dynamics in
the same manner as described in the foregoing for autonomous systems.

We begin by assuming the dynamics of the driving force to be described by

\[ \dot{q}_f = F_f(q_f) \]

where \( q_f(t) \in \mathbb{R}^p \) and \( p \geq n \), the dimensional of the original dynamical system. The first \( n \) components of \( q_f \) are taken to be \( f(t) \), i.e., \( q_f = \text{col}[f, g] \). (In the case, for example, of a forcing function with oscillatory dynamics, this generalized form is necessary.) Using feedforward training, we teach an RNN the forcing dynamics, representing it as

\[ \dot{s}_f = G_f(s_f) \]

where, in like fashion to \( G(s) \),

\[ G_f(s_f) = -\frac{1}{\tau} s_f + W_f \sigma(s_f), \]

The state vector \( s_f \in \mathbb{R}^{p+q} \) includes the \( p \) output neurons \( (\omega_f) \), which include the forcing functions, and \( q \) hidden neurons \( (\eta_f) \). The weight and bias matrices, in familiar fashion, are described as

\[ W_f = \begin{pmatrix} O & A_f \\ O & C_f \end{pmatrix}, \quad \tilde{\theta}_f = \begin{pmatrix} 0 \\ \theta_f \end{pmatrix} \]

where \( C_f \triangleq B_f A_f, A_f \in \mathbb{R}^{q,p}, B_f \in \mathbb{R}^{q} \) and \( \theta_f \in \mathbb{R}^q \). We further break down \( A_f \) as

\[ A_f \triangleq \begin{pmatrix} A_{ff} & A_{fg} \end{pmatrix} \]

where \( A_{ff} \in \mathbb{R}^{n} \) is associated with the force vector \( f \).

As before, a recurrent network, \( \dot{s} = G(s) \), is trained on the vector field of the prescribed dynamical system, \( F \). This training is performed on the system in the absence of the driving force. The task is now to merge the two networks, \( G \) and \( G_f \).

We note that the input of the \( n \) output neurons \( \omega \) of \( G \) is the approximated vector field \( \tilde{F}(\omega) \). We therefore wish to add as inputs to these neurons, though bypassing the
sigmoid operator, the outputs $\omega_f(t)$ from $G_f$. Mathematically, both $G$ and $G_f$ may be parsed as

$$
\dot{\omega} = -\frac{1}{\tau} \omega + A\sigma(\eta), \quad \dot{\omega}_f = -\frac{1}{\tau} \omega_f + A_f\sigma(\eta_f) 
$$

$$
\dot{\eta} = -\frac{1}{\tau} \eta + C\sigma(\eta) + \frac{1}{\tau} \theta, \quad \dot{\eta}_f = -\frac{1}{\tau} \eta_f + C_f\sigma(\eta_f) + \frac{1}{\tau} \theta_f 
$$

(4.23)

We need to augment the differential equation for $\omega$ to

$$
\dot{\omega} = -\frac{1}{\tau} \omega + A\sigma(\eta) + \omega_{ff} 
$$

(4.24)

The new merged system $G_\Sigma$ may then be written as

$$
\dot{s}_\Sigma = -\frac{1}{\tau} s_\Sigma + W_\Sigma \sigma(s_\Sigma) + K_\Sigma s_\Sigma 
$$

(4.25)

with

$$
s_\Sigma \triangleq \begin{pmatrix} \omega \\ \omega_f \\ \eta \\ \eta_f \end{pmatrix}, \quad W_\Sigma \triangleq \begin{pmatrix} \cdot & A & \cdot \\ \cdot & A_f & \cdot \\ \cdot & C & \cdot \\ \cdot & C_f & \cdot \end{pmatrix}, \quad K_\Sigma \triangleq \begin{pmatrix} \cdot & K_f & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix},
$$

where $K_f \triangleq \begin{bmatrix} 1 & 0 \end{bmatrix}$ and $1$ is the $n \times n$ identity matrix which has the effect of picking out only the variables corresponding to $f$, i.e., $\omega_{ff}$, in $\omega_f$; the dots represent zero matrices.

To complete the dynamical description, we take

$$
s_\Sigma(0) \triangleq \begin{pmatrix} q(0) \\ q_f(0) \\ Bq(0) + \theta \\ B_f q_f(0) + \theta_f \end{pmatrix}
$$

as the initial conditions. We may regard the new system $G_\Sigma$ as a forced recurrent neural network (FRNN).
The solution $\omega$ computed from the output neurons of the recurrent neural network $G_f$ can be guaranteed to be arbitrarily close to the solution to the true dynamical system $\dot{q} = F(q) + f$; that is, (4.5) still obtains. To show this, we follow the path of Funahashi & Nakamura [27] and for convenience we consider the general dynamical system,

$$\dot{x} = R(x) + Ex$$

(4.26)

In the present case, this system accounts for the dynamics of the forcing function as well which enters via the $Ex$ term. We define $\tilde{R}(x)$ is analogous fashion to (4.7). Provided that conditions similar to (4.13) are met,

$$\max_x \| R(x) - \tilde{R}(x) \| \leq \frac{\delta L_R}{4(e^{L_R T} - 1)}$$

(4.27)

and it follows that

$$\max_x \| R(x) + Ex - (\tilde{R}(x) + Ex) \| \leq \frac{\delta L_R}{4(e^{L_R T} - 1)}$$

(4.28)

which in turn implies

$$\max_{t \in J} \| x(t) - \tilde{x}(t) \| \leq \frac{\delta}{2}$$

(4.29)

where $\tilde{x}$ is the solution to $\dot{\tilde{x}} = \tilde{R}(x) + Ex$. Note that $\delta$ can be arbitrarily specified.

As in the unforced case, we build $S$ and $\tilde{S}$ on the blueprints of $G$ and $\tilde{G}$ and identify the overall state as $z$ in the place of $s$. Now

$$\| S(z) - \tilde{S}(z) \| < \frac{\delta L_{\tilde{S}}}{4(e^{L_{\tilde{S}} T} - 1)}$$

(4.30)

whence we also have that

$$\| S(z) + Hz - (\tilde{S}(z) + Hz) \| < \frac{\delta L_{\tilde{S}}}{4(e^{L_{\tilde{S}} T} - 1)}$$

(4.31)
where \( \mathbf{H} \triangleq \text{col} [\mathbf{E}, \mathbf{O}] \). (Note that \( \mathbf{Hz} \) takes on the role of \( \mathbf{u} \) in (4.1).) Thus

\[
\max_{t \in J} \| \tilde{\mathbf{z}}(t) - \mathbf{z}(t) \| \leq \frac{\delta}{2}
\]

(4.32)

where \( \mathbf{z} \) is the solution to \( \dot{\mathbf{z}} = \mathbf{S}(\mathbf{z}) + \mathbf{Hz} \) and \( \tilde{\mathbf{z}} \) the solution to

\[
\dot{\tilde{\mathbf{z}}} = \tilde{\mathbf{S}}(\tilde{\mathbf{z}}) + \mathbf{H}\tilde{\mathbf{z}}
\]

(4.33)

which can be realized as a forced recurrent neural network. Inequality (4.32) holds for any subset of \( \mathbf{z} \) and its companion subset of \( \tilde{\mathbf{z}} \); in particular,

\[
\max_{t \in J} \| \tilde{\mathbf{x}}(t) - \varpi(t) \| \leq \frac{\delta}{2}
\]

(4.34)

where \( \varpi(t) \) is the state of output neurons of the FRNN corresponding to \( \tilde{\mathbf{x}} \), which we recall is the solution to \( \dot{\tilde{\mathbf{x}}} = \tilde{\mathbf{R}}(\tilde{\mathbf{x}}) + \mathbf{E}\mathbf{x} \).

Combining (4.29) and (4.34), as done previously, we conclude that

\[
\max_{t \in J} \| \mathbf{x}(t) - \varpi(t) \| \leq \delta
\]

(4.35)

For the problem at hand, we recognize that \( \mathbf{q}(t) \), the solution to the true forced dynamical system, would be a subset of states in \( \mathbf{x}(t) \) and thus

\[
\max_{t \in J} \| \mathbf{q}(t) - \omega(t) \| \leq \delta
\]

(4.36)

where \( \omega(t) \) represents the states of the corresponding subset of the output FRNN neurons. Examples of forced systems will be seen two sections hence.

### 4.5 Relation to the Neural Engineering Framework

As discussed in Chapter 2, there are strong connections between the NEF and our RNN training technique, although the two methods spring from different formalisms. We now make those connections explicit.
Both the NEF and our framework use a vector-field perspective in their modeling of neuron activity. Indeed, the linear encoders $\tilde{\phi}_i$ of the NEF correspond to the first-layer weights $B$ in our feedforward networks, while the linear decoders $\phi_F^{(q)}$ correspond to the second-layer weights $A$. We can make this clear by rewriting the NEF equations in matrix-vector form. For the neuron activities we have

$$a = G(\tilde{\Phi}q),$$

where $\tilde{\Phi}$ is the matrix whose rows are given by the scaled encoding vectors $\alpha_i\tilde{\phi}_i$, $G$ is the vector of activation functions, and we ignore the input current $J^{bias}$ as we do in our networks. For the decoding stage we have

$$\hat{F}(q) = \Phi^{F(q)}a,$$

where now $\Phi^{F(q)}$ is the matrix whose columns are given by the decoding vectors $\phi^{F(q)}_i$. This makes it obvious that $B$ plays the role of $\tilde{\Phi}$ while $A$ plays the role of $\Phi^{F(q)}$.

In using the class of additive recurrent neural networks, we have restricted our approach to neurons with smooth activation functions (the sigmoid), and which act as perfect integrators. That is, the neurons are described by a $1/s$ transfer function. As mentioned previously, the NEF admits generic activation and transfer functions.

One place where our method clearly differs from the NEF is the training procedure. In the NEF, the linear encoders $\tilde{\phi}_i$ are set randomly, while the set of (mean squared) optimal linear decoders $\phi^{F(q)}_i$ is computed by minimizing the mean squared error at the output layer only. On the other hand, our training methodology uses backpropagation to train both sets of weights. Because errors are propagated through the entire model, it is possible that systems may be modeled more accurately using fewer neurons with our approach, although a direct comparison has not been undertaken. As mentioned, an advantage of the NEF approach is that neuron activation functions need not be differentiable. On the other hand, a framework like ours in which errors can be propagated throughout can be included as a module within a larger system that is trained end-to-end, as is the current trend in machine learning [30, 87].
In a later section, we will use the restricted, simpler form of our networks to develop an algebraic approximation for their hidden units. This facilitates network analysis such as the computation of network potential functions, and makes explicit that the networks simulate lower-dimensional systems. Furthermore, to our knowledge it was not known before this work that the theoretical bounds of Funahashi and Nakamura [27] could be applied to this restricted class of NEF systems.

4.6 Dynamical systems realized as RNNs

Now we construct and present a series of recurrent neural systems. Our focus is on systems with attractors in their state space because of their importance to neuroscience, and specifically memory. The networks are constructed as detailed above. Levenberg-Marquardt backpropagation is used for the training phase, with training data given by samples from the field $F(q)$ on some compact set $D$. Thus, for a given training case, $q$ is the network input and $F(q)$ the desired feedforward output. Although the number of output units $n$ is defined by the dimensionality of $F$, the number of hidden units $m$ is unconstrained and a matter of choice. More complicated dynamical systems naturally require a larger $m$ and more training data; in practice, we aim for the smallest $m$ such that training converges. The time constant $\tau$ is set in all networks to $10^6$ (variables are nondimensionalized).

Most of the systems depicted will lie in a two-dimensional state space, although the algorithm works equally well in higher dimensions. In order to demonstrate the algorithm’s capabilities in replicating general, rather arbitrary systems, we will train on vector fields containing multiple attractors. These fields are synthesized by the regularization technique presented in Chapter 3. Recall that the regularization process guarantees that $F(q)$ is $C^r$ continuous if the component vector fields are $C^r$; this is significant because the RNN synthesis process is guaranteed to work only on a dynamical system that is at least $C^1$.

The figures that follow will in general depict: (a) a vector field (black arrows) within some compact region $D$; (b) orbits $q(t)$ obtained by integrating the system’s mathemat-
ical expression from various initial conditions (green lines); (c) orbits \( \omega(t) \) starting at the same initial conditions, obtained by integrating a trained recurrent neural network (blue lines). Note that in most cases, orbits in green will not be visible because the trained RNN so closely approximates the original dynamical system. Figure captions provide training parameters and error measures. These include \( d_\# \), the number of training datapoints used; MSE, the mean squared error achieved by the trained feedforward network on the vector field \( F(q) \); \( E_{\text{max}} \), the maximum error on the vector field, which corresponds to the quantity in (4.15); \( E_{\text{orb}} \), the maximum error between points from the original system orbits and the RNN orbits (4.5), normalized to the length of the RNN trajectory; \( E_L \), the upper bound on the trajectory error (4.16); and \( t_\# \), the number of timesteps over which the system was integrated.

Example 1: Two fixed point attractors in \( \mathbb{R}^2 \)

We begin with the system of two fixed points given in the previous chapter, representing a simple associative memory with two stored patterns and two basins of attraction.

Figure 4.2 shows six separate orbits \( \omega(t) \) produced by a RNN trained on the regularized vector field of (3.6). These trajectories are superimposed on the vector field itself and orbits \( q(t) \) integrated from it. As can be seen, orbits of the recurrent network converge approximately to the intended fixed points, where they halt, closely tracing the paths of the regularized dynamical system along the way. The RNN reproduces the desired dynamics. For this simple system only \( m = 7 \) hidden neurons were required to achieve a very small trajectory error, and the matrices \( A, B, \) and \( \theta \) after training are

\[
A = \begin{pmatrix}
-0.1263 & -0.2001 & -1.0672 & -0.0366 & 1.9312 & -0.1113 & -0.4771 \\
0.0009 & 0.3941 & 0.0002 & 1.1942 & -0.0014 & -2.1799 & 0.8748
\end{pmatrix},
\]
\[
B = \begin{pmatrix}
-2.7141 & -0.0054 \\
-0.0108 & 2.2352 \\
-2.8188 & -0.0019 \\
-0.0005 & 3.1294 \\
-1.1308 & 0.0002 \\
0.0009 & 0.9245 \\
0.0063 & -1.0284
\end{pmatrix},
\]
\[
\theta = \begin{pmatrix}
-0.6997 \\
1.1148 \\
2.7858 \\
-3.0961 \\
0.9329 \\
-0.8722 \\
-0.6908
\end{pmatrix}.
\]

As reported in the figure caption, the error bound \( e_L \) is very large but the trained RNN does much better, i.e., \( e_{orb} \ll e_L \). This is the case for all of our examples so we will omit reporting \( e_L \) henceforth. We conjecture that this is the case because the systems we approximate by RNN are asymptotically stable. It is clear that if both the original dynamical system and the approximating RNN have stable fixed points or limit cycles in their state spaces, then the maximum trajectory error must be bounded by some constant.

**Example 2: Four fixed point attractors in \( \mathbb{R}^2 \)**

It is possible, of course, to store additional attractors. Figure 4.3 depicts the regularization of four fixed point attractor systems across two boundaries, along with orbits integrated from the regularization and a recurrent network. Again, there is little to distinguish between the systems’ outputs. As might be expected, more attractors require more hidden neurons to capture the complexity of the vector field: To achieve a similar error measure to the first example, \( m = 9 \) hidden neurons were required. We can continue
Figure 4.2: Vector field and orbits of a system with two fixed points. $m = 7$, $d_\# = 800$, $\text{MSE} = 8.1 \times 10^{-6}$, $e_{\text{max}} = 0.0075$, $e_{\text{orb}} = 0.033$, $e_L = 1.6 \times 10^9$, $T = 80$, $t_\# = 1000$.

this process, adding more and more attractors, as long as we increase $m$ commensurately. At this time, the increase is made empirically — that is, by picking a value for $m$, testing a trained network’s performance, and repeating if necessary.

**Example 3: Three limit cycles in $\mathbb{R}^2$ for central pattern generators**

There is no need to restrict our system dynamics to fixed point attractors. The 2-dimensional vector field given by

$$\dot{x} = -y + x(r^2 - x^2 - y^2)$$
$$\dot{y} = x + y(r^2 - x^2 - y^2)$$

exhibits a globally attracting limit cycle, centred at the origin with radius $r$. In contrast to the examples we have discussed previously this is a nonlinear system. Stable limit cycles like this are biologically important because they characterize the dynamics of central pattern generators (CPGs). CPGs are neural networks that produce rhythmic outputs
resembling motor patterns even in isolation from motor and sensory feedback; they serve many functions in animals, playing roles in locomotion, respiration, swallowing, etc. [51]. Their behavior is characterized by interacting processes that alternately increase and decrease as in harmonic motion. We will now build a system of three concentric stable limit cycles. For simplicity, we will use the same basic form for each cycle, only varying the parameter \( r \) above. We will regularize across two boundaries, with the choice of concentric limit cycles dictating that these boundaries take on circular geometry. In particular, we define \( f(x, y) = 0 = x^2 + y^2 - r^2 \). The regularized vector field, the boundaries, and several orbits are given in Figure 4.4. The boundaries divide the state space into three distinct regions: a central disc, an annulus between the two boundaries, and the remainder of \( \mathbb{R}^2 \). Each region contains an attracting limit cycle, to which all orbits within that region converge. Superimposed as usual are orbits generated by a recurrent network trained on the regularized vector field. As desired, these orbits very closely approximate the originals, reproducing the limit cycles and their basins. The complexity of the field, stemming from the nonlinearity of the underlying attractor system, the
Figure 4.4: Vector field and orbits of a system with three limit cycles. $m = 100$, $d_\# = 10000$, MSE = 0.0081, $e_{\text{max}} = 0.31$, $e_{\text{orb}} = 0.0033$, $T = 80$, $t_\# = 5000$.

number of cycles, and the nonlinear boundaries, required a relatively large network with $m = 100$ hidden neurons.

**Example 4: The Van der Pol oscillator**

The well known Van der Pol oscillator is a nonconservative oscillator with nonlinear damping, given by the second-order ODE

$$\ddot{x} + \mu(x^2 - 1)\dot{x} + \omega^2 x = 0.$$  

The parameter $\mu$ gives the damping strength. The system exhibits a limit cycle that becomes increasingly sharpened as $\mu$ increases, and extensions of the model have been used in various fields, most notably to model action potentials in neurons. A 2-dimensional phase portrait of this oscillator, along with very close RNN approximations of the displayed trajectories, is given in Figure 4.5.
Chapter 4. Synthesis of recurrent neural networks

Figure 4.5: Vector field and orbits for the Van der Pol system. $m = 10$, $d_\# = 1600$, $\text{MSE} = 9.2 \times 10^{-5}$, $e_{\text{max}} = 0.031$, $e_{\text{orb}} = 0.022$, $T = 80$, $t_\# = 10000$.

Example 5: The forced Van der Pol oscillator

Shaw [83] discovered that the forced Van der Pol oscillator exhibits steady state chaotic behaviour over a substantial range of parameter values. In particular, rewriting (4.37) as the first-order system

\[
\begin{align*}
\dot{x} &= 0.7y + 10x(0.1 - y^2) \\
\dot{y} &= -x + 0.25 \sin \left( \frac{\pi}{2} t \right)
\end{align*}
\]

and integrating gives rise to a chaotic attractor with interesting topological structure now known as the Birkhoff-Shaw attractor [94]. Although the forcing term always repeats exactly after each period (being a sinusoid) the response cycles never do. In fact they criss-cross repeatedly in the two-dimensional plane, as can be seen in Figure 4.6. Therein we plot a single trajectory integrated from Shaw’s system, along with another trajectory integrated from an approximating RNN. The RNN contained 20 hidden neurons, plus 4 more from a second RNN trained on the sinusoidal driver. These 4 neurons were folded
into the network according to the method given above.

In Figure 4.6 we have used lines instead of point markers to illustrate the attractor structure more clearly. Also note that the plotted vector field comes from the unforced system. Of course the criss-crossing occurs only in the two-dimensional projection from the full three dimensions (including $t$) of the system. For the first time we can see noticeable divergence between the original system’s trajectory and that of its RNN approximation, although structurally the two formed attractors are very similar. The divergence is the result of the extreme sensitivity of chaotic systems.

Example 6: The forced Duffing oscillator

The Duffing equation describes a damped harmonic oscillator with an additional cubic ‘hardening’ term. When driven by sinusoidal forcing the equation has a highly complex response. Fundamental and subharmonic resonances exist, as well as parameter-regimes
Chapter 4. Synthesis of recurrent neural networks

Figure 4.7: Vector field and orbits for the forced Duffing equation. $m = 10^4, d_\# = 1600$, \( \text{MSE} = 4.0 \times 10^{-10}, e_{\text{max}} = 4.4 \times 10^{-5}, e_{\text{ orb}} = 3.2 \times 10^{-4}, T = 80, t_\# = 5000. \)

of chaos. The forced Duffing equation is given by the second-order ODE

\[
\eta^2 \ddot{x} + 2\eta \zeta \dot{x} + x + \alpha x^3 = f_0 \cos(t),
\]

(4.38)

where \( \eta \) is the ratio of the forcing frequency to the undamped, unforced system’s natural frequency, \( \zeta \) is the damping coefficient, \( \alpha \) is the nonlinear stiffness parameter, and \( f_0 \) is the forcing amplitude. A 2-dimensional phase portrait of this oscillator, with parameter settings \( \eta = 2, \zeta = 0.1, \alpha = 0.05, f_0 = 2.5, \) is given in Figure 4.7. Trajectories from the approximating RNN are superimposed. Again we use the method for folding in the forcing function that was outlined above. The forcing RNN contained 4 hidden neurons. As in the previous case, trajectories cross in the projected phase plane; however, in this example, free of chaos, trajectories stay close.

**Example 7: A limit cycle and a fixed point in \( \mathbb{R}^2 \)**

The flexibility of our algorithm enables us to mix and match attractors. In Figure 4.8 we have combined a limit cycle and a fixed point, each defined as above, and regularized
Figure 4.8: Vector field and orbits of a system with one limit cycle and one fixed point. $m = 60$, $d_\# = 18000$, MSE = 0.0017, $\epsilon_{\text{max}} = 0.16$, $\epsilon_{\text{orb}} = 0.068$, $T = 80$, $t_\# = 5000$.

across a circular boundary. Again, training on the regularized field results in a RNN that reproduces the desired dynamics. This example is unique in that the boundary is not a strict separatrix; orbits within the circular boundary all converge to the limit cycle but orbits without may or may not, depending on their initial conditions. They do so if they fall in the ‘shadow’ of the limit cycle that projects outward to the left from the fixed point. A large number of hidden neurons was required here because the system features two very different dynamical regimes ($m = 60$).
Figure 4.9: Orbit of the chaotic Rössler system with RNN approximation. \( m = 40, d_\# = 39304, \text{MSE} = 3.1 \times 10^{-5}, e_{\text{max}} = 0.055, e_{\text{orb}} = 3.6 \times 10^{-4}, T = 80, t_\# = 2500. \)

Example 8: Chaotic attractors in \( \mathbb{R}^3 \)

We now construct a recurrent neural network that replicates the attractor of the Rössler system. This 3-dimensional system is defined as

\[
\begin{align*}
\dot{x} &= -y - z \\
\dot{y} &= x + ay \\
\dot{z} &= b + z(x - c).
\end{align*}
\]

Setting the system parameters to \( a = 0.1, b = 0.1, c = 9 \) yields chaotic dynamics and an attractor with fractal structure, characterized by Cantor-set-like bands and a half-twist as in the Möbius strip. Despite this complexity, a recurrent network trained on the vector field is able closely to reproduce the attractor as shown in Figure 4.9 (the vector field has been omitted for clarity and we have used lines instead of point markers to better illustrate the structure). The depicted RNN contains \( m = 40 \) hidden neurons.
Figure 4.10: Orbit of the chaotic Lorenz system with RNN approximation. $m = 60$, $d_{\#} = 91125$, MSE = 0.0025, $e_{\text{max}} = 0.6395$, $e_{\text{orb}} = 0.0067$, $T = 80$, $t_{\#} = 10000$.

atmospheric convection, is given by

\[
\begin{align*}
\dot{x} &= \sigma(y - x) \\
\dot{y} &= x(\rho - x) - y \\
\dot{z} &= xy - \beta z.
\end{align*}
\]

The system exhibits chaotic behaviour for the parameter settings $\sigma = 10$, $\beta = 8/3$, $\rho = 28$, manifest in the butterfly-like attractor presented in Figure 4.10. The system, having near-periodic orbits circling around two points, is more complicated to learn than the Rössler system and required 60 hidden neurons to replicate approximately via RNN.

**Chaos metrics**

The chaotic systems are the first cases in which significant divergence can be seen between the trajectories of the original systems and those of the approximating RNNs. This divergence should not be unexpected, because of chaotic systems are sensitive. The respective RNNs are not perfect approximators of the original ODEs, and this imperfection in the
dynamics is magnified over time and manifest in the diverging trajectories. However, it appears from the figures that the original strange attractors are closely replicated. The attracting sets of the RNNs and their original vector-field counterparts are very similar and seem to closely bound each other. Nevertheless, it is difficult to judge the quality of the approximations from the figures, particularly in the case of the Lorenz system.

We can better explore the RNN approximations’ accuracy using chaos metrics. The Lyapunov exponents have proven to be a most useful dynamical diagnostic for chaotic systems [110]. Lyapunov exponents are the average exponential rates of divergence or convergence of nearby orbits in phase space. Exponential orbital divergence means that predictive ability is rapidly lost, so that systems with irresolvable initial differences will soon behave quite differently. This sensitivity to initial conditions is, of course, the hallmark of chaos. Thus, any system containing at least one positive Lyapunov exponent is defined to be chaotic, with the magnitude of the exponent reflecting the time scale on which system dynamics become unpredictable [110].

Given a continuous \( n \)-dimensional dynamical system, the Lyapunov exponents may be measured from the long-term evolution of an infinitesimal \( n \)-sphere of initial conditions. The sphere will become an ellipsoid as a result of the locally deforming nature of the flow. The \( i \)th Lyapunov exponent is defined in terms of the length of the ellipsoid’s principal axis \( p_i(t) \):

\[
\lambda_i = \lim_{t \to \infty} \frac{1}{t} \log \frac{p_i(t)}{p_i(0)},
\]

where the exponents are typically ordered from largest to smallest. The exponents measure the rate at which the dynamical system creates or destroys information; they are expressed in units of bits per second. The orientation of the measured ellipsoid changes as it evolves, and so the directions associated with a given exponent vary in a complicated manner [110]. Furthermore, the condition of infinitesimal separations over long time series cannot be guaranteed in chaotic systems. These are the primary difficulties in computing the Lyapunov spectrum.

For systems whose equations of motion are explicitly known, there exists a technique for computing a complete Lyapunov spectrum that circumvents these difficulties. It is
due to Shimada and Nagashima [84]. They employ a tangent-space approach wherein the principal axes are defined by the evolutions via the linearized equations of motion of an orthonormal vector frame anchored to an orbit. By definition, principal axes defined by the linearized system are always infinitesimal relative to an attractor, and the directionality issue is overcome by the use of the Gram-Schmidt reothronomalization procedure on the vector frame [84].

We have implemented the method of [84] in order to quantify chaos in the Lorenz attractor and its RNN approximation (the approximation with the greatest divergence measured from the original system). For the Lorenz system with the parameters given above, the Lyapunov spectrum (in units of nats/second) is $(\lambda_1, \lambda_2, \lambda_3) = (0.905, 0.00, -14.6)$ on the time interval of 80 seconds depicted in Figure 4.10. This matches the spectrum reported by [110]. The positive first exponent indicates the existence of a strange attractor.

Because the RNN approximation of the Lorenz system is $(n + m)$-dimensional (in particular $n + m = 63$), we cannot perform a direct comparison of exponents. However, if we take the largest, middle, and smallest exponents as measured from this system, then we have $(\lambda_1, \lambda_2, \lambda_3) = (0.136, 0.00, -2.19)$ on the same time interval. The approximate attractor is indeed chaotic, as indicated by the positive first exponent. However, the exponents indicate that the RNN system expands and contracts at lower rates than the original system.

The Kaplan-Yorke dimension is another quantifier of chaos. It is conjectured [24] to be equal to the information dimension, one of several dimension measures used to analyze fractals. The Kaplan-Yorke dimension is defined as

$$D_{KY} \doteq j + \frac{\lambda_1 + \ldots + \lambda_j}{|\lambda_{j+1}|},$$

where $j$ is the largest integer for which $\lambda_1 + \ldots + \lambda_j \geq 0$ (as usual, the Lyapunov exponents are ordered from largest to smallest). Inserting exponents from the Lorenz system and its RNN approximation, we find that, in both cases, $D_{KY} = 2.062$. This confirms that the overall structure of the strange attractors is quite similar although the dynamical particulars differ.
The time constant $\tau$

The choice of a very large neuronal time constant ($\tau = 10^6$) is a safe one, meant to guarantee that the conditions enumerated by [27] are satisfied. We can generally decrease $\tau$ by several orders of magnitude with negligible effect on the accuracy of our RNN approximations. When $\tau$ does become too small, fixed points are born in the phase space. This makes intuitive sense, since the time constant governs (inversely) the inhibitory activity of the network. In the case of systems where fixed points already exist, these tend to shift location with the shift becoming noticeable for $\tau$ on the order of $10^2$. For systems with limit cycles, the cycles become stable spirals (not necessarily centred on the original cycle) for $\tau$ on the order of $10^4$. Chaotic systems are quite resilient to the formation of fixed points, although the behavior of the systems changes noticeably with decreasing $\tau$. For instance, in the Lorenz system activity tends to concentrate on one of the two wings as $\tau$ gets small, but a fixed point is not born until $\tau$ is on the order of $10^0$.

4.7 Relation to reservoir computing

The recurrent neural networks produced by our algorithm can be mapped onto a certain class of reservoir computers. Reservoir computers [89, 60] feature a large, randomly recurrently connected sea of neuron units, referred to as the “reservoir,” that feeds forward to a set of output units. Reservoir units, whose states we shall collect in vector $r(t) \in m\mathbb{R}$, feature sigmoidal activation functions, while output units, whose states we shall collect in vector $y(t) \in n\mathbb{R}$, feature linear activation functions. Let us call the random recurrent connections within the reservoir $R \in m\mathbb{R}^m$ and the feedforward connections from reservoir to output units $Q \in n\mathbb{R}^m$. Output states are determined from reservoir states as $y(t) = Qr(t)$. Note that this is an algebraic rather than a differential relation. It is possible to modify the network behaviour such that $\dot{y}(t) = Qr(t)$.

A reservoir computer’s output units may be made to approximate prescribed dynamical orbits through a unique training approach. Typically, only the output weights $Q$ that feed the output units are trained (by, e.g., linear regression or recursive least squares [89]). The random recurrent connections $R$ within the reservoir remain fixed.
Figure 4.11: Structure of a reservoir computer, with recurrently connected reservoir and feedforward connected output unit(s). The reservoir is analogous to the hidden units in our networks. Figure adapted from [89].

during this training. The reservoir may or may not have external inputs, but it features feedback connections from the output units to the reservoir; these, which we shall define as $Q_{\text{back}}$, are also randomly initialized, and fixed throughout training.

Examining Figure 4.11 and recalling eqs. (4.11) and (4.12), it should be clear that the hidden neurons $\eta(t)$ in our networks, which undergo sigmoidal activation and are connected by matrix $C$, act like a reservoir; that is, $\sigma(\eta(t))$ is analogous to $r(t)$ and $C$ is analogous to $R$.

This analogy suggests that recurrently connected hidden-unit sets, trained by our algorithm, can be used as reservoirs. Our feedforward training could be used to generate reservoirs that are tailored to desired dynamical regimes, such as limit cycles or chaos. Specific chaotic attractors with desirable properties could be trained and installed in reservoirs. The hidden-unit sets in our RNNs evolve autonomously from the output units since there are no feedback connections, and thus it is simply a matter of connecting them via $Q$ to an output-unit set $y$, and then connecting the outputs back to the reservoir with a randomly initialized $Q_{\text{back}}$.

The FORCE methodology of Sussillo and Abbott [89], used to train the output connections $Q$, was shown to adapt the behavior of random chaotic reservoirs to achieve various ordered output tasks, including human motion modeling. We conjecture that
tailored reservoirs could greatly improve the adaptability and performance of reservoir computers in general, although such investigation is beyond the scope of this research.

We can extend the analogy between reservoir computers and our recurrent networks further, by taking the previously proposed differential relation for $y(t)$. Then the linear outputs in reservoir computing act like the output neurons $\omega(t)$ in our networks, with their connections $A$ from the hidden neurons; that is, $\omega(t)$ is analogous to $y(t)$ and $A$ is analogous to $Q$. In our networks, feedback connections such as $Q_{\text{back}}$ do not exist, but would have to be inserted in the lower left block of $W$. Funahashi and Nakamura explain that introducing such connections will not significantly alter the internal hidden-unit dynamics as long as the connection weights are small [27].

RNNs produced by our algorithm thus represent a particular class of reservoir computers, in which the output unit states are described by a differential equation and the output matrix $Q$ decomposes the reservoir matrix according to $R = PQ$, where $P \in m \times n$ is arbitrary. Note that the matrix $P$ plays the same role in this equation as does $B$ in our RNNs. We can deduce then that this class of reservoir computers has an equivalence to three-layer feedforward neural networks by the mathematics of Funahashi and Nakamura [27]. It is possible that this equivalence, and the attendant vector-field perspective, may shed light on some of the unanswered questions regarding reservoir properties and network dynamics (see [60]).

### 4.8 Algebraic approximation of hidden neuron states

The recurrent neural networks we construct by our algorithm contain $n$ output neurons with states $o_i, i = 1, \ldots, n$, and $m$ hidden neurons with states $h_j, j = 1, \ldots, m$. The total network state is represented by the vector quantity $s = \text{col}[\omega, \eta]$. The function of the output neurons is clear: their states replicate the trajectories of the prescribed $n$-dimensional dynamical system being modelled. The hidden neurons are required to support this function, but their behaviour is more opaque. In this section we develop an algebraic approximation to the hidden neurons states. We show that, to within a small difference that increases linearly with time, the hidden states are given by an affine
transformation of the output states. The existence of such an approximation makes intuitive sense because the underlying system being modelled is $n$-dimensional. Replacing a differential equation with an algebraic one also facilitates further network analysis, as we will show in the next section on potential functions.

Recall that the hidden neurons are initialized at $t = 0$ as $\eta(0) = B\omega(0) + \theta$. Let $r(t) = B\omega(t) + \theta$. Further, let $\delta(t) = ||\eta(t) - r(t)||$, and note that $\delta(0) = 0$ exactly. Our aim is to show that $\dot{\delta}(t) = c$, $0 < c \ll 1$, so that $\delta(t) \approx 0$ for $t$ sufficiently less than $\tau$ (these are the timescales we are interested in). Note that such a statement should be true because otherwise trajectories initialized at different points along a single flow would not match. By this logic, and by the similarity between the form of the initial condition and that of the approximation, the validity of approximation does not come as a surprise. We demonstrate it mathematically for the sake of completeness.

Dropping the explicit time dependence and using the standard RNN equations, we have that

$$\dot{\omega} = -\frac{1}{\tau}\omega + A\sigma(\eta) \quad (4.39)$$

and

$$\dot{\eta} = -\frac{1}{\tau}\eta + BA\sigma(\eta), \quad (4.40)$$

whence

$$\dot{r} = -\frac{1}{\tau}r + BA\sigma(\eta) + \frac{1}{\tau}\theta. \quad (4.41)$$

Thus, if $r \approx \eta$ then

$$\dot{r} - \dot{\eta} \approx \tau^{-1}\theta. \quad (4.42)$$

The assumption that $r \approx \eta$ may seem circular, but recall that it is exactly true at $t = 0$. In this sense, we are almost performing an ‘induction’ on time.

Of course there must be some divergence between $r$ and $\eta$ as the system evolves. Integrating (4.42) with respect to time yields $r - \eta \approx \tau^{-1}\theta t + c$, then taking magnitudes and deriving with respect to time gives

$$\dot{\delta} = \frac{d}{dt}||r - \eta|| = \frac{1}{\tau}||\theta||. \quad (4.43)$$
Figure 4.12: Evolution of $\delta(t)$ over the course of an orbit from Figure 4.2.

Since $\tau \gg 1$, $\dot{\delta}$ is small. This is the desired result: we have shown that when $\delta$ is small, so is its rate of change (which is linear in time); since $\delta$ is initialized at zero by the network definitions, it remains so for timescales such that $t \ll \tau$. To verify (4.43) and confirm that we are safe approximating the states of the hidden neurons with $r(t)$, we have computed $\delta(t)$ for various trajectories in synthesized recurrent neural networks.

For example, in the two-attractor system depicted in Figure 4.2, it happens that $
abla^{-1}\|\theta\| = 4.7756 \times 10^{-6}$. The evolution of $\delta(t)$ along one of the depicted orbits is plotted in Figure 4.12, from which we see that it is a linear function of time whose slope almost perfectly agrees with $\nabla^{-1}\|\theta\|$. This has been the case for all networks we have evaluated.

Hence, we can be confident that the hidden unit states $\eta(t)$ remain sufficiently close to $r(t)$ over the course of an orbit for the purposes of approximation. Then the algebraic expression for $r(t)$ can be used as an approximator for the hidden neuron states. This means that we can reduce our system from $n + m$ ordinary differential equations to only $n$ differential equations. This is intuitively pleasing since the original dynamical systems being approximated are $n$-dimensional. Writing the RNN equations only in terms of the
output neurons and the algebraic approximation for the hidden units, we have

\[ \dot{\omega} = -\frac{1}{\tau} \omega + W \sigma \left( \begin{array}{c} \omega \\ r \end{array} \right) \in n \mathbb{R}. \]

4.9 Potential energy functions for recurrent networks

We now discuss how potential energy functions for our synthesized recurrent neural networks may be computed.

Mendes and Duarte [66] showed that additive neural networks can be decomposed as the sum of a gradient system and a Hamiltonian system as follows:

\[ \dot{s} = P(s) \nabla V(s) + Q(s) \nabla H(s) + Eu(t). \]  (4.44)

The scalar functions \( V \) and \( H \) are the gradient system’s potential function and the Hamiltonian system’s potential function, respectively. The third term represents inputs to the network.

The appropriate potential functions for this decomposition are given, respectively, by

\[ V(s) = \sum_{i=1}^{n} \int_{s_i}^{s_i} a_i(\zeta_i) f_i'(\zeta_i) d\zeta_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}^S f_i(s_i) f_j(s_j) \]  (4.45)

and

\[ H(s) = \sum_{i=1}^{n} \int_{s_i}^{s_i} f_i(\zeta_i) d\zeta_i, \]  (4.46)

where \( a_i(s_i) \) is the refractory parameter (in our networks \( a_i = \tau^{-1} \forall i \)), \( f_i(s_i) \) is the neuron activation function (in our networks \( f_i(s_i) = \sigma(s_i) \forall i \), the logistic sigmoid function), \( s_i \) is the state of the \( i \)th neuron, and \( w_{ij}^S \) is the \((i,j)\)-entry in the symmetric decomposition of the network’s weight matrix.

Here we can recognize a symptom of the space mismatch inherent to dynamical approximation by recurrent neural networks. The system being approximated is \( n \)-dimensional and its potential functions lie in \( n \)-dimensional space. On the other hand, the potential functions (4.45) and (4.46) depend on \( n + m \) neurons—a point in \( n \mathbb{R} \) does
not provide enough input information for the computation of potentials. We can determine potentials along integrated trajectories since we integrate for the hidden states also, but to determine potentials over the entire output space in this manner requires a great deal of work.

To determine the potentials as functions of output space only, it is also possible to use the initial condition, whose hidden component we now recognize as our approximator \( r(t) \). By the mathematics of the previous section, it is now known that this expression holds (approximately) true throughout the course of a trajectory, not just at its initial point. Therefore we will use it to simplify our computations and determine the potentials as functions of the output space only.

Since \( r \) is a function of \( \omega \), to close approximation the gradient and Hamiltonian potentials can be represented purely as functions of the output neuron states, as desired. Looking back at (4.45), we note that we can also simplify the expression by neglecting the first term, again because \( \tau \gg 1 \). The approximate gradient potential is then given by

\[
V_\omega(y) \triangleq -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}^s \sigma_i(y_i) \sigma_j(y_j),
\]

where now \( y = \text{col}[\omega, r] = \text{col}[\omega, B\omega + \theta] \). In Figure 4.13 we plot this approximate gradient for the two-attractor system of Figure 4.2. The two basins of attraction and the saddle that separates them are clearly visible in the gradient potential, as is expected. The approximate Hamiltonian potential,

\[
H_\omega(y) \triangleq \sum_{i=1}^{n} \ln(1 + e^{\mu_i}),
\]

is calculated analogously and given in Figure 4.14; it is far less interesting, but that is not a surprise given that the two-attractor system is entirely dissipative.

In Figures 4.15 and 4.16, respectively, we present the approximate gradient and Hamiltonian for the system of one fixed point and one limit cycle shown in Figure 4.8.
Figure 4.13: Gradient potential for the system depicted in Figure 4.2.

Figure 4.14: Hamiltonian potential for the system depicted in Figure 4.2.
Figure 4.15: Gradient potential for the system depicted in Figure 4.8. Note that figure has been rotated for better viewing.

Figure 4.16: Hamiltonian potential for the system depicted in Figure 4.8. Note that figure has been rotated.
4.10 A note on network potentials and catastrophic forgetting

Considering (4.47), it is clear that the gradient is made up of a sum of products of sigmoids. It seems strange that such complicated functions as depicted in Figures 4.13 and 4.15 could be built up thus, until we realize that the arguments of many of these sigmoids (those corresponding to hidden neurons) have undergone an affine transformation—in particular, the transformation \( r = B\omega + \theta \). Hence the space on which these sigmoids are evaluated is warped. For the hidden neurons, \( y_i, y_j \), and \( w_{ij}^S \) are all functions of the weight matrix \( B \). It is interesting to note that because the Hamiltonian is not an explicit function of the recurrent network weights, its only dependence on weights comes from the aforementioned affine transformation. Hence the Hamiltonian depends only on \( B \).

Recall that \( B \) is the matrix that connects input neurons to hidden neurons in the three-layer feedforward network out of which a recurrent network is formed; likewise \( A \) is the matrix that connects hidden neurons to output neurons; the recurrent weight matrix \( W \) is given by (4.11).

This means that only the feedforward connections from input neurons to hidden neurons shape the Hamiltonian potential of a recurrent network generated by our algorithm.

Despite the hidden complexity of these potential functions, their additive form suggests a method to combine separate networks together. Such combination would be an important step in the understanding of how attractors can be added incrementally to recurrent neural networks. In fact, the original vision for state-space sculpting was, metaphorically, to carve channels and basins in the potential function of a neural network the way canals or irrigation systems are carved into land.

It should be clear from the forms of \( V_\omega \) and \( H_\omega \) that, given two recurrent networks defined by weight matrices

\[
W_1 = \begin{pmatrix} 0 & A_1 \\ 0 & B_1A_1 \end{pmatrix}, \quad W_2 = \begin{pmatrix} 0 & A_2 \\ 0 & B_2A_2 \end{pmatrix},
\] (4.49)
the recurrent neural network defined by

\[
W_c = \begin{pmatrix}
0 & A_1 & A_2 \\
0 & B_1A_1 & 0 \\
0 & 0 & B_2A_2
\end{pmatrix}
\] (4.50)

will have as its gradient and Hamiltonian potentials the sums of the respective potentials from the original networks. Now if these original potentials are flat outside of nonoverlapping regions, then their sum defines a state space in which both regions exist unaltered. This is one way we could combine distinct attractors.

Unfortunately, (4.50) is not quite a valid RNN weight matrix. If we define \( A_c = [A_1 \ A_2] \) and \( B_c = [B_1 \ B_2]^T \), then

\[
W'_c = \begin{pmatrix}
0 & A_c \\
0 & B_cA_c
\end{pmatrix}
\] (4.51)

is a valid RNN matrix. We must reconcile these two so that \( W_c \) and \( W'_c \) yield the same potentials. In fact, \( W_c \) and \( W'_c \) already generate the same Hamiltonian potential since \( H_\omega \) depends only on \( B_c \). Equality of their gradient potentials, on the other hand, can be imposed by the condition

\[
B_2A_1 = -(B_1A_2)^T.
\] (4.52)

That is because (4.52) leads to zeros in the appropriate locations in the symmetric decomposition of \( W'_c \). Assuming the two networks to be joined are the same size, \( A_i \in {}^nR^m \) and \( B_j \in {}^mR^n \) where generally \( m > n \). Thus (4.52) leads to a set of \( m^2 \) equations in \( 2mn \) unknowns. As a consequence, it is unlikely that \( A_2 \) and \( B_2 \) can be found to satisfy (4.52) unless \( 2n > m \), and even then, \( A_2 \) and \( B_2 \) must also satisfy whatever constraints are necessary to generate the prescribed recurrent neural network. As such, it appears that this approach to combining existing networks is not viable.

Of course, the constraint disappears if we are allowed to alter \( A_1 \) and \( B_1 \). This conflict highlights one of the inherent difficulties of information storage in neural networks, namely, that anything new must be harmonized with all that is old. Otherwise, the
phenomenon of catastrophic forgetting occurs whereby new information overwrites information already stored in the network connections. Catastrophic forgetting is well known and well studied as we have indicated in our literature review, but it is pleasing that within our RNN framework it has a simple, specific mathematical explanation given by the generally unsatisfiable condition (4.52). We conclude by noting that the necessity of harmonizing old and new in neural networks is also a strength, since it leads to a deep interconnectedness of stored information.
Chapter 5

The episodic memory model

5.1 Overview

We are now ready to construct a computational model for episodic memory. The model is inspired by the function of the hippocampus and its interactions with the cortex. It comprises two neural components: a hippocampal analogue, manifest as a recurrent neural network, and a cortical analogue, manifest as a deep autoencoder. Together, these components encode incoming experiences into episodic memory traces, store those traces as associative orbits in a dynamical system, and remember stored experiences through dynamical evolution and episode reconstruction.

The preceding chapters have discussed synthesis of the hippocampal analogue: A recurrent neural network is trained to replicate a prescribed, possibly sculpted dynamical system that functions as an associative memory. In the following we explain the cortical analogue and then show how the two component networks are combined to store and reconstruct episodes.

5.2 Cortical model

We model the constructive function of the cortex with a deep autoencoder (of course the cortex serves many cognitive functions, but we focus only on its contribution to episodic memory). The deep autoencoder is a flavour of neural network conceived for
dimensionality reduction [47], that is, the determination of compact representations of high-dimensional data. Such representations are called codes. However, autoencoders can also be run in reverse to reconstruct high-dimensional data from compact codes. Our model takes advantage of both functions, but the reconstructive analogy in particular should be clear.

Deep autoencoders are stochastic feedforward neural networks based on the restricted Boltzmann machine (see [47] and references therein for details). The restricted Boltzmann machine (RBM) is a two-layer network with undirected connections between layers and no connections within a layer, as depicted in Figure 5.1a; one layer’s neurons model visible data (units $v \in \mathbb{R}^n$ for $n$-dimensional data), while the other layer’s neurons model stochastic hidden features (units $h \in \mathbb{R}^m$ for $m$ hidden features). A joint configuration $(v, h)$ of visible and hidden neurons has an energy given by

$$E(v, h) = - \sum_{i \in \text{data}} b_i v_i - \sum_{j \in \text{features}} b_j h_j - \sum_{i,j} v_i h_j w_{ij}, \quad (5.1)$$

where $v_i$ and $h_j$ are the states of visible unit $i$ and hidden unit $j$, $b_i$ and $b_j$ are their biases, and $w_{ij}$ is the weight between them. The network assigns a probability to datapoints using this energy function.

Hidden features in the data are extracted as follows. When an RBM is given an input datapoint this sets the states of the visible neurons $v$; then the state of each hidden neuron $h_j$ is updated in one of two ways. For binary hidden neurons, $h_j$ is set to 1 with probability $p(h_j) = \sigma(b_j + \sum_i v_i w_{ij})$ ($\sigma$ is again the logistic function). For continuous neurons, the update rule is to sample from a Gaussian distribution with unit variance and mean $b_i + \sum_i v_i w_{ij}$; in this case $p(h_j) = \sum_i v_i w_{ij}$.

The probability of a datapoint can be increased by adjusting the weights and biases to lower its energy. This is the function of training, which naturally leads to the selection of “good” hidden features for data reconstruction. Training proceeds by producing a confabulation by updating the visible neurons based on the states of the hidden neurons; specifically, setting $v_i$ to 1 with probability $\sigma(b_i + \sum_j h_j w_{ij})$ for binary neurons and analogously to above for continuous neurons. The hidden neurons are then updated once
more to represent the hidden features of the confabulation. The change in weights is given by

$$\Delta W = \alpha((vh^T)|_{\text{data}} - (vh^T)|_{\text{confab}}),$$

(5.2)

where $\alpha$ is a learning rate. A simplified version of (5.2) is used for the biases. Note that learning to this point is unsupervised.

An autoencoder is produced by stacking RBMs. After learning the weights in one RBM, the activities of its hidden neurons (when being driven by input data) can in turn be used as data for the visible neurons of a second RBM (see Figure 5.1b). Training in the second RBM proceeds as before, and this layer-by-layer learning can be repeated as many times as desired. For dimensionality reduction, each successive layer typically has fewer neurons than the last, resulting in a small set of hidden features at the highest level that distill underlying order in the data. When a datapoint is fed into the visible units at a trained autoencoder’s lowest level, the highest-level neuron activations comprise that datapoint’s code. As mentioned, the network can be run in reverse, starting with a code and updating successively lower layers until a datapoint is reconstructed at the visible neurons. Note that after training is complete, the autoencoder is always run in a feedforward sense, either bottom-to-top or top-to-bottom.

Figure 5.1: a) A restricted Boltzmann machine. b) Two stacked RBMs forming an autoencoder.
In the next subsection we summarize how the autoencoder’s compression and recon-
struction capabilities can be combined with the associative function of recurrent neural
networks. The result is an episodic memory model inspired by the hippocampus’s inter-
action with the cortex.

5.3 Cortico-hippocampal interaction

To summarize the two components of our model: There is (i) a recurrent neural network
in which arbitrary trajectories may be made attracting, and (ii) a deep autoencoder that
can both compress high-dimensional data and reconstruct it from compressed codes.
Conceptually, these parts interact for the storage and retrieval of memory episodes as
follows.

Say the system is presented with an episode. For illustrative purposes this consists
of a movie-like sequence of image frames through time, as is depicted in Figure 5.3.
Instantaneous input from this sequence is fed through a trained autoencoder to produce
a code in some code space, and the set of all such codes over the episode’s course forms a
trajectory. In practice such sets will not form continuous curves in code space but we fit
curves through them. To render this episode an associative memory, a recurrent neural
network is trained according to the algorithm in Chapter 4 such that the code-space
trajectory is attracting. We refer to the embedded trajectory as the episodic memory
trace. When future input is presented to the system, that input is compressed by the
autoencoder leading to an initial condition for the RNN. The RNN evolves through time
from this initial condition, converging on the attracting memory trace and following it. As
it follows the memory trace the RNN’s output is fed through the autoencoder in reverse.
This reconstructs the full episode at the encoder’s visible neurons. The combination of
dynamical evolution and reconstruction embodies the mental simulation of the episode.

A schematic depiction of the model can be found in Figure 5.2. We explicitly demon-
strate the storage and retrieval of memory episodes in the next section.
5.4 Storage and retrieval examples

Figure 5.3 shows eight sequential frames of an abstract figure running (this looks more realistic as a looped animation). This sequence will act as an episode to be stored and retrieved by our model. Since running is cyclical, the episode’s memory trace in the hippocampal RNN will be stored in the form of an attracting limit cycle. The dimensionality of the data is an issue here, but this is where the compressive function of the autoencoder comes in. The $(52 \times 40)$-pixel grayscale images composing the episode exist in $[0, 1]^{(52\times40)}$, a space in which it would be practically vexing (although theoretically possible) to construct a limit cycle. It is clear, however, that a comparatively small number of variables underlies the depicted motion—there are eight joints that undergo coupled oscillations. This makes the images good candidates for dimensionality reduction.

The first step in building up the memory model is to train an autoencoder on the image set. This is done according to (5.2), using code provided in the supplementary
materials of [47] that we have modified. The autoencoder we train features 6 layers of neurons, sized as $2100 \rightarrow 2500 \rightarrow 700 \rightarrow 350 \rightarrow 40 \rightarrow 2$. Neurons in the bottom and top layers are continuous, while those in the intervening layers are binary. Two neurons were chosen for the top layer because $\mathbb{R}^2$ is the minimal space in which a limit cycle can be embedded.

Training the autoencoder to compress/reconstruct the image data builds up an internal representation of that data. After training, running the autoencoder upwards on each image generates a code (i.e., a point in $\mathbb{R}^2$). Likewise, running the encoder downwards on a code generates an approximate reconstruction of the corresponding image.

In Figure 5.4 we depict the code space of our trained autoencoder by plotting code points in $\mathbb{R}^2$ along with the images that are reconstructed from them. These codes can be seen to lie approximately on a smooth curve. Although it is always possible to form a curve through a set of points in $\mathbb{R}^2$, the ‘niceness’ of this curve is in some sense fortuitous, because not every autoencoder training session leads to a code space in which this is so. In another sense this curve is expected, since, as mentioned, a small number of oscillating variables underlies the running motion. Autoencoders are very good at extracting this sort of hidden order.

The entire eight-frame running episode can be reconstructed from a trajectory through these points. A benefit of the autoencoder is that, because of its internal representation, a continuous path in the code space yields an intuitively continuous image flow at its visible neurons. By “intuitively continuous” we mean that the image flow looks natural to the human eye.

Now we need to construct the desired episode trajectory. First we approximate a curve through the points in Figure 5.4 by fitting an eight-term Fourier series in polar coordinates. This series, given by $r = f_8(\theta)$, is also plotted in the figure, in red. Although it does not pass exactly through the codes, we will see that it generates good image reconstructions nonetheless.

Our goal is to have the hippocampal RNN approximate a dynamical system having $r = f_8(\theta)$ as its attracting limit cycle. We can construct such a dynamical system according to the methods of Sverdlove [91]. Recalling Chapter 2, it is proved in [91] that,
Figure 5.4: Codes in $\mathbb{R}^2$ and the images they reconstruct when fed back through a trained autoencoder. A fitted Fourier series is superimposed.

with a prescribed curve represented by $L(x, y) = 0$, the vector field

$$\dot{x} = \frac{\partial L}{\partial y} + LH \frac{\partial L}{\partial x}$$

$$\dot{y} = -\frac{\partial L}{\partial x} + LH \frac{\partial L}{\partial y}$$

has $L(x, y) = 0$ as its only limit cycle. Furthermore, this cycle is attracting if the arbitrary function $H(x, y) < 0$ in a neighbourhood of it. In this case we use $L(x, y) = 0 = x^2 + y^2 - f_8(\tan^{-1}(y/x))$ and $H(x, y) = \alpha(x^2 + y^2 - f_8^2 - \beta)$, where $\alpha$ can be used to adjust the convergence speed of the limit cycle and $\beta$ determines how large the attractor’s basin is.

The vector field so constructed, along with an integrated trajectory (green), is shown in Figure 5.5. The algorithm of Chapter 4 can now be used to train the hippocampal RNN on this vector data. Figure 5.5 shows a trajectory (blue) produced by the trained RNN, which indicates that an approximation of the desired limit cycle has indeed been stored therein. The RNN contains 2 output neurons and 60 hidden neurons.

Finally we combine the RNN with the autoencoder. As the network evolves through time, converging from its initial condition in code space onto $L = 0$, the states of the
RNN output units are fed backward through the autoencoder. The autoencoder’s visible neurons then reconstruct the episode. Figure 5.6 shows a corrupted input image used as initial condition along with eight frames sampled from the system thereafter at 100-timestep intervals, representing about one full pass around $L = 0$. These frames closely match the originals from Figure 5.3. Continued evolution of the system yields a looped animation that is strikingly evocative of running, and manifests a remembered visual episode stored entirely in neural structures.

This episode is associative. We can fine-tune the limit cycle’s attractor basin in code space, for example by adjusting the region in which $H < 0$; then the system will converge to the episode from all input images whose codes fall within the tailored basin. It is also possible to store and retrieve multiple episodes. All that is required is a suitably trained autoencoder and a recurrent neural network with multiple stored attractors.
Figure 5.7 depicts two limit cycles along with superimposed images. As before, the limit cycles (in this case ellipses for simplicity) lie in the code space of an autoencoder. This time the autoencoder has been trained on frames from the previous running episode as well as an additional walking episode. The code space structure can be inferred from the superimposed images. The upper limit cycle remembers the running episode (shown in the top row of Figure 5.8), while the lower cycle remembers the walking episode (shown in Figure 5.8’s lower row).

The two limit cycles are constructed according to Sverdlove’s method and then embedded in a hippocampal RNN. In this case there is one tweak: To embed multiple limit cycles in a single dynamical system, we have used regularization. The dashed red line in Figure 5.7 represents the separatrix between the cycles’ attractor basins, over which we have design control [97]. Here we have used a linear boundary for the sake of simplicity.

Figure 5.8 shows two corrupted input images used as initial conditions (each initial image code lying in a different attractor basin), along with frames sampled from the system thereafter at regular intervals. These sampled frames clearly reconstruct the desired running and walking episodes. The RNN trained on this more complicated multiattractor system contained 100 hidden neurons.

We note that the remembered running episode consists of fewer frames than did the original; this is because we have used elliptical limit cycles rather than the more complicated (but ultimately more accurate) Fourier series of before. This is also the case for the walking episode, though we did not depict its original frames. Note also that, equally important, the vector field synthesis techniques in Chapter 3 and [91] enable us to adjust the speed of each limit cycle independently. In this example, the walking episode proceeds at approximately one-quarter the speed of the running episode (frames in the lower row of Figure 5.8 were sampled at one-quarter the frequency).

The pair of synthesized ellipses does not intersect in the code space, fortunately, but as more episodes are stored in such a system it seems inevitable that intersections will occur. This issue can always be circumvented by moving to a higher-dimensional code space. Although both these episodes were cyclical, and therefore manifest as limit cycles, it was shown in Chapter 4 that RNNs can be trained to manifest fixed points and strange
Figure 5.7: Orbits integrated from a trained RNN, along with attractor-basin separatrix and superimposed frame reconstructions. The RNN remembers two unique motion episodes shown in Figure 5.8.

attractors too. This enables the model to store episodes of a different nature.

The visual episodes we have presented are reminiscent of the motion modelling achieved in [93], using binary latent variables; however, motions in [93] were represented as joint angles. The motion episodes here are entirely visual; there are no explicit low-dimensional variables, rather the model must develop its own low-dimensional representation (in the code space) through an unsupervised process. Our model is applicable to any visual episode, not only cyclical joint motions, and can be extended to different sensory modes

Figure 5.8: Input images and reconstructed frames of running and walking episodes. Frames are reconstructed by an autoencoder-RNN memory system from the elliptical limit cycles in Figure 5.7.
such as audition.

5.5 Conclusion

Our episodic memory model has demonstrated its ability to store and retrieve animated, dynamic episodes. Such an ability has never been demonstrated before, to our knowledge (although the memory model of Hasselmo [42] can associate sensory inputs to points along a stored trajectory, its representational capacity for high-dimensional inputs is limited by its strictly Hebbian nature and its lack of layer-wise depth).

In our model, dynamic traces are stored naturally within dynamical systems, as attracting trajectories. Our synthesis techniques give us a great deal of control over these trajectories and the neighbourhoods in state space over which they are attracting. The model has two layers of what might be called pattern convergence: first, high-dimensional images converge to low-dimensional codes in an autoencoder; second, arbitrary initial codes converge to attractor codes in the state space of a recurrent neural network. The first convergence is based on probability distributions and the second on dynamical evolution.

Memory recall in our model occurs as output from the hippocampal RNN feeds through the generative autoencoder to reconstruct a sensory experience. It is important to discuss how this model of sensory reconstruction is not homuncular. We have used the analogy of replayed “movies” to characterize reconstructed episodes, and surely there is no homunculus to “watch” these movies in the brain. However, in our view these reconstructions exist in the same representational space that the brain uses for incoming sensory experience; thus, the episodes are not watched by some other brain process but “relived” in the same modality as they were originally experienced — the same patterns of activity (to some approximation) are reinstated in the appropriate cortical circuits.

The idea of memory as a system of attractors is not new, though it is intuitive. The question of how memory is actually stored in the brain is still open, and so our model could very well be biologically inaccurate in terms of the details, though it is phenomenologically sound. Our use of RNNs was inspired by the recurrent structure
of the hippocampus, and dynamical systems made sense for the modelling of dynamic episodes, but it remains to be seen if the model can provide us with any insight into the workings of the mind. Questions like these will be discussed in the next chapter.
Chapter 6

Discussion and Future Work

6.1 Dynamical systems: sculpting and exploring

If human memory is based in dynamical systems then those systems must be rather complex. We can imagine subsystems on submanifolds of the state space that overlap and interleave, so that a higher-dimensional region can be associated with multiple attractors. It might also be possible for a point or locus of points to be attracting within a manifold but repelling elsewhere, or for manifolds to be warped and twisted like wormholes, connecting seemingly distant neighborhoods of the state space over shared similarities. Attractor submanifolds might be embedded in higher dimensional spaces where, off the submanifold, chaotic dynamics dominate. Freeman and Barrie [25] conjecture that such chaos might prepare the system for rapid convergence to a memory, and Kaneko and Tsuda’s model of chaotic itinerancy [55] posits that shifts between low-dimensional ordered behaviour on attractor “ruins” and intervening high-dimensional chaos may act as the neural correlate of a temporal episode. Attractor ruins are attractors from which it is possible to escape, eventually.

All of this is rather abstract, conceptually. The point is that we have only scratched the surface of what is possible in high-dimensional dynamical systems, and many of the bizarre phenomena that characterize those systems may be useful to memory. A concept that we have not explored to great depth in terms of its memory applications is the effect of system inputs. Of course human memory is highly sensitive to context and what is
occurring in the outside world at the time of recall. Inputs to a dynamical system are one way to model this notion of context. We have seen some examples of forced systems in vector-field form and as synthesized RNNs, so exploration along this line is certainly possible within the framework we have developed. This should be a goal in future work.

The technique of regularization combined with Sverdlove’s solution to the inverse problem has enabled us to construct dynamical systems with desired state-space structure and various embedded attractors. Most of the sculpted vector fields that we presented in examples were two-dimensional, and indeed, Sverdlove’s solution is limited to two-dimensional manifolds. Extending his methods for the construction of prescribed fixed points and limit cycles should be a goal of future research since it will facilitate the construction of higher-dimensional dynamical systems. In higher dimensions, more interesting dynamics including chaos can occur.

Stochastic dynamical systems are another possibility for exploration. In stochastic systems, boundaries between attractors are not necessarily “hard” and attractors themselves lose their permanence. Stochasticity could provide memory systems with a greater degree of flexibility and context-dependence. It should be straightforward to realize stochastic systems as RNNs by training the RNNs on the deterministic dynamics and then adding a source of randomness with the same moments as that in the original system.

Finally, we note that the construction of such complex dynamical systems as have been theorized here is not necessarily required for the synthesis of memory systems. In the next section we will discuss the possibility of training recurrent neural networks directly on trajectory data from observed dynamical systems (rather than constructed ones). If an algorithm for that kind of training can be developed then we can circumvent the rather artificial step of constructing a vector field for training data.
6.2 RNN synthesis: limitations, extensions, and biological plausibility

Our algorithm for RNN synthesis has proved successful in generating recurrent neural networks that replicate general dynamical systems. This is to be expected, since the synthesis process is founded upon the rigorous mathematics of Funahashi and Nakamura [27]. The one potential stumbling block is the training phase. Although parameters are guaranteed to exist such that the feedforward neural network defined by $m$, $A$, $B$, and $\theta$ approximates $F(q)$ sufficiently closely, Levenberg-Marquardt backpropagation is not guaranteed to find them—gradient-descent training methods are always susceptible to getting caught at local minima. In practice, we have found that this difficulty can usually be overcome by increasing the number of hidden neurons and/or the density of the training data.

A more fundamental issue is the required three-layer structure of the feedforward network. This limits us to a single hidden layer of neurons. Although this is all that is theoretically required to reproduce any smooth function [26], it can again be problematic in practice. To compensate for the network’s shallowness, the hidden layer must be made large to accurately reproduce more complicated systems; a deeper network might be able to capture the same complexity more parsimoniously using higher-level dependencies. While hidden-layer size is not an issue in and of itself, the large recurrent weight matrices that result are more difficult to analyze. Unfortunately it is not immediately clear if this issue can be overcome. A feedforward network with 2 hidden layers would take the form

$$F_{NN}(q) = A_2 \sigma(A_1 \sigma(Bq + \theta_1) + \theta_2).$$

Transforming this to recurrent form by a process analogous to that outlined in Chapter 4 leads to a weight matrix with nonconstant terms coming from the derivative of $\sigma$. This means that $W$ is itself not constant, and the resultant RNN cannot be expressed in the form of (4.1). In this case the weight matrix $W$ becomes dynamical, evolving through time. Such a system is certainly interesting, and might prove illuminating for
concepts and modelling of working memory (also known as short-term memory). In various theories of working memory it is proposed that, as neurons fire, their neurotransmitter stores are depleted and that this pattern of depletion is iconic of the remembered stimulus \cite{68}. This temporary depletion means that effective synaptic strengths can vary on much shorter timescales. However, this is beyond the scope of this research.

On the other hand, the use of straightforward backpropagation and the limit to three network layers have some benefits. First, backpropagation is well studied and well understood, and numerous augmentations and heuristic tricks are known for improving its performance. Second, the limit to three layers means that error signals propagate locally during training. Backpropagation itself is biologically plausible in the sense that the action potential of a neuron creates a voltage spike both at the end of the axon (normal propagation) and back through to the dendrites \cite{104}. It has been shown that this process can be used to adjust synaptic strengths in biological networks just as the backpropagation algorithm can be used for machine learning; however, one of the primary criticisms of the plausibility of machine-learning-style backpropagation is the need for error signals to travel large distances (through many connections and layers) within networks \cite{12}. There is no known biological mechanism by which this might occur. But in a three-layer network, error signals need only travel across the single hidden layer. This renders learning in such shallow networks more realistic, biologically.

That being said, we are not aware of a biological mechanism by which the transformation from feedforward network to recurrent network might be enacted. That transformation involves rewiring an existing network with connections defined by a matrix product of previously learned connections (or alternatively, constructing a network with those matrix-product connections over another group of neurons).

As the examples of Chapter 4 demonstrate, the training approach has been successful in spite of the aforementioned limitations. Moreover, what a training approach relinquishes in exactness it makes up in flexibility. We can define analytically intractable vector fields through recursive regularization and then use these for training input just the same as we would the exceedingly simple field $F(x, y) = (1, 1)$. However, our training approach is not biologically plausible as a learning mechanism for further reasons. First,
the synthesis of episodic traces, as described in Chapter 5, is not autonomous: a curve was fitted through discrete code points, this curve was made the attracting limit cycle in a 2-dimensional dynamical system according to [91], and that system was taught to a RNN by supervised backpropagation. The model would be considerably more plausible and more powerful if this process were autonomous. Second and related, it is hard to imagine a realistic situation in which global, fine-grained samples from a vector field and its domain are available.

With these two points in mind, we have performed some preliminary experiments on the incremental training of RNNs from trajectory data. More specifically: supposing an episode comes from observed experience, it should consist of a continuous stream of input images (in the visual case); propagating this stream upwards through an autoencoder yields (ideally) a continuous trajectory in code space; tangent vectors from this trajectory can then be used in our scheme as training data, to teach the episodic trace to a RNN.

We have found that networks can indeed be trained to replicate single-attractor dynamical systems in this manner. Figure 6.1 shows a sequence depicting RNN performance after incremental training runs on a system with a fixed-point attractor. In each run, tangent vector data from the orbit indicated by the red “T” has been fed to a feedforward network for training; the trained network is then transformed to a RNN, and integrated from the plotted set of four initial conditions. Training is not started anew at each run, but rather the network weights are initialized where the previous run left off so that the network builds its internal model of the state space incrementally. As can be seen, the most recent training orbit is the one best approximated by the network. Furthermore, the network’s approximation of previously trained orbits worsens on some runs (an example of catastrophic forgetting). The network seems to have built up a complete and fairly accurate model of the attractor as of the fourth training run, and we suspect this is because it has been shown information from “all four corners” of the state space.

Figure 6.2 shows a similar sequence for incremental training on trajectories from a limit-cycle attractor system. In this case fewer training runs were necessary to achieve good performance on multiple trajectories (including the inner trajectory, which was never trained). This is probably because orbits from this system typically travel through
Figure 6.1: A sequence depicting RNN performance after incremental training runs on a fixed-point attractor system. The most recently trained trajectory is indicated in each subfigure by the red “T”. The sequence proceeds right-to-left, top-to-bottom.

more of the region of interest (all 360 degrees) of the state space.

We note that such incremental training does not always converge, and the final result is sensitive to the order in which training orbits are presented. We have also found that orbits must be sampled to very high resolution to achieve levels of accuracy comparable to those of our global scheme. All the orbits depicted in the preceding figures were 80 nondimensionalized seconds long and sampled to 10000 timesteps. Yet another issue with incremental training is that the network cannot be made too large. This is because, as we have found, an excess of neurons usually leads to the existence of spurious attractors in the state space which are not “trained out” by the limited trajectory data.

Unfortunately, when attempting to train incrementally on multiatractor systems the problem of catastrophic forgetting rears its head more viciously. Previously learned attractors are overwritten, often completely, by more recently learned ones.

We have begun experimenting with a pseudopattern training model (see, e.g., [8], [43]) to overcome this problem. The idea is that, before each training run, a set of
Figure 6.2: A sequence depicting RNN performance after incremental training runs on a limit-cycle attractor system. The most recently trained trajectory is indicated in each subfigure by the red “T”. The sequence proceeds top-to-bottom.
pseudotrajectories is extracted from the network and used to supplement the training data. Pseudotrajectory data are obtained by running the network from a set of randomly selected initial conditions. The pseudotrajectories thereby encode the information stored in the network at the moment of training and present it back to the network. When combined with the new, observed trajectory, these supplemental training data ideally reinforce the network’s existing memory traces while the new trace is added.

In Figure 6.3 we plot trajectories from the familiar two-fixed-point system of (3.6), along with trajectories from a RNN that has been trained incrementally. In the top subfigure, the RNN has been trained only on the four trajectories that converge to the attractor in the left half-plane. Hence, the system has no knowledge of the other attractor, and indeed, trajectories initialized at points in the right half-plane converge to the attractor on the left. The training in this case has not produced a RNN that reproduces the desired trajectories particularly accurately; however, the location of the fixed point has been learned effectively. In the bottom subfigure, the RNN has been further trained on the four original-system trajectories in the right half-plane, as well as a set of fifteen pseudotrajectories (in magenta). Pseudotrajectories were sampled from the network before any training on the right fixed point began, and thus all converge to the left fixed point. As we can see in the lower subfigure, the system has indeed learned the location of both fixed points. Once again, however, the RNN trajectories themselves do not accurately represent the originals.

Clearly, the pseudotrajectory model for incremental training on multiattractor systems is not yet performing optimally. Development of a working incremental training scheme is important for the biological plausibility of the memory system and should be an immediate goal of future work. There are various obstacles and issues at play, which we enumerate here for the benefit of further research.

Numerous training parameters exist and must be fine-tuned for optimal performance. These include the number and resolution of pseudotrajectories and the weight-matrix state at which pseudotrajectories are sampled. There is generally a point during trajectory-based training at which the network has not been exposed to “enough” of a new attractor. Existing attractors are by definition surrounded by basins of attraction, and parts
Figure 6.3: A sequence depicting RNN trajectories after incremental training runs on a system with two fixed points. In the top subfigure only trajectories in the left half-plane have been presented to the RNN. In the bottom subfigure, trajectories from the right half-plane along with fifteen pseudotrajectories (plotted in magenta) have been presented for training.
of these basins must typically be overwritten in order to store a new attractor. At the point we are describing, there is not sufficient new information to overwrite an existing basin. Compounding the problem, sampled pseudotrajectories tend to reinforce the existing basin and it may be that the new training trajectory has no significant effect on the network. Thus, even exposure to multiple trajectories from a new attractor fails to embed the attractor in the network’s state space. It is a challenge to find the balance of pseudotrajectories with new data that is capable of storing a new attractor without completely destroying an old one. One possible solution to this problem is to imbue each attractor with a finite, circumscribed basin that tapers in strength near its perimeter. This would hypothetically make overwriting easier in regions away from existing attractors. In a sense, this issue cuts to the core of learning itself. There is always a delicate balance between old and new memory; sometimes old memories must be overwritten, whether this results from the finite capacity of any storage device, including the brain, or from the fact that two memories could be too similar to coexist. Sometimes old memories are obsolete or incorrect, and ought to be overwritten. Likewise, some memories may be especially important and especially strong, in which case they ought to resist modification.

Another issue we have encountered is that of spurious attractors. If these exist (and, as mentioned, they tend to, with incremental training) then they are often reinforced by randomly sampled pseudotrajectories since there is a reasonable probability that the pseudotrajectories converge on them. This issue can be mitigated by keeping networks small, but at some level of state-space complexity (viz., multiplicity of attractors and their types) a network must grow in order to model that complexity accurately. This requires the recruitment of new neurons. Returning again to biological plausibility, it is also rather unlikely that all of episodic memory is stored in a single monolithic network. There almost surely exist hierarchies of memory networks, perhaps with the trajectories or attractors stored in some acting as the initial conditions for others. Determining an efficient, possibly hierarchical network structure that allows the recruitment of additional neurons will be important for improving our memory model’s capacity, and avoiding spurious attractors.
6.3 The episodic model

Our episodic memory model is built up from two artificial neural networks that act as analogues for two respective brain structures: a recurrent neural network models the associative function of the hippocampus by approximating prescribed dynamical systems, and a deep autoencoder models the reconstructive function of the cortex through inferential and generative probability functions. Nevertheless, the memory model was not designed to be especially accurate physiologically. We have not rigorously reproduced the detailed architecture of these regions, such as their substructures and the components that connect them, as has been done in some similar models (examples being those of Rolls [78] and Vineyard et al. [103]). Our motivation has been an understanding of memory mechanisms from a wider dynamical-systems perspective.

The idea of modelling memory and other cognitive functions with dynamical systems has been extant and studied for some time, as our literature review in Chapter 2 made clear. We have aimed to extend existing theories and models by developing methods for the synthesis of arbitrary dynamical systems and for the approximation of those systems by recurrent neural networks. Dynamical systems and recurrent neural networks make intuitive sense for the modelling of dynamic episodes. However, there is not yet a neuroscientific consensus on what the true nature of memory traces is, and so it remains to be seen if the model can provide us with real insight into the workings of the mind.

We have modelled the two brain functions of associativity and reconstruction rather coarsely. Although the recurrent structure of our hippocampal system was inspired by the equivalent biological architecture, the neuron units in our RNN and autoencoder are modelled abstractly. Biological neurons interact via spike trains rather than the smoothly varying ODE (4.1) that defines the class of additive networks or the probability distribution-sampling of RBMs. The question is how accurate these abstractions are in capturing the underlying dynamics of memory. Experimental evidence suggests that some brain phenomena are dependent on the exact timing of spikes [54, 95, 74], and certainly memory may be one of these. To answer this question we must test the model against real neuroscientific and psychological data. Some possible tests are described in
the next section.

We must also explore the model’s limitations. For instance, how many episodes can be stored, and how varied can these episodes be? The answers to these questions depend on the flexibility of the two neural components. Regarding the autoencoder, separate or relatively large (even at the code layer) networks may be required to encode numerous or visually distinct episodes. On the other hand, highly similar inputs tend to be conflated by encoders, resulting in jumbled reconstructions\(^1\). There is also no direct control over the structure of code space. The cortical encoding of input images arises from autonomous unsupervised learning, the goal of which is to increase the probability of generating those images from their codes. The result is a continuous code space whose structure is often conducive to formation of episode trajectories. However, the lack of control means that very tangled and kinked episode traces are possible. These would pose difficulties for the hippocampal recurrent neural network, which learns more effectively on smoother trajectories. Furthermore, RNN training becomes more difficult as more attractors are embedded because separatrices and distinct dynamical regimes must be formed in the network state space.

In the episodic memory examples of Chapter 5, the synthesis of the memory traces was not autonomous. A curve was fitted through discrete code points (in one case a Fourier series, in the other an ellipse), this curve was made the attracting limit cycle in a 2-dimensional dynamical system according to the methods of Sverdlove [91], and that system was taught to a RNN by supervised backpropagation. The model would be considerably more powerful if the trace-synthesis process were autonomous. This would happen naturally if the model were able to learn from incremental, experienced trajectory data, but as we have seen, learning multiple-attractor systems from incremental trajectories is ineffective as yet.

\(^1\)Although we are all aware that similar jumbling occurs in our own memories.
6.4 Principles of neural organization

Moving forward, and even leaving memory itself aside, we hope that state-space sculpting and our RNN synthesis process may be used to illuminate principles of neural organization. In particular, it should be possible to investigate, in a controlled manner, the link between recurrent weight matrix structure and network dynamics. The goal would be to answer questions like:

- What matrix properties give rise to fixed-point behavior versus limit-cycle behavior versus chaotic behavior?
- How does the matrix structure change when we build a state space that combines multiple elemental behaviors?
- Can we ‘see’ in the matrices $A$, $B$, $C$, or $W$ the different dynamical regimes of a system?
- What weight matrix properties or values correspond to separatrices and the partitioning of state space?
- What matrix operations, if any, can be used to combine existing networks together?

These questions might be investigated by generating a series of sculpted vector fields with incrementally progressing dynamical forms, then instantiating them as RNNs and analyzing the resultant weight matrices. Matrix properties to focus on, for correlating structure with observed dynamics, might include the spectra, the graph Laplacians, and so on.

Once relationships have been established between system dynamics and weight matrix structure (and hence neural connectivity), it will be possible to make testable predictions about neural connectivity in the brain. Programs for mapping the detailed connectivity of the human brain are under way in the United States and Europe, specifically the Brain Activity Map Project and the Human Brain Project. Given the information these programs will yield, it should be possible to analyze biological networks in terms of the same properties mentioned above, and to correlate structure with function in the
simple subnetworks whose functions are well understood. If these correlations match those determined from our synthesized RNNs then our methods could be used to help understand more complicated brain regions.

We have discussed in Chapter 4 the relationship between our synthesized recurrent networks and the well known class of echo state networks. Understanding the relationships between weight matrix structure and system dynamics will also help researchers to generate reservoirs that are tailored to prescribed dynamical regimes. Random reservoirs might be drawn from the distribution of all reservoirs having a given graph Laplacian, for instance.
Chapter 7

Conclusion

7.1 Overview

In this dissertation, we developed and presented a computational model for episodic memory. The model is inspired by the function of the hippocampus and its interactions with the cortex. It comprises two neural components: a hippocampal analogue, manifest as a recurrent neural network, and a cortical analogue, manifest as a deep autoencoder. Together, these components encode incoming experiences as episodic memory traces, store those traces as associative orbits in a dynamical system, and remember stored episodes through dynamical evolution and input reconstruction.

The hippocampus is a brain structure that plays a critical role in the encoding and retrieval of episodic memory. We embodied our model’s hippocampal analogue as a recurrent neural network because the hippocampus is characterized by highly recurrent connectivity. RNNs represent a large class of computational model designed to represent the brain abstractly, and as a consequence of the closed cycles that are the defining feature of their connection topology, RNNs may exhibit self-sustained dynamics. Mathematically, they are dynamical systems.

A dynamical system can be made to exhibit an associative-memory effect by embedding attractors within its state space: Attractor states can be used to represent specific memory patterns; then similar or incomplete patterns that fall within an attractor’s basin will, through dynamical evolution, eventually converge on the original memory
pattern. By harnessing the method of vector field regularization and the inverse solution of Sverdlove, we demonstrated how arbitrary systems of attractors can be synthesized (at least in low-dimensional state spaces). We refer to synthesis by the application of those techniques as “state-space sculpting”.

Given a prescribed dynamical system in the form of a general vector field, $\dot{q} = F(q)$, a model hippocampal network can be constructed to approximate it using an algorithm we developed. First, a three-layer feedforward neural network, given by $F_{NN}(q) = A\sigma(Bq + \theta)$, is trained on input-output data from $\dot{q} = F(q)$. Then the feedforward network is transformed into a recurrent one based on Funahashi and Nakamura’s work [27]. The recurrent network so constructed closely reproduces the original system’s dynamics, as reported in Chapter 4.

Therein we used our RNN-synthesis algorithm to produce neural systems of fixed-point attractors, limit cycles, and strange attractors, both autonomous and forced, and combinations of the same in regularized systems. To capture systems with highly non-linear dynamics or distinct dynamical regimes requires a large number of neurons and successful training becomes more difficult.

The role of the cortex in memory recall is believed to be a reconstructive one. Our model’s cortical analogue was developed to subsume the dual functions of compression for memory storage and decompression for memory reconstruction. It is embodied as a deep autoencoder. The autoencoder’s inferential phase is used to convert high-dimensional sensory inputs into low-dimensional codes, whereby continuous experiences are transformed into trajectories in some low-dimensional space. This is the code space, which also acts as the hippocampal RNN’s state space. The structure of the hippocampal state space is defined through the autoencoder’s dimensionality-reducing transformation. The trajectories encoded from input experiences are the episodic memory traces; they are made into attracting sets in the hippocampal state space by the process outlined above. During recall the hippocampal unit is prompted and converges to the appropriate memory trace, whence the cortical analogue enters its generative phase and performs its reconstruction: it unfolds the low-dimensional trajectory in the hippocampal state space into a flow of high-dimensional, remembered sensory patterns (sights, sounds, etc., though in
our examples we limited ourselves to the first for ease of presentation).

In Chapters 4 and 5 we detailed the mathematics underlying both components of our model and explained their interaction. In Chapter 5, two visual episodes, consisting of abstract animations of a figure running and walking, were memorized by an instance of the model. The animations consisted respectively of 8 and 6 image frames. Each image was in black and white, consisting of $52 \times 40$ pixels. Memory storage involved unsupervised training of an autoencoder, dynamical system synthesis in the code space, and supervised training of a RNN on this dynamical system. The stream of reconstructed images that the model generates closely match the original episodes, constituting a remembrance of things past.

The primary motivation in developing our memory model was the creation of more intelligent, more capable robotic systems. Our system of coupled artificial neural networks was realized in software and proved capable of remembering animated visual episodes from related prompts. That software comprises a memory system with rudimentary human-like abilities that can be utilized in robotic applications.

Our secondary motivation was to contribute to the understanding of human memory dynamics. The model was not intended to mimic microscopic brain function with high fidelity, nor to replicate the entire zoo of brain structures that is involved in memory storage and recall. It is mesoscopic in nature, meant to capture episodic memory from an abstract and simplified dynamical-systems perspective. But dynamical systems are well understood and mathematically rigorous. It is this aspect of our approach that we believe might lead to a greater knowledge of memory in the brain. In particular, the recurrent networks made to reproduce dynamical systems might, by their weight-matrix structure, illuminate and predict principles of biological neural organization.

Future work should extend in several directions. Sverdlove’s solution should be extended to higher dimensions to facilitate the synthesis of more complex dynamical systems. The hippocampal analogue should be extended to incorporate inputs as models of context, and experiments on the replication of stochastic dynamical systems should be carried out. Perhaps most importantly, a method should be developed by which recurrent neural networks can learn systems of multiple attractors from incremental trajectory
data. This would render the model largely autonomous, able to learn from experience without human input. Currently an overseer is required to sculpt the dynamical system that stores desired episodic memory traces. Finally, the model should be used to make testable predictions about neural organization in memory regions of the brain. This requires an understanding of the function of those regions from a dynamical systems perspective.

7.2 Contributions to the field

Our work aimed to fill in several major gaps in current episodic memory research. First: with the exception of Eliasmith’s Neural Engineering Framework, existing techniques for RNN training apply to discrete-time networks, or are incapable of accurately approximating the rich multiattractor systems that must form the basis of episodic memory. The training approach we developed largely parallels that of the NEF, but operates on a restricted class of neural networks that can be shown to obey theoretical bounds on their performance. Second: existing memory models capture some of the associative component of episodic memory but do not directly store episodic traces in the sense of dynamical trajectories, or their capacity to retrieve high-dimensional, continuous-time episodes is limited by the lack of a cortical model for compression and reconstruction. Remembering these internal replays from the past that we see in our mind’s eye was the primary goal for our model.

Our novel contributions to the field are enumerated below.

- A framework, state-space sculpting, was developed to construct arbitrary, prescribed dynamical systems
  - Sverdlove’s solution [91] is used to generate fixed points and limit cycles of prescribed type and location in two dimensions
  - Regularization [85] is used to combine discontinuous vector fields as continuous wholes, and to delineate attractor basins
• The state-space structure of the well known Hopfield network was characterized mathematically within the context of this framework

• An algorithm was developed to synthesize recurrent neural networks that replicate arbitrary dynamical systems
  ◦ The continuous-time additive networks are used
  ◦ A feedforward network is trained on the vector-field representation of the dynamical system
  ◦ The feedforward network is transformed into a recurrent network based on work by Funahashi and Nakamura [27]
  ◦ Synthesized recurrent networks exhibit the prescribed dynamics with high accuracy, particularly attractors and their basins
  ◦ The restricted form of these RNNs facilitated theoretical and dynamical analysis

• A model of episodic memory was developed, capable of remembering animated visual episodes
  ◦ The model combines a recurrent neural network with an autoencoder
  ◦ Episodes are stored as low-dimensional trajectories in an RNN and reconstructed therefrom by an autoencoder
  ◦ Reconstructed episodes closely match the originals both visually and temporally

7.3 Closing thoughts

Memory is a pillar of intelligence. Episodic memory, the memory of autobiographical events, that collection of experiences from times and places gone, through which we travel back and forth, forms the very core of who we are. As individuals we are collectors. The limbs of our cerebral trees reach out, to wrest from the chaos around us some shards of
dream and form, and decide these ones, thus. Flouting the second law of thermodynamics, we consume information, arrange it, and add it to ourselves, just as we do metabolically with matter and energy. Memories have meaning because they change us; they make an impression, with all the mechanical connotations of that word, imprinting themselves on our synapses. They change how we see all that comes after. We hope, through this work, to have revealed some small piece of that immense edifice that upholds the sessions of thought.

. . . when nothing subsists of an old past, after the death of people, after the destruction of things, alone, frailer but more enduring, more immaterial, more persistent, more faithful, smell and taste still remain for a long time, like souls remembering, waiting, hoping, upon the ruins of all the rest, bearing without giving way, on their almost impalpable droplet, the immense edifice of memory.

— Marcel Proust
Bibliography


hessian-free optimization. In Proceedings of the 28th International Conference on

[64] James L McClelland, Bruce L McNaughton, and Randall C O’Reilly. Why there
are complementary learning systems in the hippocampus and neocortex: insights
from the successes and failures of connectionist models of learning and memory.

[65] Bruce L McNaughton and Richard GM Morris. Hippocampal synaptic enhance-
ment and information storage within a distributed memory system. Trends in


[67] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek
Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. Using
recurrent neural networks for slot filling in spoken language understanding. Audio,
Speech, and Language Processing, IEEE/ACM Transactions on, 23(3):530–539,
2015.

[68] Gianluigi Mongillo, Omri Barak, and Misha Tsodyks. Synaptic theory of working

[69] J Nagumo and S Sato. On a response characteristic of a mathematical neuron

[70] Jinichi Nagumo, S Arimoto, and S Yoshizawa. An active pulse transmission line

[71] Shigetoshi Nara, Peter Davis, Masayoshi Kawachi, and H Totsuji. Memory dynamics
in a recurrent neural network with cycle memories embedded by pseudo-inverse
method. In Neural Networks, 1993. IJCNN’93-Nagoya. Proceedings of 1993 Inter-


