To attract or to oscillate: Validating dynamics with behavior

by

Keith T. Murray

B.S. Computation and Cognition & Linguistics and Philosophy, MIT, 2022

Submitted to the Department of Brain and Cognitive Sciences
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN COMPUTATION AND COGNITION

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2024

Authored by: Keith T. Murray
Department of Electrical Engineering and Computer Science
January 19, 2024

Certified by: Nancy A. Lynch
NEC Professor of Software Science and Engineering, Thesis Supervisor

Accepted by: Mehrdad Jazayeri
Associate Professor
Director of Education, Department of Brain and Cognitive Sciences

To attract or to oscillate: Validating dynamics with behavior

by

Keith T. Murray

Submitted to the Department of Brain and Cognitive Sciences
on January 19, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN COMPUTATION AND COGNITION

ABSTRACT

In recent years, the 'computation-through-dynamics' framework has gained traction within the neuroscience community as a means of describing how neurological processes implement behavioral computations. The framework argues that computations in neural systems are best explained through dynamical systems in which behaviorally-relevant variables are represented and manipulated via dynamical phenomena. While a variety of previous works have demonstrated the framework's productivity, there are a number of challenges surrounding its efficacy. In this thesis, we identify and address two challenges concerning the existence of multiple dynamical systems which perform the same computation.

We show that a continuous-time recurrent neural network (CT-RNN) can implement two distinct dynamical systems, termed the "attractive mechanism" and the "oscillatory mechanism", to compute a novel modular arithmetic task inspired by the card game SET. The attractive mechanism computes modular arithmetic through traversing a lattice of fixed-point attractors. The oscillatory mechanism computes modular arithmetic through phase-shifts on a limit cycle. The existence of these two dynamical mechanisms raises two challenges for the 'computation-through-dynamics' framework:

1. How can computationally similar, yet dynamically distinct systems be experimentally identified?

2. What criteria determine the implementation of one dynamical system versus another?

We address these questions by advocating for the use of behavioral phenomena. Through two experiments, we show how our dynamical mechanisms produce distinct psychometric curves when classifying ambiguous stimuli and generalize to unseen stimuli at different rates when trained on partial datasets. We further argue how these behavioral phenomena can serve as ecological criteria in determining the implementation of a mechanism. These results underscore the utility of behavior in the 'computation-through-dynamics' framework.

We conclude this thesis by formulating levels of abstraction for the 'computation-through-dynamics' framework, termed 'levels of neural computation'. Levels of abstraction were critically important in establishing the efficacy of digital computation; therefore, we speculate that the 'levels of neural computation' will further advance the efficacy of the framework. These levels argue for interpreting dynamical systems as implementations for more abstract 'geometric representations and manipulations' that effectively serve as neural algorithms.

Thesis supervisor: Nancy A. Lynch
Title: NEC Professor of Software Science and Engineering

Acknowledgments

I would first like to acknowledge the guidance of my thesis advisor, Prof. Nancy Lynch. I've worked with Prof. Lynch since September of 2020, and many times I've been reminded how fortunate I am to have a research advisor as encouraging and knowledgeable as her. During a meeting in October of 2022, I introduced her to the card game SET and sketched the outline of this thesis. While my ideas were extremely preliminary and I could sense her concern, she provided invaluable feedback and support that allowed this thesis to take its current form.

Next, I would like to acknowledge the mentorship of Brabeeba Wang. In February of 2020, I responded to a research ad that Brabeeba had posted on the MIT SuperUROP website detailing a project involving modeling the retina with differential equations. At the time, I knew very little about neuroscience, but was excited for the opportunity to begin to understand the brain. Over the course of the COVID-19 pandemic, Brabeeba shaped my excitement for neuroscience into an awareness of the challenges of the field and its open questions. He instilled in me a sense of rigour and curiosity for research that I will forever be grateful for.

Comments from the entire Theory of Distributive Systems group were immensely helpful in completing this thesis. Thank you Sabrina Drammis, Noble Harasha, Adithya Balachandran, and Grace Cai.

My family deserves a large portion of credit for maintaining my sanity over the course of this thesis. From weekly calls with my mother about my current progress, to playing SET around the kitchen table during one of my few trips back to Indiana, they are the most supportive family I could have ever hoped for. Mother, Father, Max, Jessica, Samantha, and Destiny, thank you all so much for the immeasurable love and encouragement.

A student's success at MIT is not a function of their intelligence, but a function of the friends they surround themselves with. I'd like to thank my friends at MIT, especially Nikasha Patel, Luke Igel, Julian Liu, Josh Liu, Kenny Cox, Kevin Ly, Jose Lavariega, Anthony Baez, Will Archer, the Break Through Tech for Artificial Intelligence teaching staff, the brothers of Phi Delta Theta, the MIT Men's Heavyweight Crew Team, and the Concourse learning community.

I am obliged to acknowledge the assistance of ChatGPT in editing portions of this thesis and implementing Python and Julia code.

Finally, I would like to acknowledge all of my professors, instructors, and mentors at MIT, especially Prof. Suzanne Flynn, Dr. Kimberle Koile, Prof. Mark Harnett, Dr. Lukas Fischer, Prof. Robert Yang, Prof. Nir Shavit, Prof. Ila Fiete, and Prof. Bob Berwick.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Despite decades of research, the neurological underpinnings of cognition remain a mystery. In the wake of this mystery, a variety of frameworks have emerged that attempt to describe how cognitive behaviors manifest from neurological processes. In this thesis, we focus on the 'computation-through-dynamics' framework where we identify and address challenges concerning the existence of multiple dynamical systems which perform the same computation.

1.1 Motivation

Explicitly formulated in [1], the 'computation-through-dynamics' framework seeks to use the language and techniques of dynamical systems theory to uncover how the activity of neuronal populations can implement computations necessary for behavior. For a review of dynamical systems theory, refer to [2]. At the heart of the framework lies a simple argument [3], [4]:

> The activity of neuronal populations is necessarily variable in time and responsible for behavior; therefore, the computations underlying behavior must have corresponding dynamical systems.

While this argument is in principle correct, its correctness does not guarantee its efficacy towards explaining cognition. As argued in [5], a neuroscientist using techniques standard in the 'computation-through-dynamics' framework would fail to understand the computations performed by a microprocessor because the framework fails to meaningfully capture the hierarchy of information processing implemented in the processor; therefore, it could be the case that the framework is insufficient to describe the computations responsible for cognition in neural systems. To establish its efficacy in explaining how cognition arises from neural activity, the framework needs to address the following questions:

1. Are neural dynamics low-dimensional?
2. How do neural substrates implement dynamical systems?
3. Do dynamical systems explain causal relationships among task variables in neural systems?
4. How could separate dynamical systems be combined to produce complex behaviors?

5. How could dynamical systems perform symbolic operations?

6. How can computationally similar, yet dynamically distinct systems be experimentally identified?

7. What criteria determine the implementation of one dynamical system versus another?

Previous works have addressed questions 1-6. In this thesis, we readdress question 6 and propose question 7. To our knowledge, question 7 is novel and has yet to be addressed.

We argue that questions 6 and 7 are of particular importance for the 'computation-through-dynamics' framework because they challenge the existence of a function that can map computations to dynamical systems. Consider X to be the set of all computational operations x and Y to be the set of all dynamical systems y. The 'computation-through-dynamics' framework would benefit tremendously from the existence of a function f that can map computations to dynamical systems that implement them $f : X \to Y$. With such a function, an experimental neuroscientist investigating the dynamical system associated with a behavior would only have to specify the computations involved in the behavior, x_0, and then apply the function to find the dynamical system $f(x_0) = y_0$; thus, eliminating the need to collect data from a neural system performing the behavior. However, question 6 presupposes that this function cannot exist because one computation might correspond to multiple distinct dynamical systems. If it is the case that one computation corresponds to multiple dynamical systems, the 'computation-through-dynamics' framework need not be discarded. We must understand how computationally-similar dynamical systems can be experimentally disambiguated, question 6, and why a neural system might implement one dynamical system versus others, question 7.

1.2 Overview

To address question 6, we first identified a task that could elicit two distinct dynamical systems from task-optimized continuous-time recurrent neural networks (CT-RNN). This task was a modular arithmetic task based on the card game SET [6]. We found CT-RNNs trained on our modular arithmetic task learned two distinct *dynamical mechanisms*: the *attractive mechanism*, where the network's dynamics are characterized by fixed-point attractors, and the *oscillatory mechanism*, where the network's dynamics are characterized by a limit cycle. Having identified a task that elicits distinct dynamical mechanisms and characterized those mechanisms, we theorized how these mechanisms could be experimentally identified in a biological agent performing our modular arithmetic task.

As mentioned earlier, previous works have identified and addressed question 6. These previous works employed novel neural analysis techniques to infer the dynamical and temporal structures responsible for computational processes [7], [8]. These methods relied on precisely measured neurological data to identify dynamical systems, but in this thesis, we opt for using behavioral data to disambiguate our computationally-similar dynamical mechanisms. We show that there are rich behavioral phenomena, namely *psychometric signatures* and *grokking curves*, that can act as a window into the dynamical mechanisms of a computational process, and thus could be used to address question 6. *Psychometric signatures*

are functions that describe the types of mistakes dynamical mechanisms perform when computing modular arithmetic for ambiguous stimuli; we show that the attractive mechanism exhibits more false positive errors than the oscillatory mechanism. *Grokking curves* are curves that describe the probability of a dynamical mechanism generalizing to unseen data; we show that the oscillatory mechanism generalizes more frequently than the attractive mechanism when trained on relatively few modular arithmetic examples.

After having addressed question 6, we address question 7 by speculating about the criteria that might influence the implementation of the attractive or oscillatory mechanism to compute modular arithmetic. Instead of using behavioral phenomena to identify dynamical mechanisms, like in question 6, we used behavioral phenomena and ecological criteria to explain the implementation of one particular dynamical mechanism over others.

To conclude this thesis, we identify the need for a method of abstraction in the 'computation-through-dynamics' framework and subsequently propose our own inspired by David Marr's levels of analysis [9] and levels of abstraction in digital computation [10]. Notably, our proposed levels of abstraction, termed "levels of neural computation", argue that 'dynamical systems' implement abstractions specified at a 'geometric representation and manipulation' level and argue that algorithms in neural systems are most effectively formulated as geometric abstractions.

1.3 Structure

This thesis is structured in the following manner:

- **Chapter** 1: A summary of the challenges to the 'computation-through-dynamics' framework and an outline of this thesis.
- **Chapter** 2: A background of influential works in the 'computation-through-dynamics' framework [11]–[14] and an explanation of how these works address questions 1-5.
- **Chapter** 3: A description of our modular arithmetic task and its relationship to the card game SET [6].
- **Chapter** 4: A description of the *attractive* and *oscillatory mechanisms* and their relationship to finite-state machines (FSMs). Parts of this chapter were taken from our previous work in [15].
- **Chapter** 5: A summary of our experiments using behavioral phenomena to distinguish the *dynamical mechanisms* described in **Chapter** 4. This chapter addresses questions 6 and 7.
- **Chapter** 6: A summary of this thesis, our 'levels of neural computation', and future directions.

Chapter 2

Computation-Through-Dynamics

As described in **Chapter** 1 and in [1], the 'computation-through-dynamics' framework seeks to use dynamical systems theory to explain how neural circuits perform behaviorally-relevant computations. In this chapter, we review four previous works[1] that exemplify the language and results of this framework. In addition, we highlight how these works address a variety of challenges to the 'computation-through-dynamics' framework, specifically questions 1-5 in **Section** 1.1.

In short, the four works reviewed in this chapter detail how simple tasks give rise to low-dimensional, interpretable dynamics in recurrent neural networks (RNNs) and how these dynamics perform computational operations. In **Section** 2.1, we review [11], the *3-bit flip-flop* task and its corresponding variable assignment := computation, and *sine wave generator* task. In **Section** 2.2, we review [12], the *context-dependent decision making* task, and the concept of *latent circuits*. In **Section** 2.3, we review [13], the *memory guided saccade* task and its corresponding sine and cosine computations, and the concept of *dynamical mechanisms*. In **Section** 2.4, we review [14], the *transitive inference* task, and the greater than $>$ and less than $<$ computations. These works provide a host of examples of how RNNs learn to compute with dynamical systems in a manner exemplary of results typical of the 'computation-through-dynamics' framework.

2.1 Opening the black box

As stated in **Section** 1.1, the first challenge facing the 'computation-through-dynamics' framework concerns the dimensionality of neural dynamics:

1. Are neural dynamics low-dimensional?

Given that our geometric intuitions are restricted to three dimensions, the insights provided by the 'computation-through-dynamics' framework would be of little utility if neural dynamics were high-dimensional. For example, in a neural circuit of 100 neurons, a 100-dimensional dynamical system might provide a full account of the computations performed, but such an

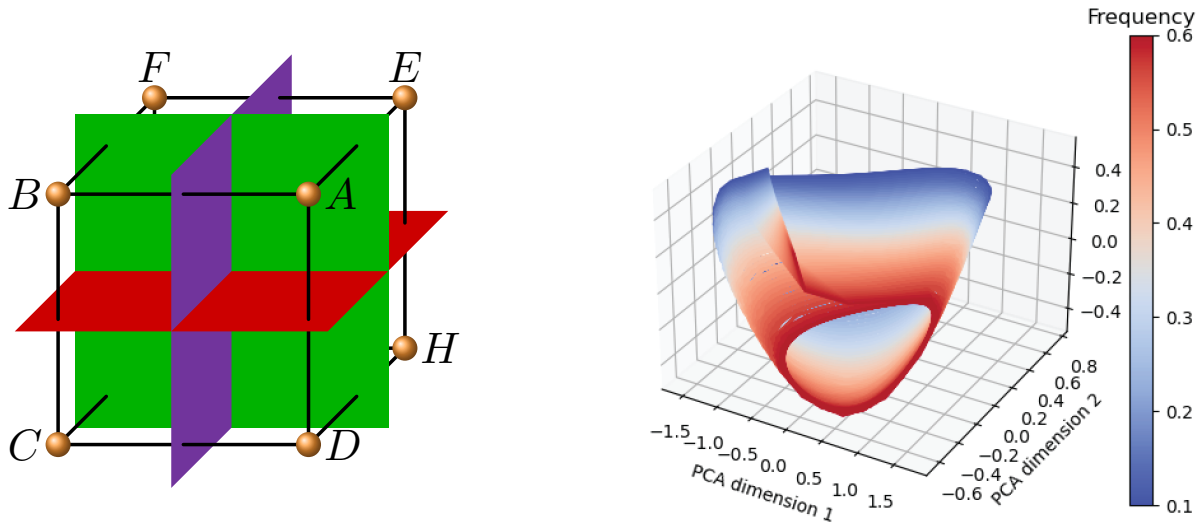
[1] We include figures to aid in explaining the tasks and dynamics described in these works. These figures were not directly taken from these works but reinterpreted by us to avoid any potential copyright issues.

account would not be interpretable, and thus would not be useful to a researcher interested in comprehending the computations performed by that circuit.

In "Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks", Sussillo and Barak [11] provide examples of how neural systems can exhibit low-dimensional, interpretable dynamics. Sussillo and Barak created these examples through training RNNs on a series of three tasks:

- 3-bit flip-flop
- Sine wave generator
- 2-point moving average

In this section, we review the tasks and associated low-dimensional dynamics for the 3-bit flip-flop and sine wave generator tasks (**Sec.** 2.1.1 and **Sec.** 2.1.2, respectively).



(a) 3-bit flip-flop

(b) Sine wave generator

Figure 2.1: RNN dynamics for the Sussillo and Barak tasks

2.1.1 3-bit flip-flop task

In this task, an RNN is trained to emulate a 3-bit shift register. At any time t_i, the network can receive an impulse of -1 or $+1$ in one of three input channels and for time $t > t_i$, the network must continually output that impulse in its respective output channel until another impulse for that channel arrives.

To solve this task, an RNN learns a 'cube' of fixed-point attractors where each corner is an attractor and the edges of the 'cube' are paths that the activity of the network can take to transition between bits. **Figure** 2.1a depicts the arrangement of attractors with three classification planes denoting how bit-flips are computed. For example, consider attractor **A** corresponds to a bit configuration of $(+1, +1, +1)$. If an impulse of -1 arrives in the second channel, the activity of the network traverses the $\overline{AB}$ path, crosses the purple plane

corresponding to the bit classification for the second bit, and falls into the basin of attraction for attractor B at coordinate $(+1, -1, +1)$. The geometry of three orthogonal classification planes and a cube configuration of attractors allows for one bit to 'flip' without affecting the status of the other bits.

In essence, the computation elicited by the 3-bit flip-flop task is that of variable assignment (:=). Digital computations and algorithms typically involve variables capable of arbitrary and flexible value assignment, and the RNN dynamics learned from the flip-flop task exemplify how an RNN could assign and remember multiple values for multiple variables without interference.

2.1.2 Sine wave generator task

While the 3-bit flip-flop task emulates a shift register, the sine wave generator task emulates a signal generator. For this task, the network receives a constant amplitude input and is tasked with outputting a sine wave with a frequency corresponding to the amplitude of the input.

To solve this task, an RNN learns a 'cone' of limit cycles where the amplitude of the input changes the height coordinate of the RNN's activity on the cone. **Figure** 2.1b illustrates how higher frequencies correspond to the network's activity being in a narrower region of the cone and lower frequencies correspond to wider regions. It is worth noting that not all frequencies of the cone are active at the same time. Inputs activate the limit cycle on the cone corresponding to the output frequency and deactivate other limit cycles.

In essence, the computation elicited by the sine wave generator task is that of pattern generation. Pattern generators are implicated in various neurophysiological phenomena [16] and the RNN dynamics learned from this task demonstrate how neural systems could learn to produce a continuum of frequencies needed for pattern generation.

2.2 Latent circuit inference

The second challenge facing the computation-through-dynamics' framework concerns the relationship between a neural circuit's dynamics and connectivity (**Sec.** 1.1):

2. How do neural dynamics arise from the connectivity among neurons?

Mirroring how high-dimensional RNNs learn low-dimensional dynamics [11], "Latent circuit inference from heterogeneous neural responses during cognitive tasks" by Langdon and Engel [12] investigates whether high-dimensional RNNs learn low-dimensional circuits termed *latent circuits*. If present, these latent circuits could provide clues as to how the connectivity and dynamics of an RNN are related.

2.2.1 Context-dependent decision making task

The task used by Langdon and Engel was the context-dependent decision making task proposed in [17]. The task involves integrating information from two conceptually different sources and making a decision based on the integrated information. The task is based on

the moving-dots task where macaques are tasked with looking at two mixed populations of moving dots and deciding which direction contains the majority of moving dots. The context-dependent decision-making task further expands on the moving-dots task by coloring the moving dots. Now, there are 4 populations of dots:

1. Dots moving left
2. Dots moving right
3. Red dots
4. Green dots

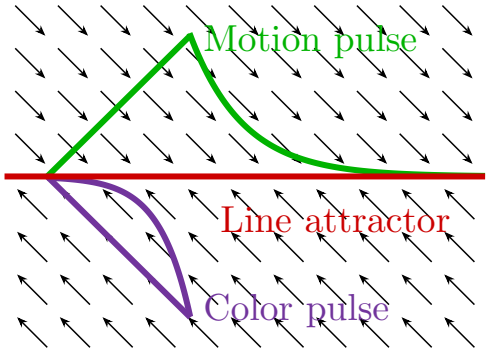
While between the direction categories and between the color categories, the populations are mutually exclusive (e.g. no dot can be both red and green or move left and right); the direction and color categories do mix (e.g. a dot can be red and move left or be red and move right). A macaque performing this task is first given a *context*, motion or color, and then integrates information depending on the context to decide which category is most prevalent in the populations of dots. For example, if a macaque is given the motion context, then it must selectively pay attention to the direction, left or right, in which the majority of dots are moving.

The authors of [17] found that macaques and RNNs performing the context-dependent decision making task learned a simple dynamical system characterized by a line attractor, an orthogonal encoding of motion and color input, and a vector field modulated by the context. **Figure** 2.2a illustrates this dynamical system with an example of the dynamics integrating motion information and suppressing color information. Motion information is integrated because the surrounding vector field is orthogonal to the motion encoding vector, allowing for the activity of the network to decay to the line attractor at a position further than before the motion information was presented. Color information is suppressed because the surrounding vector field is anti-parallel to the color encoding vector, allowing for the activity of the network to decay to the line attractor at its original position.

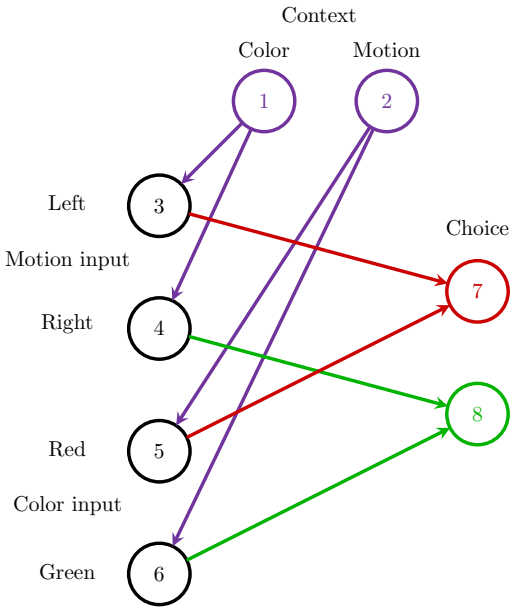
The discovery of this dynamical system was unexpected, as it was previously believed that context-irrelevant information would be suppressed prior to reaching the prefrontal cortex (PFC), the region where macaque neural activity was recorded, as noted in [17]. Instead, the dynamical system indicated that context-irrelevant and context-relevant information are both passed to the PFC and it is the dynamics of the PFC that suppress irrelevant information.

2.2.2 Limitations of dimensionality reduction

In [12], Langdon and Engel trained RNNs to perform the same context-dependent decision making task as in [17], but further analyzed these RNNs with their *latent circuit inference* method. This method allowed the authors to *distill* low-dimensional *latent circuits* from high-dimensional trained RNNs; these circuits indicated that context-irrelevant information was suppressed in a manner inconsistent with the dynamical system identified in [17]. **Figure** 2.2b illustrates the latent circuit identified by Langdon and Engel. Here, there are context specific neurons that inhibit information coming from the context-irrelevant input



(a) Context-dependent dynamical system



(b) Latent circuit

Figure 2.2: Content-dependent decision making mechanisms

neurons. A circuit reflective of the dynamical system in [17] would have connectivity that initially allows for the inflow of context-irrelevant information from input neurons with suppression occurring in some intermediate mechanism.

After identifying the existence of a latent circuit that conflicts with the dynamical system in [17], Langdon and Engel were tasked with identifying how these two mechanisms could co-exist [12]. They concluded that the dynamical system emerged due to the dimensionality reduction technique used in [17]: principal component analysis (PCA). PCA works by extracting dimensions in high-dimensional data that exhibit high variability; however, the variability in these dimensions are identified based on correlational relationships among task variables, not causal relationships [12]. Langdon and Engel argue that the dynamical system identified in [17] describes only how task variables, like motion input and color context, are correlated to each other, while the latent circuit is a causal mechanism that describes how task variables are causally related. They support this argument via targeted perturbations of RNNs that result in psychometric perturbations: perturbing the dimensions of activity identified by PCA do not result in behavioral differences; conversely, perturbing the dimensions of activity identified by the latent circuit inference method do result in behavioral differences.

Langdon and Engel's work highlights the third challenge for the 'computation-through-dynamics' framework (**Sec.** 1.1):

3. Do dynamical systems explain causal relationships among task variables in neural systems?

This question is of particular importance due to previous work that has identified the existence of *phantom oscillations* in PCA trajectories of otherwise non-oscillatory neural sys-

tems [18]. In summary, the utility of the 'computation-through-dynamics' framework can only be realized if it is able to explain how neurological processes *cause* cognitive computations; dynamical systems that merely explain how neurological processes are *correlated* with cognitive computations will ultimately fail to be meaningful.

2.3 Dynamical motifs

A key feature of conventional, digital computation is its ability to combine and reuse computational primitives. For example, a programmer does not need to redefine or reimplement the modulo operator % every time they write a unique program. The algorithm underlying the modulo operator % is already predefined and preimplemented allowing for efficient and convenient programming and computation. By translating this programming feature to the 'computation-through-dynamics' framework, we can highlight the importance of the fourth question from **Section** 1.1:

4. How could separate dynamical systems be combined to produce complex behaviors?

In "Flexible multitask computation in recurrent networks utilizes shared dynamical motifs", Driscoll, Shenoy, and Sussillo [13] sought to understand how neural systems could implement *dynamical motifs* that could be flexibly recombined to perform behaviorally-relevant computations. The term "dynamical motifs" refers to dynamical elements, such as fixed-point attractors, rotations, and decision boundaries, that were reused across a variety of different tasks [13]. Driscoll, Shenoy, and Sussillo investigated flexible computation and dynamical motifs using the *memory guided saccade task*.

2.3.1 Memory guided saccade task

The memory guided saccade task, used extensively in previous non-human primate literature [19], can be described as occurring over the course of four phases:

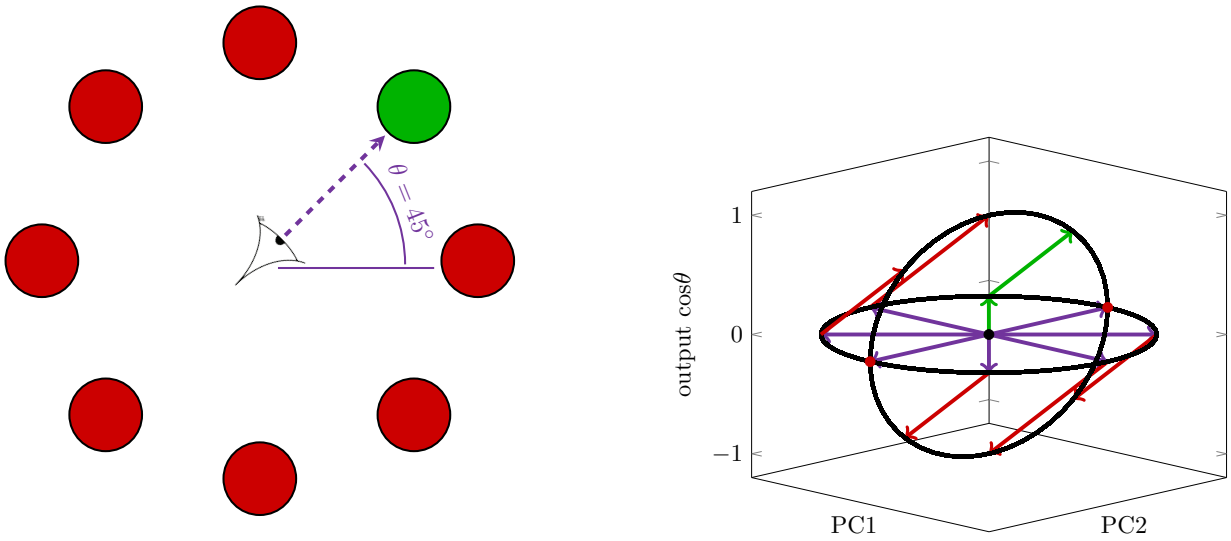
1. **Fixation**: The agent must fixate their gaze at a point (FP) in the center of the screen. The eye in **Figure** 2.3a denotes FP.

2. **Stimulus**: A stimulus target (T) flashes in the peripheral visual field of the agent. The green circle denotes T (**Fig.** 2.3a). The agent must continue to fixate on FP as T appears.

3. **Memory**: As the agent continues to fixate on FP, T disappears. The agent must remember the position of T during this phase.

4. **Response**: FP now disappears and the agent must saccade to fixate on the remembered location (R) of T. The distance between R, the agent's memory of T's location, and T, the actual target location, is the error (E). **Figure** 2.3a denotes the agent in the **Response** phase fixating on a T at 45°.

Although the memory guided saccade task is simple, Driscoll, Shenoy, and Sussillo leveraged its simplicity to parameterize the task and create a variety of saccade tasks with overlapping computations. For example, the authors created another task called the *MemoryAnti* similar to the memory guided saccade task with the exception that the agent must fixate on a target point ($\tilde{\text{T}}$) 180° opposite to the stimulus target (T) presented during the **Stimulus** phase [13]. The authors found that an RNN trained on both tasks utilized the same attractors and decision boundary; thus, providing evidence for dynamical motifs being reused in multiple tasks sharing similar computations.



(a) Memory guided saccade task

(b) Associated dynamical mechanism

Figure 2.3: The memory-saccade task and associated dynamical mechanism

2.3.2 Flexible sinusoidal computations

Figure 2.3b provides a visual example of how dynamics could be reused for different computations. Let's say that the goal of the computation is to compute the cosine of the angle of the target point (θ) in **Figure** 2.3a. The computation of the RNN throughout the task can be described as follows:

1. **Fixation**: The activity of the network begins in the central fixed-point attractor (the black point in **Fig.** 2.3b).

2. **Stimulus**: The target stimulus pushes the network's activity to a ring of fixed-point attractors, referred to as a *ring attractor*, where the angle on the activity on the ring attractor corresponds to the angle of the target (θ). The purple arrows in **Figure** 2.3b denote the target stimulus.

3. **Memory**: As the target stimulus disappears, the attractive dynamics of the ring attractor maintain the memory of T's position.

4. **Response**: When the FP disappears, the ring attractor rotates 90° out of its original plane. In **Figure** 2.3b, the ring attractor rotates out of the principal component 1 (PC1) and PC2 plane to occupy the 'output $\cos\theta$' and PC2 plane. Now, the position of the network's activity on the 'output $\cos\theta$' axis is equivalent to the cosine of the angle of the target point (θ), which was to be shown.

Conveniently, we can now compute either the cosine or sine of θ. To compute $\sin\theta$, we only need to change the plane that the ring attractor rotates to: rotating to occupy the output-PC1 plane computes $\sin\theta$ and rotating to occupy the output-PC2 plane computes $\cos\theta$. The ring attractor and output axis dynamical motifs remain the same while the rotation dynamical motif changes.

Like most tasks and dynamical systems in this chapter, the memory guided saccade task and associated dynamical motifs are simple; however, they provide a framework and language to theorize about how neural systems could flexibly compute a variety of tasks. Like a programmer stringing together functions in a program, cortical areas could compile together dynamical motifs to compute complex behaviors without learning entirely new dynamical systems.

2.4 Transitive inference

Perhaps the foremost question at the intersection of neuroscience and cognitive science is the following:

> How do neural systems perform symbolic operations?

This question has seen many answers [20], and yet, it still remains an open question [21]. In the context of the 'computation-through-dynamics' framework, the question transforms into the following question from **Section** 1.1 [22], [23]:

5. How could dynamical systems perform symbolic operations?

In "Neural dynamics and geometry for transitive inference", Kay et al. provided a first step towards answering this question [14].

2.4.1 Symbolic tasks

Kay et al. investigated how neural systems perform symbolic operations in the context of transitive inference. Transitive inference is a symbolic computation that consists of determining an abstract relation between novel combinations of symbols [14]. For example, if A>B and B>C, even though A and C are not explicitly compared, we can deduce that A>C. In the context of RNNs, the authors were curious if ANNs could exhibit transitive inference and if their transitive inference computations had associated dynamical phenomena.

Kay et al.'s transitive inference task consisted of presenting a network with two symbols at disparate times and tasking the network to indicate if the first symbol was "greater than" (>) the second symbol. The task consisted of the following elements:

- **Alphabet**: A set of symbols {A,B,C,D,E,F,G} with the associated alphabetical ordering.
- **Training set**: The set of relations and labels used for training the network.
  - **Valid relations**: {(A,B),(B,C),(C,D),(D,E),(E,F),(F,G)} are associated with a label of $+1$.
  - **Invalid relations**: {(B,A),(C,B),(D,C),(E,D),(F,E),(G,F)} are associated with a label of -1.
  - Note that only pairs of symbols ordered adjacently are in the **Training set**.
- **Testing set**: All the relations not included in the **Training set**.

An example trial over the course of time t might look like the following:

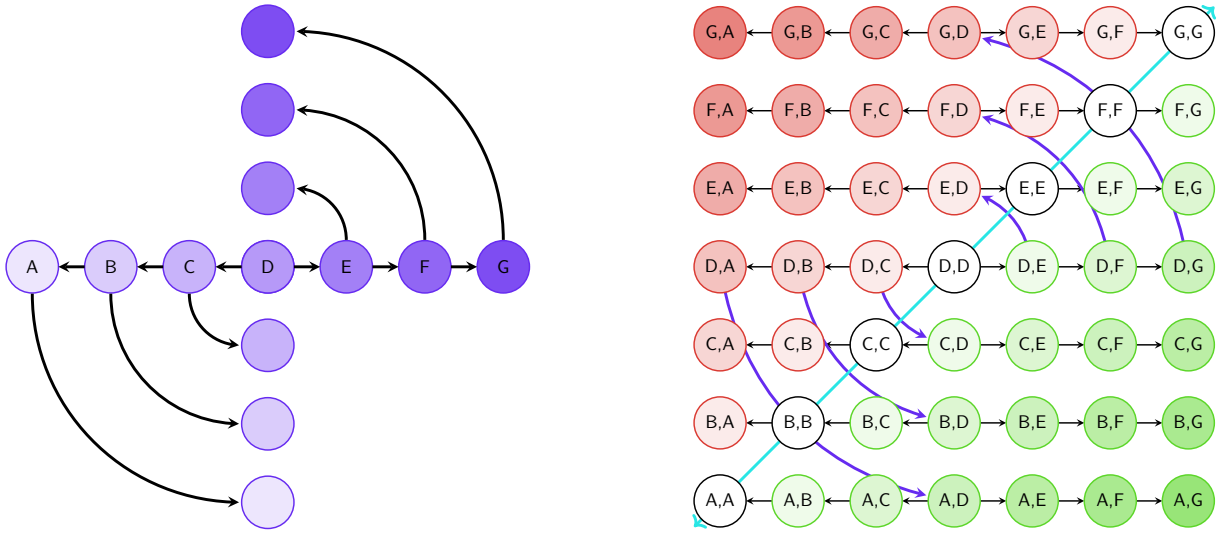
1. For $t_0 \leq t < t_1$: No inputs are given.
2. At $t = t_1$: The first symbol of F is given as input to the network.
3. For $t_1 < t < t_2$: No inputs are given and the network must remember the first symbol.
4. At $t = t_2$: The second symbol of E is given as input.
5. For $t > t_2$: The network must output -1 because F$\not>$E.

The authors found that RNNs exhibited transitive inference for all relations when trained only on the **Training set** containing adjacently ordered relations. Furthermore, the dynamical system underlying transitive inference computations in RNNs was particularly striking (**Fig.** 2.4). Notably, RNNs learned to encode symbols in the **Alphabet** as vectors with the same direction but differing magnitudes. In **Figure** 2.4a, the nodes labeled with symbols indicate the vector encodings. All symbol encodings had a direction of $0°$ with magnitudes ranging from $+1$ for G to -1 for A.

2.4.2 Subtractive dynamical systems

To illustrate how this dynamical system computes transitive inference, let's revisit the example trial:

1. For $t_0 \leq t < t_1$: The network's activity is at rest at the origin. The origin is denoted by the node D, in **Figure** 2.4a, where the D symbol has a vector encoding of 0 magnitude.
2. At $t = t_1$: The first symbol, F, pushes the network's activity to node F in **Figure** 2.4a.
3. For $t_1 < t < t_2$: The network's activity follows a quarter-circular trajectory around the origin to rotate its position by $90°$.
4. At $t = t_2$: The second symbol, E, pushes the network's activity to node F,E in **Figure** 2.4b.



(a) First input followed by oscillation

(b) Second input allows for classification

Figure 2.4: The transitive inference dynamical mechanism

5. For $t > t_2$: The RNN outputs -1, corresponding to the $\not>$ relationship between symbols, because the activity is above the teal classification boundary in **Figure** 2.4b.

    - Note that the classification computation is essentially $\text{sgn}(\begin{bmatrix}1 & -1\end{bmatrix}\vec{x})$ where $\begin{bmatrix}1 & -1\end{bmatrix}^T$ represents the tangent vector of the decision boundary and $\vec{x}$ is position of the network's activity in **Figure** 2.4b.

In essence, this dynamical system computes the greater than operation ($>$) via subtracting vector-encoded symbols with oscillatory dynamics. To compute the less than operation ($<$), the network can rotate its activity by $-90°$ during the $t_1 < t < t_2$ period, thus subtracting symbols in the opposite direction, or change the tangent vector of the decision boundary to $\begin{bmatrix}-1 & 1\end{bmatrix}^T$.

While the greater than ($>$) and less than ($<$) operations are not a full account of all symbolic operations, Kay et al. provided examples of how the 'computation-through-dynamics' framework can explain symbolic computations in neural systems. Future work at the intersection of computation-through-dynamics and symbolic computation should seek to test more symbolic operations and create theories concerning how these symbolic operations might be compiled together to perform more complex symbolic computations and algorithms.

Chapter 3

SET and Modular Arithmetic

In this chapter, we provide an explanation of our modular arithmetic task and its connections to the card game SET [6]. Incidentally, our original purpose for designing our modular arithmetic task was not to elicit multiple dynamical systems from trained RNNs, as described in **Chapter** 1. Instead, we initially intended to design a task entirely representative of SET and compare task-optimized RNNs to behavioral data collected from humans playing SET. However, the modular arithmetic computations at the core of the card game proved to be a fruitful task relevant for addressing challenges to the 'computation-through-dynamics' framework; therefore, we chose to solely focus on a modular arithmetic task in this thesis.

In light of our shifting aims during the course of our investigations, we will describe our modular arithmetic task by first explaining the rules of SET (**Sec.** 3.1), then detailing how these rules can be formulated with modular arithmetic (**Sec.** 3.2), and conclude by explaining the specific details of our modular arithmetic task (**Sec.** 3.3) and our trained continuous-time recurrent neural networks (CT-RNNs) (**Sec.** 3.4).

3.1 The card game SET

SET is a card game where players race to identify *sets*[1] of three cards out of a field of 12 cards. Each card has four attributes: shape, color, number, and shading (**Fig.** 3.1). Each of these attributes has three possible values. A valid *set* is found when for each attribute, all the values pertaining to the attribute across all three cards are the *same* or *different* (**Fig.** 3.2). Refer to **Table** 3.1 for guidance on the terminology we employ when referring to SET.

SET primarily involves two distinct cognitive abilities: pattern recognition and visual search. While visual search has engendered a sizeable psychology and cognitive science literature [24], this thesis focuses on modeling the neural dynamics associated with pattern recognition in SET. Future work could build off of our SET-related pattern recognition insights to create a comprehensive model of how visual search and pattern recognition abilities are jointly employed in SET (**Sec.** 6.3.4).

[1]We refer to the word "set" in two distinct ways: "SET" refers to the card game and "*set*" refers to groupings of cards in SET.

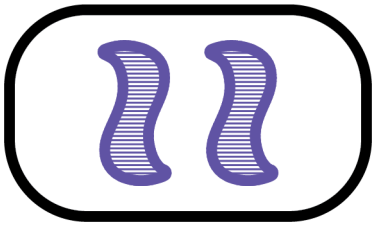
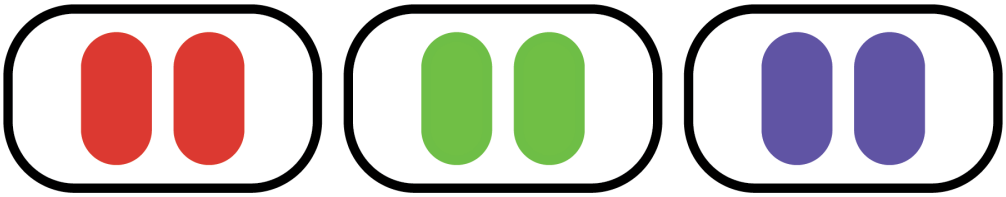
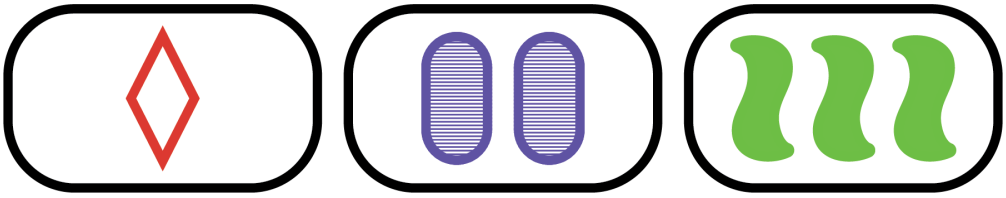


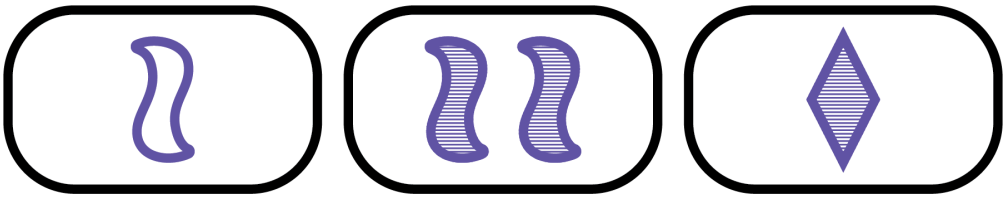
Figure 3.1: An example of a card in SET



(a) Valid *set*



(b) Valid *set*



(c) Invalid *set*

Figure 3.2: Three example *sets*

Table 3.1: Explanation of SET terminology

Term	**Explanation**
SET	The name of the card game.
Set	A group of three cards.
Field	A matrix of 12 cards where players search for *sets*.
Card	A SET card has four attributes, each with 1 value.
Attribute	Generic property of a card: shape, color, shading, or number
Value	Specific attribute of a card.
Shape	Figures on a card: oval, squiggle, or diamond
Color	Color of a shape: red, green, or purple
Shading	Fill of a shape: solid, striped, or empty
Number	Number of shapes on a card: one, two, or three

3.1.1 Pattern recognition

SET engages pattern recognition abilities by prompting the player to classify if three cards constitute a valid or invalid *set*. The *rule of SET* is employed to classify whether a *set* is valid or invalid:

> *Rule of SET*: For each attribute, the values pertaining to that attribute across all three cards must be the *same* or *different*.

The *rule of SET* can also be expressed using first-order logic:

$$\forall a \in A,\ [v(a,c_1) = v(a,c_2) = v(a,c_3)]\ \vee \\ [v(a,c_1) \neq v(a,c_2) \wedge v(a,c_1) \neq v(a,c_3) \wedge v(a,c_2) \neq v(a,c_3)] \quad (3.1)$$

Here, the variable a denotes a single attribute in the set of all attributes, A. c_1, c_2, c_3 denotes each of the three cards in a *set* and the function $v(a, c_i)$ denotes the value of attribute a in card c_i. **Figure** 3.2 provides examples of groups of cards which satisfy and violate the *rule of SET*.

3.1.2 Visual search

SET engages visual search abilities by prompting the player to search through a field of 12 cards to identify a valid *set* (**Fig.** 3.3). While a valid *set* can be in the visual field of the player, identifying that *set* is not often immediate due to presence of 220 potential *sets*, the vast majority of which are invalid.

$$\binom{12}{3} = \frac{12!}{3! \times (12-3)!} = 220$$

It is impractical for players to search through all 220 potential *sets* to find a single valid *set*; therefore, it is likely that players employ search algorithms to efficiently traverse the field of cards. Formulating and validating the potential search algorithms and strategies for SET is beyond the scope of this thesis, but is an area of future research (**Sec.** 6.3.4).

3.2 Modular arithmetic

In **Section** 3.1.1, we formulated the *rule of SET* in natural language and first-order logic. Here, we will reformulate the *rule of SET* using modular arithmetic. Our formulation was entirely motivated by [6]. We further note how our modular arithmetic formulation echos modular arithmetic tasks in the *grokking* literature [25].

3.2.1 Modular arithmetic in SET

First, we integer encode every value across all attributes in **Table** 3.2. The possible integers are $\{0, 1, 2\}$ and the specific integer encoding is arbitrary. Now, we can represent individual

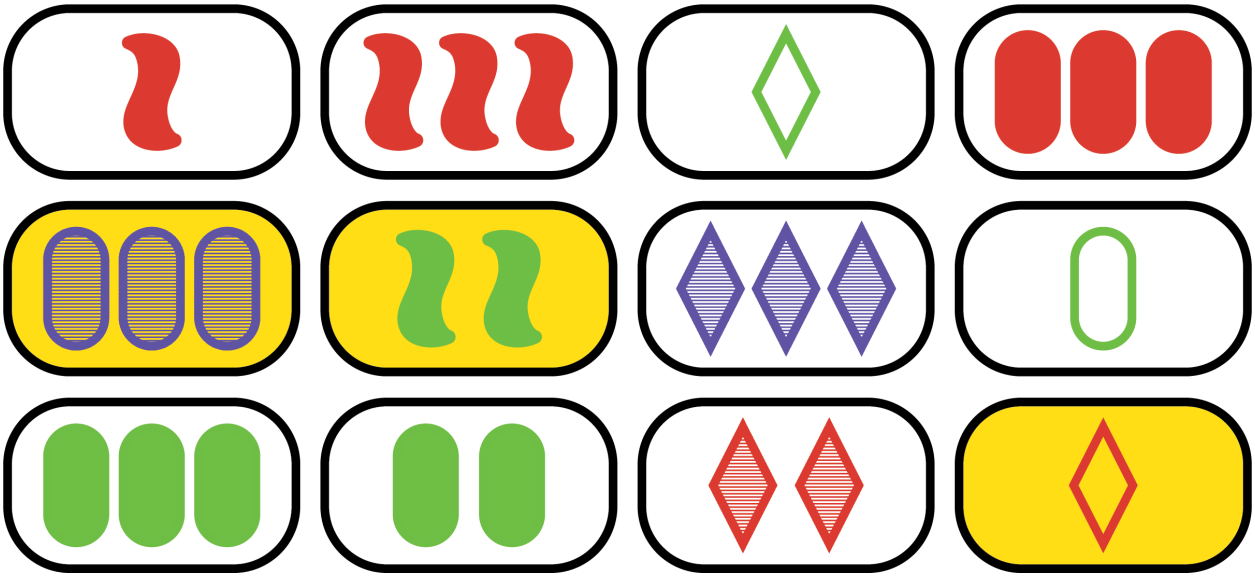


Figure 3.3: An example field of cards with a valid *set* highlighted

Table 3.2: Integer encoding of SET values

Attribute	**Values**	**Encoding**
Shape	oval, squiggle, diamond	0,1,2
Color	red, green, purple	0,1,2
Shading	solid, striped, empty	0,1,2
Number	one, two, three	0,1,2

cards through a vector of their integer encoded values. Here is the vector representation of **Figure** 3.2a:

$$\begin{bmatrix}0\\0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\\1\end{bmatrix}, \begin{bmatrix}0\\2\\0\\1\end{bmatrix}$$

To evaluate if these cards constitute a valid *set*, we add them:

$$\begin{bmatrix}0\\0\\0\\1\end{bmatrix} + \begin{bmatrix}0\\1\\0\\1\end{bmatrix} + \begin{bmatrix}0\\2\\0\\1\end{bmatrix} = \begin{bmatrix}0\\3\\0\\3\end{bmatrix}$$

Notice how each the value of each index of the resulting summed vector is either 0 or 3. The value 6 is also possible. While we could describe the summed vector as must having values of $\{0, 3, 6\}$ to constitute a valid *set*, a more elegant formulation would involve computing the

residue of the summed vector (mod 3):

$$\begin{bmatrix}0\\3\\0\\3\end{bmatrix} \equiv \begin{bmatrix}0\\0\\0\\0\end{bmatrix} \pmod 3$$

Using modular arithmetic, we can simply ask if the summed vector is congruent to the zero vector (mod 3). Here is an example of this formulation using **Figure** 3.2c:

$$\begin{bmatrix}1\\2\\2\\0\end{bmatrix} + \begin{bmatrix}1\\2\\1\\1\end{bmatrix} + \begin{bmatrix}2\\2\\1\\1\end{bmatrix} = \begin{bmatrix}4\\6\\4\\2\end{bmatrix} \not\equiv \begin{bmatrix}0\\0\\0\\0\end{bmatrix} \pmod 3$$

From these examples, it is evident that pattern recognition in SET is most elegantly represented as modular arithmetic instead of as logic or natural language (**Sec.** 3.1.1). Therefore, we chose to design our machine learning task using modular arithmetic. To simplify the task further, we focused on validating a single attribute. The task was essentially the following:

$$a + b + c \equiv 0 \pmod 3 \tag{3.2}$$

Here, $a, b, c \in \{0, 1, 2\}$ and there are $3^3 = 27$ possible combinations of values. To translate (3.2) into a typical machine learning task, models are tasked with classifying if the congruence relation holds:

$$1 + 0 + 2 \equiv 0 \pmod 3$$

$$2 + 0 + 2 \not\equiv 0 \pmod 3$$

For some a, b, c input, a model is tasked with outputting $-1, +1$ corresponding to an invalid or valid congruence relation.

In this thesis, we only train RNNs using the single-attribute validation task and inputs of $a, b, c \in \{0, 1, 2\}$ with the intent of eliciting low-dimensional dynamics. However, in our analyses of the learned dynamical mechanisms, we will sketch out how these mechanisms could be extended to compute the multi-attribute validation tasks (**Ch.** 4).

3.2.2 Grokking and modular arithmetic

Modular arithmetic tasks are commonly used in the *grokking literature* [25], [26]. The term "grokking", in the context of deep learning, refers to a phenomenon where small multilayer perceptrons (MLPs) and transformers trained on algorithmic tasks generalize on withheld validation data many training epochs later than fitting to the training data (**Fig.** 3.4)[2]. Refer to **Section** 5.3.1 for a more complete description of the term "grokking".

The modular arithmetic task in the grokking literature takes the following form:

$$a + b \equiv c \pmod p \tag{3.3}$$

[2]This figure is a recreation from [27].

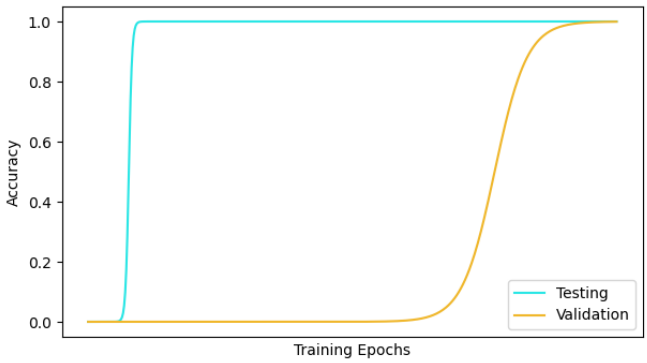


Figure 3.4: An illustration of the grokking phenomenon

In (3.3), $a, b, c \in \{0, 1, ..., p-1\}$ and p is a preset prime number. To perform the task, models are tasked with computing the residue c given inputs a, b. This instance of modular arithmetic is essentially multi-class classification where a model classifies c, unlike (3.2), which is a binary classification task.

There are 2^p possible combinations of inputs and, to elicit grokking, a transformer is trained on 30% on combinations and generalizes to the 70% of combinations left out. Changing the ratio of included and excluded combinations impacts the rate at which models grok [26].

3.3 Task details

Following from (3.2), the design of our machine learning task took the following approach:

1. At some randomly selected time t_0, input value a into the model.
2. At another randomly selected time t_1 where $t_1 > t_0$, input value b.
3. At another randomly selected time t_2 where $t_2 > t_1$, input value c.
4. At a final predetermined time t_3 where $t_3 > t_2$, the model outputs a classification concerning whether the congruence relation was valid or invalid.

Times t_0, t_1, t_2 were randomly selected, more specifically, drawn from a uniform distribution, so as to elicit computational mechanisms that are robust against specific input timings. By designing a task with stimuli presented through time, the trained model is required to develop some form of working memory [11]–[14].

A single instance of our task, referred to as a trial, consisted of 50 time steps. During trials, values were presented in the front-half of the trial and the model's classification was evaluated in the back-half of the trial. Each input was presented to the model for two time steps, followed by a gap where no inputs would be presented for three consecutive time steps. The final 10 time steps of the trial consisted of no inputs. The time steps eligible for input

presentation were selected uniformly at random. **Figure** 3.5 illustrates an example trial from our task with values presented at randomly selected time intervals.

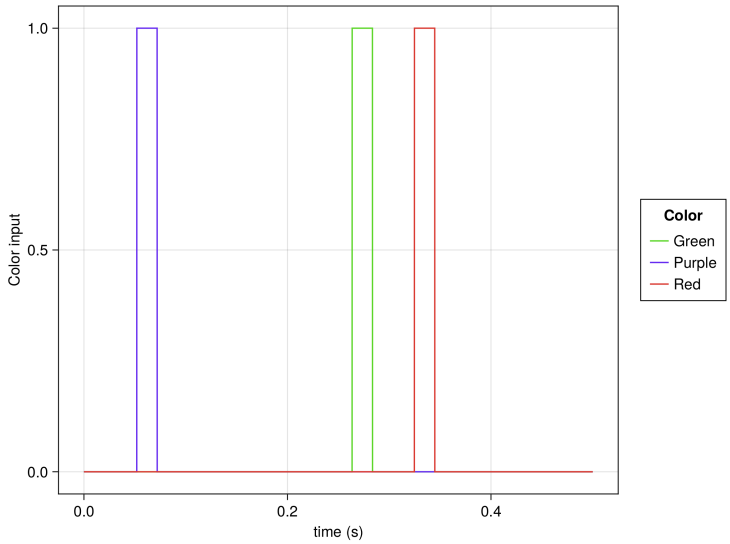


Figure 3.5: An example trial from our task

Throughout this work, we refer to the values of the task, $a, b, c \in \{0, 1, 2\}$, with colors, $a, b, c \in \{green, purple, red\}$. Following from previous work [14], values were embedded within a 100-dimensional vector. Each entry of this vector was sampled from a normal distribution, $\mathcal{N}(0, 1)$. Our choice of 100-dimensional embedding vectors was motivated by our selection of 100 CT-RNN neurons (**Tab.** 3.3); by choosing an embedding vector dimension equivalent to the number of CT-RNN neurons, we minimally bias the dynamics of the CT-RNN because the embedding vectors will have low correlation in the activity space of the network [14]. Within a given training instance, the embedding vectors remained consistent; however, they varied across different training instances. In the absence of any input, a 100-dimensional zero vector was presented to the model.

3.4 Continuous-time recurrent neural networks

As evident from **Chapter** 2, RNNs have a remarkable ability to display a variety of dynamical phenomena. In this section, we lay out the implementation details concerning the RNNs used for this work. **Chapter** 4 describes how particular parameters of an RNN can vastly change the learned dynamics; thus, we place particular emphasis on the parameters selected in our work.

3.4.1 CT-RNN details

The type of RNN implemented in this work was a CT-RNN similar to the RNNs used in the works reviewed in **Chapter** 2 [11]–[14].

$$\tau \dot{x}_i(t) = -x_i(t) + \sum_{k=1}^{N} J_{ik} r_k(t) + \sum_{k=1}^{N^{\text{in}}} B_{ik} u_k(t) + b_i + \eta_i(t) \tag{3.4}$$

$$r_i(t) = \tanh(x_i(t)) \tag{3.5}$$

$$z(t) = \sum_{k=1}^{N} W_k r_k(t) + b_{\text{out}} \tag{3.6}$$

Parameter values and descriptions for (3.4), (3.5), and (3.6) are provided in **Table** 3.3. Equation 3.4 was discretized using Euler's method and took the following form:

$$x_i(t+1) = (1 - \frac{\Delta t}{\tau}) x_i(t) + \frac{\Delta t}{\tau} (\sum_{k=1}^{N} J_{ik} r_k(t) + \sum_{k=1}^{N^{\text{in}}} B_{ik} u_k(t) + b_i + \eta_i(t)) \tag{3.7}$$

Given that our task was a binary classification task, we used a mean squared error (MSE) loss function to evaluate a CT-RNNs performance on our task. We added an L2 regularization onto the rates of the CT-RNN (equation 3.5) which has been shown to increase interpretability [28], [29]. The loss function took the following form:

$$\mathcal{L}(\hat{z}, z, \mathbf{r}) = \frac{1}{5} \sum_{t=T-5}^{T} (\hat{z} - z(t))^2 + \frac{\lambda}{TN} \sum_{t,i=1}^{T,N} r_i^2(t) \Delta t \tag{3.8}$$

Notice how in (3.8), the MSE term only evaluates $z(t)$ for the last five time steps of the trial. This MSE term effectively satisfies criteria 4 in the design of our task from **Section** 3.3.

Trainable parameters $J, B, \vec{b}, W, b_{\text{out}}$ were updated using the AdamW learning algorithm [30]. The learning rate was set to 10^{-3} and the weight decay was set to 10^{-4}. Trainable parameters J, B, W were initialized using the Glorot uniform initializer with the scale set to 1.0 [31]. Trainable parameters $\vec{b}, b_{\text{out}}$ were initialized as zero vectors.

3.4.2 Programming details

Code pertaining to our CT-RNN implementation was written using the Julia programming language [32] with the SciML ecosystem [33] and Lux library [34] and the Python programming language [35] with the JAX ecosystem [36] and Flax library [37]. We initially used Julia due to previous work showing how RNNs trained on the 3-bit flip-flop task could be analyzed using existing Julia libraries [38], but later switched to Python and JAX due to speed considerations.

All code was executed using Intel Xeon Platinum 8260 CPUs on the Massachusetts Institute of Technology (MIT) Supercloud High Performance Cluster (HPC) [39].

Table 3.3: CT-RNN parameters, values, and descriptions

Parameter	**Value**	**Description**
$x_i(t)$	$(-\infty, \infty)$	Voltage of the ith neuron at time t
$u_k(t)$	$(-\infty, \infty)$	kth index of the input value at time t
B_{ik}	$(-\infty, \infty)$	Input weight from kth input index to ith neuron
b_i	$(-\infty, \infty)$	Bias of the ith neuron
$r_t(t)$	$(-1, 1)$	Rate of the ith neuron at time t
J_{ik}	$(-\infty, \infty)$	Recurrent weight between neurons i and k
$z(t)$	$(-\infty, \infty)$	Rate of the output neuron at time t
W_k	$(-\infty, \infty)$	Readout weight from kth neuron
b_{out}	$(-\infty, \infty)$	Bias of the output neuron
$\eta_i(t)$	$\mathcal{N}(0, 0.10)$	Injected noise of the ith neuron at time t
τ	$\{10\text{ms}, 100\text{ms}\}$	Time constant
Δt	10ms	Step size used for Euler's method
N	100	Number of CT-RNN neurons
N^{in}	100	Input dimension
$\hat{z}$	$\{-1, 1\}$	Congruence relation label
T	50	Number of time steps per trial
λ	10^{-4}	L2 regularization

Chapter 4

Dynamical Mechanisms

Having described our modular arithmetic task in the previous chapter (**Ch.** 3), in this chapter, we give a full account of the *dynamical mechanisms* learned by CT-RNNs trained to compute our task. Notably, CT-RNNs can employ one of two distinct dynamical mechanisms to compute modular arithmetic: the *attractive mechanism* or the *oscillatory mechanism*. The *attractive mechanism* computes modular arithmetic by assigning each potential input a vector encoding and summing all encoded vectors given in a trial. The top row of **Figure** 4.1 depicts the attractive mechanism and its encoded vector summations in the PCA space of the CT-RNNs rates (3.5). The *oscillatory mechanism* computes modular arithmetic by assigning each potential input an angular encoding and summing all encoded angles given in a trial. The bottom row of **Figure** 4.1 depicts the oscillatory mechanism and its encoded angle summations in PCA space.

These two mechanisms prove the validity of question 6 in **Section** 1.1:

6. How can computationally similar, yet dynamically distinct systems be experimentally identified?

In principle, both of these mechanisms are plausible dynamical systems a neural system could implement to compute modular arithmetic; therefore, it is a challenge for the 'computation-through-dynamics' framework to experimentally identify these dynamical mechanisms (question 6 in **Sec.** 1.1) and to justify the implementation of one dynamical mechanism versus the other (question 7 in **Sec.** 1.1).

Throughout this chapter, we articulate these dynamical mechanisms with abstracted models. To explain the attractive mechanism, we model it as matrix multiplication with a vector of inputs (**Sec.** 4.1.1). To explain the oscillatory mechanism, we reverse engineer a trained CT-RNN implementing the mechanism (**Sec.** 4.2.1) and formalize our insights with a sinusoidal model (**Sec.** 4.2.2). A benefit to formalizing these abstract models is that they provide insight as to how these dynamical mechanisms could be extended to the multi-attribute classification task (**Sec.** 4.1.2 and **Sec.** 4.2.3). At the end of this chapter, we introduce the framework of finite-state machines (FSM) as a method to formally compare the computations performed by the attractive and oscillatory mechanisms (**Sec.** 4.3).

We would like to briefly note that there are a variety of terms used to describe dynamical systems. Throughout **Chapters** 1 and 2, we primarily used the terms "dynamical systems" and "dynamical phenomena". In this chapter and beyond, we use the term "dynamical

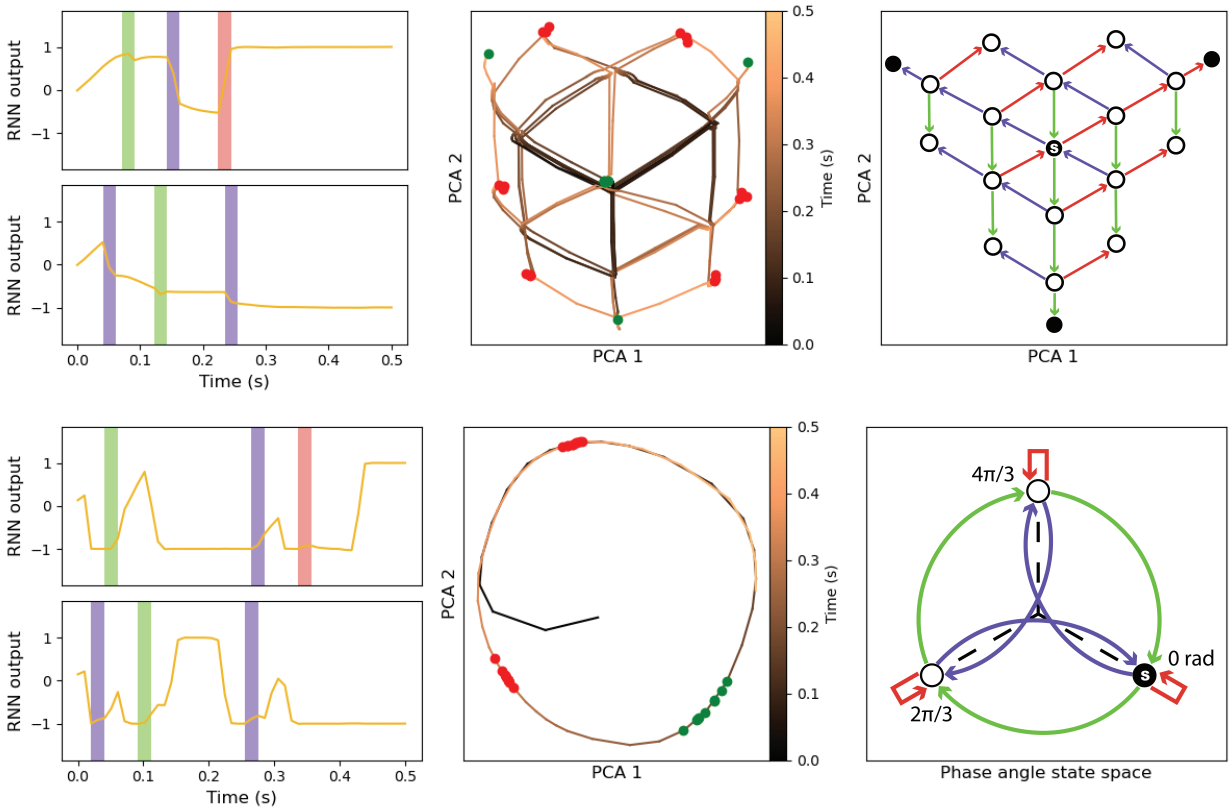


Figure 4.1: Comparing the attractive and oscillatory mechanisms

mechanism" to describe the dynamical systems learned by our trained CT-RNNs to compute modular arithmetic. Our choice of the term "dynamical mechanisms" was motivated by previous philosophical literature that sought to defend a mechanistic account of physical, computational processes [40]. While we make no claims about the validity of the account described in [40], we find their use of the term "mechanism" as being an apt indicator of *computational operations*. In this thesis, we use the term "dynamical mechanisms" as a means of emphasizing not only the dynamical systems learned by our trained CT-RNNs, but also their computational operations.

4.1 Attractive mechanism

The top row of **Figure** 4.1 showcases the prominent characteristics of the *attractive mechanism*. Input values cause an instantaneous change in the activity of the CT-RNN. Visualized in the space of the first two PCs, the input values cause the activity of the CR-RNN to transition between different fixed-point attractors. These attractors are arranged in a lattice-like configuration so as to allow inputs of similar values to cause regular transitions among the attractors. Notice how some attractors correspond to valid congruence relations (green points in **Fig.** 4.1) and others correspond to invalid relations (red points in **Fig.** 4.1).

To formally articulate how the attractive mechanism computes modular arithmetic, we construct a model of the mechanism rooted in linear algebra. This model, called the *vector abstraction*, views the CT-RNN as learning a matrix, where input combinations are mapped to state-space vectors, and a feature map, were state-space vectors are mapped to classification outputs. While there are other ways to interpret the attractive mechanism, as described in **Appendix** A, the vector abstraction is particularly effective due to its parallels with feedforward multilayer perceptrons (MLPs) and its ability to naturally extend to the multi-attribute task.

4.1.1 Vector abstraction

To formulate the vector abstraction, we first observe, in the top row of **Figure** 4.1, that input values can be represented by vectors:

$$\vec{g} \coloneqq \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad \vec{p} \coloneqq \begin{bmatrix} -\frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix}, \quad \vec{r} \coloneqq \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix}$$

Here, $\vec{g}$ corresponds to a *green* color input and a value of 0, $\vec{p}$ corresponds to a *purple* color input and a value of 1, and $\vec{r}$ corresponds to a *red* color input and a value of 2. The final point of a CT-RNN's activity can now be described through adding the representation vectors for the corresponding input value combination:

$$\vec{g} + \vec{p} + \vec{r} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} + \begin{bmatrix} -\frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix} + \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\vec{g} + \vec{p} + \vec{p} = \vec{g} + 2\vec{p} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} + 2 \begin{bmatrix} -\frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} -\sqrt{3} \\ 0 \end{bmatrix}$$

It follows that each representation vector can be a column in a matrix where the input vector is an integer encoding of the number of input values presented during a trial and the output vector describes the final point of a CT-RNN's activity:

$$W\vec{x} = \begin{bmatrix} | & | & | \\ \vec{g} & \vec{p} & \vec{r} \\ | & | & | \end{bmatrix} \vec{x} = \begin{bmatrix} 0 & -\frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \\ -1 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \vec{x} = \vec{y} \tag{4.1}$$

Here, $\vec{x}$ is the vector of input color combinations, $\vec{x}^T = \begin{bmatrix} x_0 & x_1 & x_2 \end{bmatrix}$, x_0 corresponds to the number of *green* colors presented during a trial, x_1 corresponds to the number of *purple* colors, and x_2 corresponds to the number of *red* colors. Further note how an input vector of all ones, $\vec{x}^T = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$, maps to the null space of W. This is a desirable property as the attractive mechanism maps trials of all different stimuli to the origin of its PCA space (top row of **Fig.** 4.1). **Figure** 4.2 depicts a variety of possible state space vectors $\vec{y}$ from (4.1).

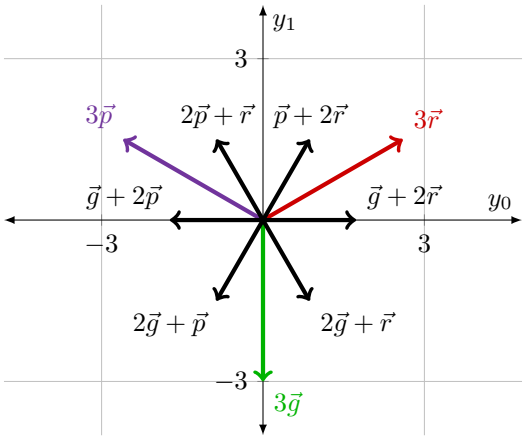


Figure 4.2: State space vectors $\vec{y}$

Notice how state-space vectors $\vec{y}$ mapping to valid congruence relations, classification of $+1$, and invalid congruence relations, classification of -1, are not linearly separable. Taking inspiration from *kernel methods* [41], we can make state-space vectors $\vec{y}$ linearly separable with a feature map:

$$F(\vec{y}) = 1 - \frac{(\vec{y}^T\vec{y})^2}{a} e^{-\frac{(\vec{y}^T\vec{y})^2}{b}} \tag{4.2}$$

$$\Phi(\vec{y}) = (y_0, y_1, F(\vec{y})) \tag{4.3}$$

We used a modified 2D Mexican hat wavelet[1] as the feature map (4.2) to map valid congruence relation state-space vectors $\vec{y}$ to 1 and invalid vectors to -1. **Figure** 4.3 illustrates how (4.2) and (4.3) correctly map state space vectors to their appropriate classification and allow for linear classification with a hyperplane at $y_2 = 0$.

[1]Using the SymPy python library [42], we solved for parameters a, b as $a = 0.931194835465213$ and $b = 5.0625$.

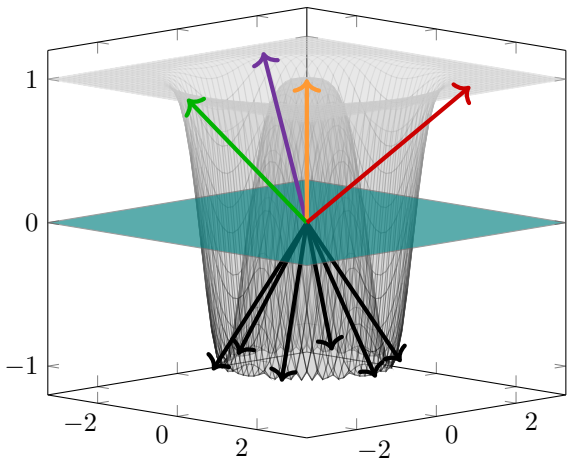


Figure 4.3: State space vectors $\vec{y}$ with the feature map

Incidentally, while the vector abstraction model applies to the attractive mechanism learned by a CT-RNN, a recurrent network, the mathematical form of the interpretation is similar to a feedforward MLP.

$$F(W\vec{x}) = z \tag{4.4}$$

Equation (4.4) not only describes the application of the feature map (4.2) after the matrix multiplication with representation vectors (4.1), but also the application of an activation function after the matrix multiplication with a weight matrix. While the activation function (4.2) is nonlinear like many activation functions used for MLPs, it is nonstandard in the deep learning literature [43].

4.1.2 Multi-attribute classification with vectors

In **Section** 3.1, we described the card game SET as involving multiple attributes and validating the *rule of SET* across all of these attributes. However, in **Section** 3.3, we simplified our modular arithmetic task to concern the validation of only one attribute, referred to as the color attribute. Through modeling the attractive mechanism with the vector abstraction, the extension of the mechanism to the multi-attribute task is simple.

Consider a multi-attribute task to involve validating two attributes: color and shape. We can modify (4.1) to be the following:

$$W\vec{x} = \begin{bmatrix} | & | & | & | & | & | \\ \vec{g} & \vec{p} & \vec{r} & \vec{o} & \vec{s} & \vec{d} \\ | & | & | & | & | & | \end{bmatrix} \vec{x} = \vec{y} \tag{4.5}$$

Vectors $\vec{g}$, $\vec{p}$, and $\vec{r}$ still refer to the values of the color attribute; but now, we introduce the vectors of $\vec{o}$, $\vec{s}$, and $\vec{d}$ to refer to the values of the shape attribute: oval, squiggle, and diamond. Essentially, to extend the attractive mechanism to the multi-attribute task, we need only to vector encode all values across all attributes with vector $\vec{x}$ now corresponding to a vector of all value combinations.

While we could rely on the PCA trajectories of CT-RNNs implementing the attractive mechanism as a guide for creating vector encodings in the case of the single-attribute task (top-middle plot in **Fig.** 4.1), we did not train a CT-RNN on the multi-attribute task, so we are left to construct the vector encodings for the multi-attribute task by hand. Here is the simplest encoding:

$$\vec{g} := \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix}, \quad \vec{p} := \begin{bmatrix} -\frac{\sqrt{3}}{2} \\ \frac{1}{2} \\ 0 \\ 0 \end{bmatrix}, \quad \vec{r} := \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \\ 0 \\ 0 \end{bmatrix}, \quad \vec{o} := \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}, \quad \vec{s} := \begin{bmatrix} 0 \\ 0 \\ -\frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix}, \quad \vec{d} := \begin{bmatrix} 0 \\ 0 \\ \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix}$$

Notice how each shape vectors is essentially a permuted color vector (e.g., $\vec{o}$ is a permuted $\vec{g}$). This choice of encoding allows for the two attributes to be validated in separate spaces, concatenated together, and mapped to a classification using our previously defined feature map (4.3). This style of vector encoding can also extend to multi-attribute tasks of 3 or more attributes.

4.2 Oscillatory mechanism

The bottom row of **Figure** 4.1 showcases the prominent characteristics of the oscillatory mechanism. The mechanism implements a limit cycle in the state space of the CT-RNN and input values cause an instantaneous shift in the phase angle of the limit cycle. The CT-RNN begins with a phase angle corresponding to valid congruence relations and input values shift the phase angle to invalid phases. This mechanism essentially mirrors a clock, the natural representation for modular arithmetic, and input values cause the phase angle to traverse around the clock. Computing $2 + 2 + 1 \equiv 2 \pmod 3$ is similar to computing $8\text{am} + 6 \text{ hours} = 2\text{pm}$, and the oscillatory mechanism captures this similarity.

Given the attention that attractive dynamics have seen from the neuroscience community [44], we find the oscillatory mechanism to be especially noteworthy. The emergence of the mechanism was conditioned on setting τ in (3.4) to 10ms, while the emergence of the attractive mechanism was conditioned on $\tau = 100\text{ms}$. Previous RNN literature has noted the emergence of oscillatory dynamics; however, these works either had associated time constants considerably greater than 10ms [14], [45] or did not investigate CT-RNNs [46]. Remarkably, this mechanism bears a resemblance to oscillatory models of grid cells hypothesized to exist at the cellular level [47], potentially mirrored more closely by setting τ to 10ms.

To formally articulate how the oscillatory mechanism computes modular arithmetic, we *reverse engineer* a trained CT-RNN implementing the mechanism (**Sec.** 4.2.1), construct a model of the mechanism (**Sec.** 4.2.2), and extend this model to the multi-attribute task (**Sec.** 4.2.3). The model, called the *sinusoidal abstraction*, views the CT-RNN as learning a cosine function that integrates angular encoded inputs in the phase-shift term.

4.2.1 Reverse engineering

Reverse engineering has been widely employed to understand the computations of trained RNNs [11], [17], [28], [48]. While there is no explicit methodology for how to reverse engineer

an RNN, we use this term to broadly refer to systematic efforts that seek to understand how RNNs compute. Note that our efforts to reverse engineer the oscillatory mechanism are entirely taken from our previous work [15] and that our results were coded and plotted using the Julia programming language [32].

We began the reverse engineering process by interpreting the outputs of a trained CT-RNN employing the oscillatory mechanism. Under the condition of no input values, the output of the mechanism traverses through two full cycles over the course of a trial (**Fig.** 4.4, middle row). Under the presence of an input value, the mechanism experiences a *perturbation* which appears to shift the current phase of the oscillation (**Fig.** 4.4, top and bottom rows). However, interpreting the output of the mechanism alone is insufficient to understand the computational significance of these perturbations.

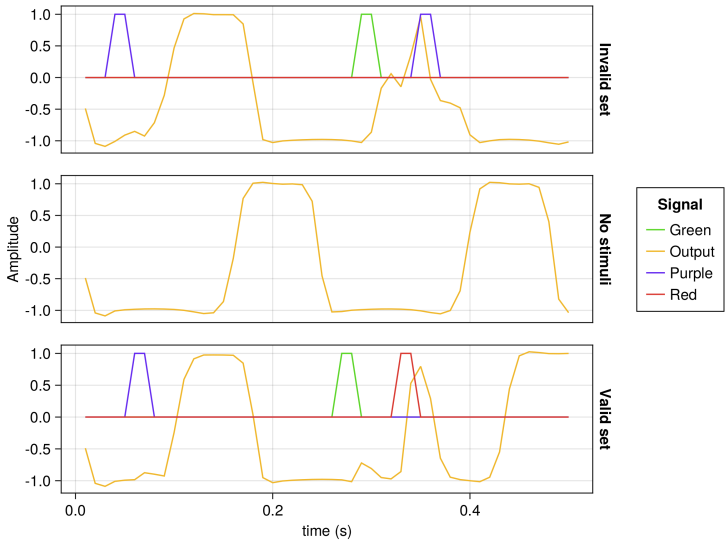


Figure 4.4: Example outputs of the oscillatory mechanism

Principal component analysis (PCA) on the rates of a CT-RNN (3.5) revealed how CT-RNNs implementing the oscillatory mechanism learn a low-dimensional limit cycle (**Fig.** 4.5). In **Figure** 4.5, there are a variety of *end points*, red or green, plotted around the limit cycle. These end points refer to the final point of the CT-RNNs dynamics during a trial; green points corresponding to trials with valid congruence relations and red corresponding to trials with invalid congruence relations. These end points are grouped into three distinct distributions with each distribution corresponding uniformly to trials with valid or invalid congruence relations. We used these distributions as evidence as to how the perturbations seen in **Figure** 4.4 might impact the phase angle of the limit cycle.

To further understand how input values affected the phase angle of the oscillatory mechanism's limit cycle, we rotated the PCA trajectories by 60^o. Note how when projecting all rates onto PC 1, overlap occurs between the valid end point distribution and one of the invalid distributions. By rotating the PCA trajectories by 60^o, we minimize overlap between valid and invalid distributions in the first rotated principal component (rPC 1). **Figure** 4.6a plots the *oscillatory mechanisms* dynamics onto rPC 1 for various trials of uniform input value. Here, we can distinguish the unique effects that each type of input value has on the

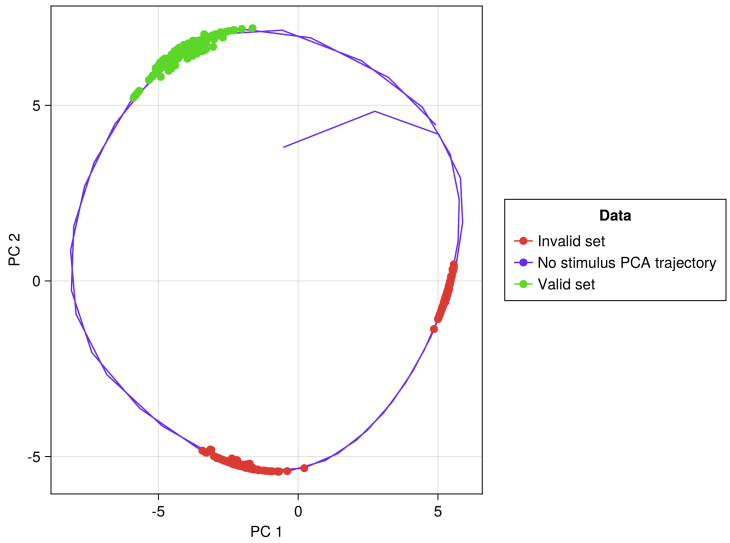
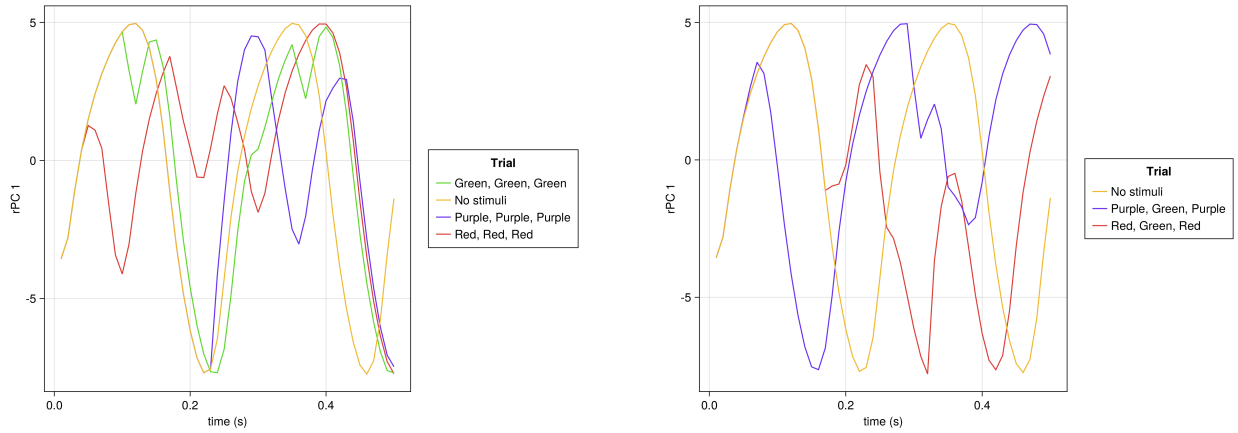


Figure 4.5: PCA of the oscillatory mechanism

phase angle of the mechanism's cycle.

Our first observation was how there is little difference between a trial consisting of only *green* values, corresponding to an input value of 0, and a trial with no input values (**Fig.** 4.6a, green trajectory). This indicated that the *green* values had little impact on the phase of the mechanism's cycle. Our second observation was how the dynamics for a trial consisting of only *purple* values, corresponding to an input value of 1, appear to make three complete cycles over the course of a trial. We hypothesized that *purple* values shift the phase angle of the limit cycle forward by $\frac{2\pi}{3}$, so that when three *purple* values are presented, the limit cycle returns to its initial phase and indicates a valid congruence relation. Our third and final observation was how, conversely with *purple* values, the dynamics for a trial consisting of only *red* values, corresponding to an input value of 2, appear to only complete a single cycle during the trial. We hypothesized that *red* values shift the phase angle of the limit cycle backward by $\frac{2\pi}{3}$.

To summarize our observations from **Figure** 4.6a, we hypothesized that *green* input values don't shift the phase angle of the limit cycle, *purple* values shift the cycle by $\frac{2\pi}{3}$, and *red* values shift the cycle by $-\frac{2\pi}{3}$. To test our hypotheses, we predicted the ending phase of the mechanism's dynamics for trials consisting of invalid congruence relations. For a trial consisting of a *purple*, *green*, *purple* input value combination, the end point of the network's dynamics should be $\frac{2\pi}{3}$ radians *behind* the phase angle corresponding to a valid congruence relation. For a trial consisting of a *red*, *green*, *red* input value combination, the end point of the network's dynamics should be $\frac{2\pi}{3}$ radians *ahead* of the valid congruence relation phase angle. **Figure** 4.6b confirms these predictions and supports our reverse-engineered interpretation of the oscillatory mechanism.



(a) Valid congruence relations

(b) Invalid congruence relations

Figure 4.6: rPC 1 over time for various congruence relations

4.2.2 Sinusoidal abstraction

In reverse engineering the oscillatory mechanism, we found that the mechanism works by transforming input values into phase angle shifts (**Sec.** 4.2.1). Given its simplicity, we can mathematically formulate this mechanism. Consider a cosine function to represent the network's limit cycle dynamics:

$$g(t) = \cos(wt) \tag{4.6}$$

The oscillatory mechanism modifies (4.6) to:

$$f(t) = \cos\left(wt + \int_0^t \phi(t')dt'\right) \tag{4.7}$$

$\phi(t')$ denotes the phase shift at time t', derived from the external input:

$$\phi(t) = \begin{cases} +\frac{2\pi}{3} & \text{if green at } t \\ -\frac{2\pi}{3} & \text{if purple at } t \\ 0 & \text{if red at } t \end{cases} \tag{4.8}$$

Note that the specific angular encodings described in (4.8) are arbitrary. For the case of the oscillatory mechanism described in **Figure** 4.4, (4.8) would not describe the correct encodings.

In **Figure** 4.7, we plot (4.6), referred to as the "valid congruence phase", and (4.7) with (4.8) referred to as the oscillatory mechanism. Here, it is clear to see how the presentation of input values shifts the mechanism in and out of phase with the valid congruence phase depending on the history of presented input values.

Furthermore, not only do (4.7) and (4.8) have descriptive power in explaining the oscillatory mechanism, but also predictive power in predicting the dynamics of a CT-RNN implementing the mechanism. To illustrate this point, we can modify (4.7) and (4.8) to the

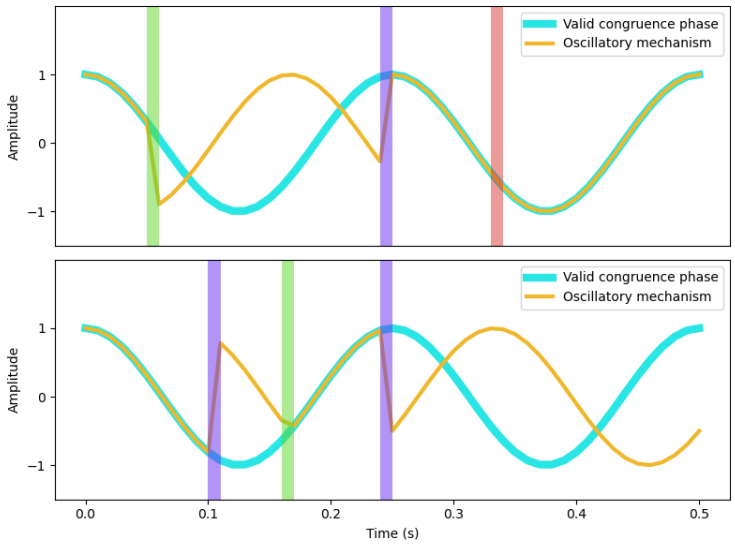


Figure 4.7: The sinusoidal model of the oscillatory mechanism

following[2]:

$$f(t) = 7\sin\left(\frac{2\pi}{0.29}t + \int_0^t \phi(t')dt'\right) - 1.5 \tag{4.9}$$

$$\phi(t) = \begin{cases} +\frac{2\pi}{3} & \text{if purple at } t \\ -\frac{2\pi}{3} & \text{if red at } t \\ 0 & \text{if green at } t \end{cases} \tag{4.10}$$

Plotting (4.9) and (4.10) against the rPC 1 dynamics described in **Figure** 4.6a and **Figure** 4.6b, we can see how (4.9) and (4.10) reproduce the trajectory of the CT-RNN after a perturbation caused by an input (**Fig.** 4.8).

While the sinusoidal abstraction of the oscillatory mechanism has descriptive and predictive merits, we made two notable simplifications. In **Figure** 4.6a, it is evident that *green* input values do have a slight impact on the limit cycle's phase because of the difference in ending phases between the only *green* trial and the no stimulus trial. We performed this simplification because incorporating non-zero *green* value phase shifts into the sinusoidal abstraction would detract from the simplicity of the phase-angle integration in (4.8). In (4.7), we simplified the oscillation to be uniform; however, **Figure** 4.5 shows how trajectories along the limit cycle are non-uniformly sampled[3], and thus, indicative of how the oscillatory mechanism is a nonuniform oscillator[4]. We performed this simplification because accounting for nonuniform oscillations in (4.7) would require differential equations that would ultimately not serve a computational purpose. To justify all simplifications made, we refer to the predictive power of the sinusoidal abstraction (**Fig.** 4.8).

[2]Note that the parameters used in (4.9) were obtained through an iterative "guess-and-check" approach instead of a more principled optimization method.

[3]Refer to how the purple trajectory near the green points is more jagged than the purple trajectory near the red points.

[4]Refer to Ch. 4 of [2] for an overview on uniform and nonuniform oscillators.

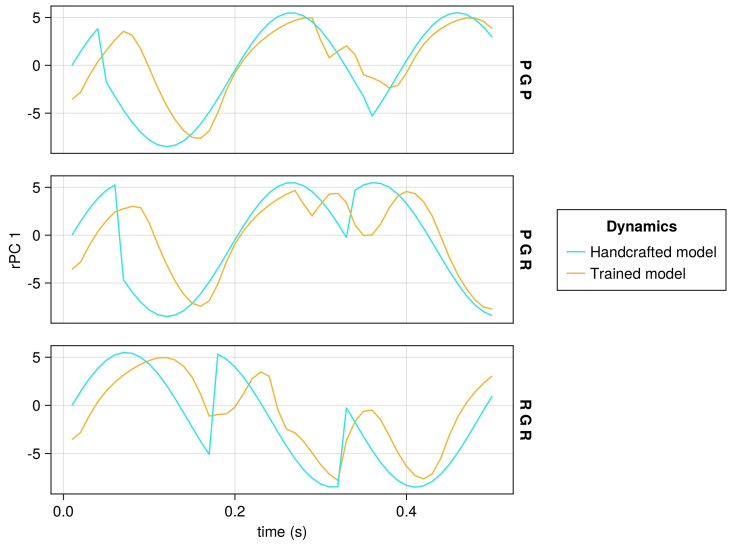


Figure 4.8: The sinusoidal abstraction can predict CT-RNN dynamics

One more observation to note is how the oscillatory mechanism does not need to oscillate in order to compute modular arithmetic. Consider setting $w = 0$ in (4.7). The sinusoidal abstraction can still accurately classify congruence relations and might even have greater biological plausibility given its similarities with the head-direction system [49], [50]. One potential reason for the oscillatory activity of this mechanism is how oscillations might allow for more persistent learning signals during backpropagation [46], [51]. In **Appendix** B, we make note of one particular instance where a ring attractor mechanism did emerge.

4.2.3 Multi-attribute classification with sinusoids

Just as extending the attractive mechanism to compute the multi-attribute task becomes simple with the vector abstraction (**Sec.** 4.1.2), extending the oscillatory mechanism to compute the multi-attribute task also becomes simple with the sinusoidal abstraction. In the case of multi-attribute validation with color and shape attributes, we can modify (4.7) and (4.8) to the following:

$$f(t) = \cos\left(wt + \int_0^t \phi(t')dt'\right) + \cos\left(wt + \int_0^t \varphi(t')dt'\right) \tag{4.11}$$

$$\phi(t) = \begin{cases} +\frac{2\pi}{3} & \text{if green at } t \\ -\frac{2\pi}{3} & \text{if purple at } t \\ 0 & \text{if red at } t \end{cases}, \quad \varphi(t) = \begin{cases} +\frac{2\pi}{3} & \text{if oval at } t \\ -\frac{2\pi}{3} & \text{if squiggle at } t \\ 0 & \text{if diamond at } t \end{cases} \tag{4.12}$$

To extend the sinusoidal abstraction to the multi-attribute task, each attribute is assigned a unique cosine function and phase-shift integrator. In (4.11) and (4.12), $\phi(t')$ corresponds to color input values and $\varphi(t')$ corresponds to shape input values with arbitrary angular encodings. Like in (4.7), classifying valid input combinations for the multi-attribute task becomes a matter of detecting if (4.11) is in phase with the valid congruence phase (4.6).

4.3 Finite-state machines

The attractive and oscillatory mechanisms are remarkable in how they capture the same computation, modular arithmetic, with entirely distinct dynamics. Having articulated these dynamical mechanisms with simple mathematical models in **Sections** 4.1 and 4.2, we now compare these mechanisms using the framework of finite-state machines (FSMs) in *automata theory*. Automata theory is the study of abstract machines and has a rich history in computer science [52] and cognitive science [53]. In this section, we explain the FSM formalization using turnstiles (**Sec.** 4.3.1) and then formalize our dynamical mechanisms with FSMs to show how their primary computational difference resides in the FSM transition function, δ (**Sec.** 4.3.2).

4.3.1 A toy example: turnstiles

A turnstile is often used as the prototypical example of a FSM, see, for instance, [54]. When a user approaches a turnstile, the *starting* state is the *locked* state. To pass through, the user inputs a *coin* and then *pushes* on the bar of the turnstile. The *coin* transitions the turnstiles state into the *unlocked* state, and by *pushing* the bar, the user transitions the turnstile back to the *locked* state. The turnstile is essentially a machine of two states, *locked* and *unlocked*, capable of two actions, *coin* and *push*. **Figure** 4.9 illustrates a graphical representation of the FSM model of a turnstile. Note the presence of two state-action pairs that were not explicitly described: *pushing* a *locked* turnstile and inserting a *coin* in an *unlocked* turnstile both fail to change the turnstile's state.

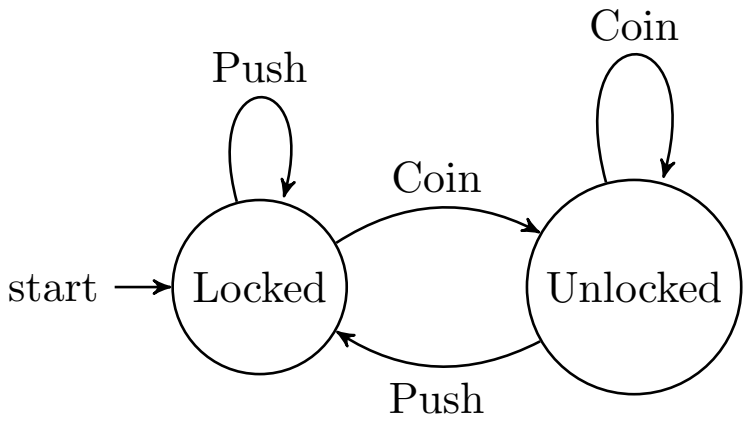


Figure 4.9: A finite state machine model of a turnstile

To formalize our FSM model of the turnstile, we use the definition of an FSM as described in [54]. A *finite-state machine* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q represents the set of all states, $Q = \{q_0, q_1, \ldots\}$
- Σ represents the set of all inputs called the *alphabet*, $\Sigma = \{a_0, a_1, \ldots\}$
- δ represents the transition function of states and inputs to states, $\delta(q_i, a_i) \rightarrow q_j$

- q_0 represents the starting state
- F represents the set of all *accepting* states, $F \subseteq Q$

In the case of our turnstile example, the 5-tuple is now

- $Q = \{\text{locked}, \text{unlocked}\}$
- $\Sigma = \{\text{coin}, \text{push}\}$
- δ is described as

	coin	push
locked	unlocked	locked
unlocked	unlocked	locked

- $q_0 = \text{locked}$
- $F = \emptyset$

In the turnstile example, there are no accepting states, so F is the empty set.

Now, with the 5-tuple formalized, we can input *strings* into the turnstile FSM. A string is an ordered list of actions passed into the FSM. Each action in the string causes the FSM to transition states according to δ. For example, the string [*coin*,*push*,*push*,*coin*] would cause the turnstile FSM to traverse states [*locked*,*unlocked*,*locked*,*locked*,*unlocked*] with the first state corresponding to the starting state q_0.

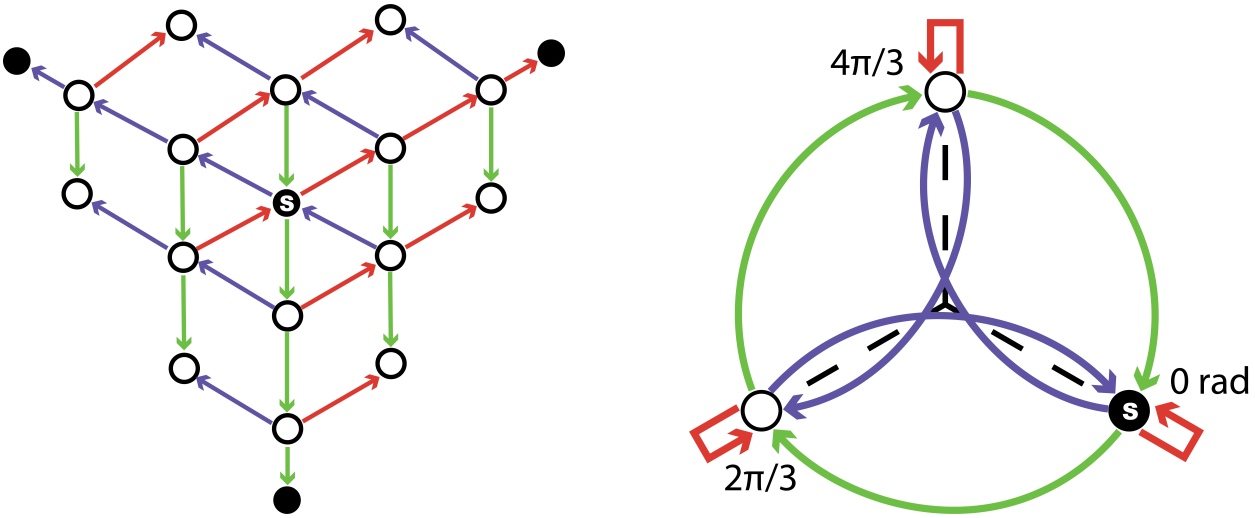
While the turnstile example is simple, it illustrates how to formally reason about states, actions, and state-action functions. Using this terminology and formalism, we can now describe and formally compare the attractive and oscillatory mechanisms.

4.3.2 State machines and dynamical mechanisms

The interpretation of the attractive mechanism as an FSM is straightforward. Each fixed-point attractor in the phase space of the CT-RNN corresponds to a state and the three input values correspond to an input in the alphabet. The interpretation of the oscillatory mechanism as an FSM is similar to the attractive mechanism, but now the phase angles of the limit cycle correspond to states in the FSM. The accepting states are now the fixed-point attractors and limit cycle phase angles that correspond to valid congruence relations. **Figure** 4.10 depicts the FSMs for both mechanisms with filled nodes indicating accepting states and colored arrows represent input value transitions between states.

The attractive mechanism FSM (**Fig.** 4.10a) can be formalized as follows:

- $Q = \{q_i \mid 0 \leq i \leq 18\}$
- $\Sigma = \{g, p, r\}$



(a) Attractive mechanism FSM

(b) Oscillatory mechanism FSM

Figure 4.10: Finite-state machines for both dynamical mechanisms

- δ is described as

	g	p	r
q_0	q_4	q_6	q_2
q_1	q_0	q_{10}	q_{11}
q_2	q_3	q_2	q_{12}
q_3	q_{15}	q_0	q_{14}
q_4	q_{16}	q_5	q_3
q_5	q_{18}	q_7	q_0
q_6	q_5	q_8	q_1
q_7	—	—	—
q_8	q_7	q_9	q_{10}
q_9	—	—	—
q_{10}	—	—	—
q_{11}	—	—	—
q_{12}	q_{14}	q_{11}	q_{13}
q_{13}	—	—	—
q_{14}	—	—	—
q_{15}	—	—	—
q_{16}	q_{17}	q_{18}	q_{15}
q_{17}	—	—	—
q_{18}	—	—	—

- Starting state is q_0
- $F = \{q_0, q_9, q_{13}, q_{17}\}$

In the transition function δ, states with rows of "—" indicate that the transition function is not specified for those states since our modular arithmetic task concerns only strings with three symbols.

The *oscillatory mechanism* FSM (**Fig.** 4.10b) can be formalized as follows:

- $Q = \{q_0, q_{\frac{2\pi}{3}}, q_{\frac{4\pi}{3}}\}$
- $\Sigma = \{g, p, r\}$
- δ is described as

	g	p	r
q_0	$q_{\frac{2\pi}{3}}$	$q_{\frac{4\pi}{3}}$	q_0
$q_{\frac{2\pi}{3}}$	$q_{\frac{4\pi}{3}}$	q_0	$q_{\frac{2\pi}{3}}$
$q_{\frac{4\pi}{3}}$	q_0	$q_{\frac{2\pi}{3}}$	$q_{\frac{4\pi}{3}}$

- Starting state is q_0
- $F = \{q_0\}$

On first inspection, it is apparent that the oscillatory mechanism FSM most naturally computes our modular arithmetic task despite both FSMs recognizing the same language[5]. The oscillatory mechanism has 3 states compared to the attractive mechanism's 19 states. This is due to the oscillatory mechanism's angular encoding of input values compared to the attractive mechanism's vector encoding. From these observations, we hypothesized that the oscillatory mechanism would generalize more frequently to unseen congruence relations than the attractive mechanism. We tested this hypothesis in **Section** 5.3.

To conclude this chapter, we would like to note a historical irony. The creation of the artificial neuron [55], sometimes referred to as the McCulloch-Pitts neuron, spawned two disparate fields of research: automata theory [56] and neural networks [57]. Now, our FSM interpretations of the dynamical mechanisms learned in trained CT-RNN seek to join these long separated fields. While interpreting the computations of RNNs with FSM is not entirely novel [58], we speculate that further investigating the intersection of RNNs and FSM will lead to enhanced interpretability of trained RNNs.

[5]Note that we use the term "language" here to refer to all possible three symbol strings of alphabet Σ.

Chapter 5

Behavioral Phenomena

In **Chapter** 3, we described our modular arithmetic task, and in **Chapter** 4, we described the dynamical mechanisms, the attractive and oscillatory mechanisms, capable of computing our task. Both of these chapters effectively serve as background for our answer to question 6 in **Chapter** 1:

6. How can computationally similar, yet dynamically distinct systems be experimentally identified?

This chapter answers question 6 in the following manner:

> Computationally-similar dynamical systems can be experimentally identified with *behavioral phenomena*.

While the attractive and oscillatory mechanisms both compute modular arithmetic, their computational operations are dynamically distinct. Through designing behavioral experiments that target the differences between these computational operations, we can experimentally distinguish between computationally-similar dynamical systems. In addition, these behavioral experiments also provide insight as to why one dynamical mechanism might be implemented versus the other mechanism, thus addressing question 7 in **Chapter** 1.

Before detailing our behavioral experiments, we first outline our motivations for investigating behavioral means to identify dynamical systems as opposed to neurological means (**Sec.** 5.1). Then, we describe our first behavioral experiment using *psychometric signatures* to show how CT-RNNs with different time constants τ produce distinct psychometric curves (**Sec.** 5.2). Next, we describe our second behavioral experiment using *grokking curves* to show how CT-RNNs implementing different dynamical mechanisms generalize to unseen data with different probabilities (**Sec.** 5.3). Finally, we conclude the chapter by using behavioral phenomena to address question 7 (**Sec.** 5.4).

5.1 Motivating behavior to identify mechanisms

To uncover the dynamical system implemented by a neural system performing behaviorally-relevant computations, an experimental neuroscientist can implant a variety tetrodes into the neural system, record from the system as it performs the computations, and analyze

the recorded data to extract the dynamical system. Such experiments are routine among experimental neuroscientists and have provided fruitful dynamical insights into how neural systems compute [7], [17], [59], [60]. However, as discussed in **Section** 2.2, the dynamical systems inferred from neural data can inaccurately depict the implemented computational mechanisms. But furthermore, as discussed in [7], [8], the *geometry* of two dynamical systems can appear similar with their corresponding dynamical systems being dynamically distinct.

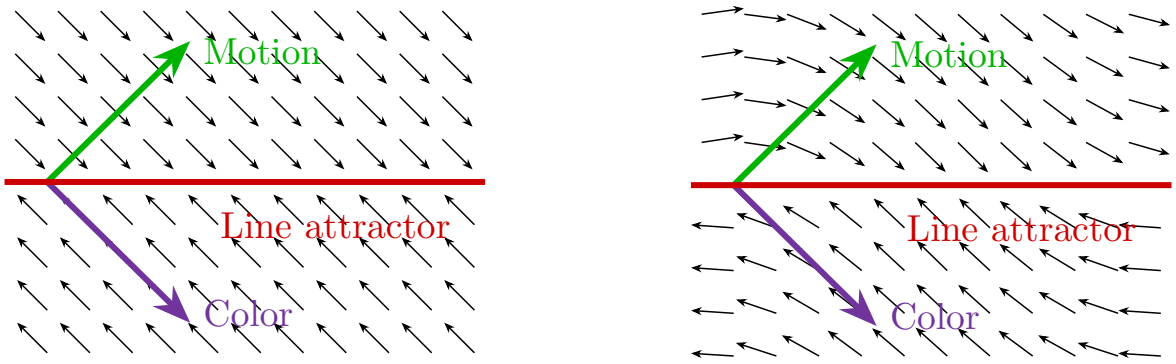
For example, **Figure** 5.1 depicts two line attractors capable of computing the context-dependent decision making task described in **Section** 2.2.1; yet, these dynamical systems exhibit distinct vector fields. For the dynamical system in **Figure** 5.1a, the vector field is described as the following:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -y \end{aligned} \tag{5.1}$$

For the dynamical system in **Figure** 5.1b, the vector field is described as the following:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -\sin y \cos x \end{aligned} \tag{5.2}$$

These two dynamical systems, (5.1) and (5.2), are distinct dynamical systems, yet they exhibit similar geometries and perform the same computation. An experimental neuroscientist performing PCA on neural data associated with the context-dependent decision making task would not be able to identify which dynamical system the neural system implemented; therefore, new neural analysis techniques were developed in [7], [8] to allow for a more accurate and rigorous identification of dynamical systems.



(a) Typical line attractor

(b) Sinusoidal line attractor

Figure 5.1: Different dynamics compute the content-dependent decision making task

To experimentally distinguish between the dynamical mechanisms for modular arithmetic described in **Chapter** 4, we could advocate for the use of the neural analysis methods described in [7], [8]; however, these methods were designed to experimentally distinguish *geometrically similar* dynamical systems, like in **Figure** 5.1. Given that our dynamical mechanisms are *geometrically distinct*, as apparent in **Figure** 4.1, we speculated that neural

analysis methods are not necessary to experimentally distinguish our dynamical mechanisms. Instead, we hypothesized that behavioral phenomena would be sufficient to experimentally distinguish our dynamical mechanisms.

Human behavior is known to produce experimentally rich data that are indicative of underlying computational mechanisms; yet, behavior has been underused in building neural network models of cognition [61]. Our motivation for using behavioral phenomena to distinguish dynamical mechanisms in this chapter was to provide examples as to how behavior can play a key role in the 'computation-through-dynamics' framework. In particular, we show that using behavioral phenomena can sufficiently answer question 6 without the need to resort to neurological phenomena.

We use the term "behavioral phenomena" in this thesis as a means of distinguishing *input/output behavior* and *internal behavior*. For example, our dynamical mechanisms both achieve perfect classification accuracy on our modular arithmetic task, so in a sense, these mechanisms exhibit the same "behavior". However, this usage of "behavior" is only related to performance on the task, an input/output behavior, and not related to the operations the neural system took in performing the task, i.e. how the neural system *implemented* the task. *Behavioral phenomena* are behaviors related to the operations performed by a mechanism, i.e. the internal behavior. For example, the *types of errors* a mechanism performs in a task is indicative of internal behavior, but the *overall error rate* is indicative of input/output behavior.

5.2 Psychometric signatures

Our first experiment detailing how behavioral phenomena can address question 6 sought to use methods from the field of psychophysics. The field of psychophysics seeks to understand how parametrically varied stimuli can result in measurably-different behavior [62]. Functions that relate parametrically varied stimuli to measured behavior are termed "psychometric curves". In this section, we perform a psychometric experiment on CT-RNNs computing our modular arithmetic task and show how CT-RNNs implementing different dynamical mechanisms produce distinct psychometric curves. We refer to these curves as "psychometric signatures" since they could be used to identify the dynamical mechanism responsible for modular arithmetic in neural systems.

5.2.1 Psychophysics literature

Before describing our psychometric experiment and associated results, we review two previous works that motivated our investigation of psychometric signatures.

In [63], the authors investigated the *explore-versus-exploit dilemma* in a two-armed bandit task through the lens of two distinct algorithms: the *Upper Bound Confidence* (UBC) algorithm and the *Thompson sampling* algorithm. The authors show how these algorithms produced two distinct sigmoidal-shaped, 'probability-of-choice' curves with the UBC algorithm influencing the bias of the curve and the Thompson sampling algorithm influencing the slope of the curve. These 'probability-of-choice' curves are effectively psychometric curves where the input, the x-axis, is the expected value difference between two task-related choices

and the output, the y-axis, is the probability of choosing the first task-related choice over the second. Via collecting experimental data on humans performing the same task, they further showed how a hybrid of the two algorithms provides the best quantitative account of human behavior because the 'probability-of-choice' curve produced by the hybrid algorithms had the best fit to the curve produced by the experimental data. In summary, this work highlights how different algorithms and computational mechanisms can produce distinct quantitative and qualitative differences in behavior that can be exemplified through psychometric curves.

In [48], the authors trained RNNs to compute a hierarchical inference task and compared the RNNs learned computations to a *normative Bayesian baseline*. In the hierarchical inference task, RNNs were tasked with classifying the *preferred signal* between two presented noisy signals, where the preferred signal was sampled from a Gaussian distribution with a higher mean than the Gaussian distribution associated with the non-preferred signal. From the two possible signals, termed 'left" and "right", the noisy signal was determined by the task through randomly sampling from a *block*. A block is a consecutive group of trials in which one signal, left or right, is more often the preferred signal. The task involved hierarchical inference because the agent must not only infer which signal is the preferred signal, but must also infer which type of block, *left-preferred* or *right-preferred*, is associated with a given trial. Given that the task involved inference, the authors are able to formulate the optimal algorithm with Bayesian inference. The authors were then able to compare the performance of trained RNNs to the performance of a normative Bayesian baseline through psychometric curves. These psychometric curves quantified how the contrast between the preferred and non-preferred signals affect a model's classification accuracy. The authors showed how trained RNNs are able to optimally compute the task because their psychometric curves reproduce the psychometric curves associated with the normative Bayesian baseline. In summary, this work is an excellent example as to how an computations of an RNN can be evaluated through psychometric comparisons with other algorithms and models.

We find these works, [63] and [48], to be motivating examples of how psychometric experiments can provide computational insights into cognitive agents and neural systems. In our psychometric experiments, we drew heavy inspiration from the methods and techniques outlined in both of these works.

5.2.2 Experiment classifying ambiguous stimuli

Our psychometric experiment to behaviorally distinguish our dynamical mechanisms involved parametrically mixing the stimuli involved in our modular arithmetic task. In **Section** 3.3, we described how input values were embedded within a 100-dimensional vector. Consider the embedding vectors being denoted as $\vec{e}_g, \vec{e}_p, \vec{e}_r$ with $\vec{e}_g$ being the embedding vector for the *green* input value, $\vec{e}_p$ for *purple*, and $\vec{e}_r$ for *red*. To change the contrast between different input values, we can 'mix' embedding vectors to create a (n)*ew embedding vector*:

$$\begin{aligned} \vec{n}_g(\beta) &= \beta\vec{e}_p + (1-\beta)\vec{e}_g \\ \vec{n}_p(\beta) &= \beta\vec{e}_r + (1-\beta)\vec{e}_p \\ \vec{n}_r(\beta) &= \beta\vec{e}_g + (1-\beta)\vec{e}_r \end{aligned} \tag{5.3}$$

By changing β, we parametrically change the contrast between different input values, and thus, change the 'difficulty' of our modular arithmetic task.

In **Figure** 5.2, we depict how the false positive rate (FPR) changes for various CT-RNNs as the β parameter is systematically varied from $[0-1]$. We defined the FPR as follows:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{5.4}$$

In (5.4), "FP" refers to the number of invalid congruence relations incorrectly classified as valid congruence relations and "TN" refers to the number of invalid congruence relations correctly classified as invalid congruence relations. For each time constant τ, we trained 10 CT-RNNs with non-mixed embedding vectors $\vec{e}_g, \vec{e}_p, \vec{e}_r$ and evaluated their FPR with mixed embedding vectors $\vec{n}_g(\beta), \vec{n}_p(\beta), \vec{n}_r(\beta)$ for the set $\{\beta \mid \beta = 0.0, 0.1, 0.2, \ldots, 1.0\}$. The curves in **Figure** 5.2 plot the average FPR as a function of β with error bars corresponding to the standard error of the mean (SEM).

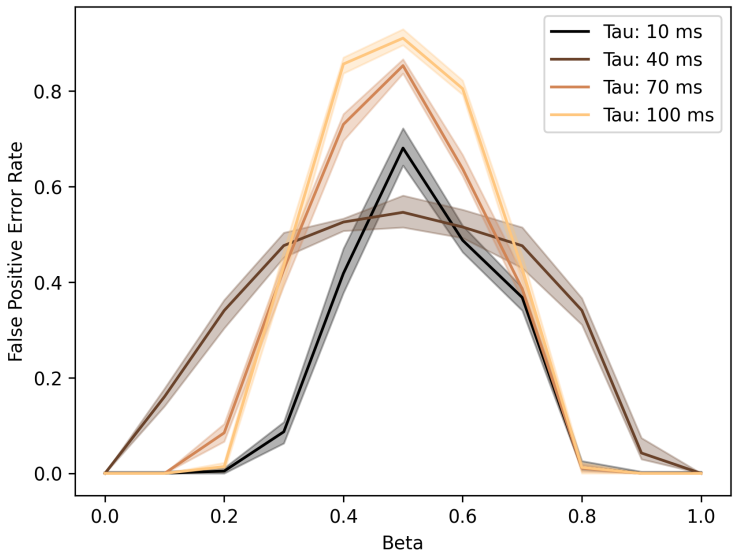


Figure 5.2: Psychometric signatures for various time constants

In **Figure** 5.2, it is apparent that CT-RNNs trained with a particular τ produce a distinct psychometric signature. As noted in **Section** 4.2, CT-RNNs with $\tau = 100$ms learn the attractive mechanism and CT-RNNs with $\tau = 10$ms learn the oscillatory mechanism. We speculate these psychometric signatures could be used to identify the implemented dynamical mechanism in neural systems computing modular arithmetic. Furthermore, we also speculate that these psychometric signatures could be used to infer the timescales of neural systems computing modular arithmetic.

5.2.3 Hierarchy of time constants

Given that different time constants τ result in distinct psychometric signatures (**Fig.** 5.2), a mammal exemplifying a specific psychometric signature could have the associated time constant τ in the cortical area responsible for computing the task. While it is unclear if the time constant τ of (3.4) translates to the timescales of cortical areas, we hypothesize that a

group of tasks activating a group of distinct cortical areas could, in principle, elicit associated psychometric signatures that effectively 'tag' each cortical area with a time constant τ. These cortical areas could then be ordered based on the magnitude of their 'tagged' time constants. Previous evidence has indicated that a hierarchy of timescales exists in the cortex and this hierarchy of timescales reflects a computational hierarchy of cortical areas [64]. We speculate that psychometric signatures could be used to establish a hierarchy of cortical areas in mammals where collecting neurophysiological data is infeasible.

5.3 Grokking curves

Our second experiment detailing how behavioral phenomena can address question 6 sought to quantify how the attractive and oscillatory mechanisms generalize to unseen congruence relations. Our motivation for this experiment originates from **Section** 4.3.2 where we show that the oscillatory mechanism FSM has a transition function δ with fewer states and transitions than the attractive mechanism FSM's transition function. In principle, the oscillatory mechanism FSM would need to see far fewer examples of congruence relations to learn all of its state-action transitions than the attractive mechanism FSM. Therefore, we hypothesized that the oscillatory mechanism would generalize more frequently to unseen congruence relations than the attractive mechanism.

To test this hypothesis, we trained 16,128 CT-RNNs[1] with many varieties of congruence relations excluded from the training set. We quantify the rates at which the dynamical mechanisms generalize to the excluded congruence relations through plotting the percentage of congruence relations excluded from the training set against the percentage of CT-RNNs, implementing a particular dynamical mechanism, achieving above a 95% classification accuracy on the congruence relations excluded. We term the resulting curves "grokking curves" in reference to the grokking phenomenon mentioned in **Section** 3.2.2. These grokking curves are behavioral phenomena distinct to each dynamical mechanism and could be used to identify dynamical mechanisms in neural systems, thus, further addressing question 6.

We quantify a variety of grokking curves associated with each dynamical mechanism. To quantify these curves, we conducted three sub-experiments: the first sub-experiment excluded both valid and invalid congruence relations from the training set (**Sec.** 5.3.2), the second sub-experiment excluded only invalid congruence relations (**Sec.** 5.3.3), and the third sub-experiment excluded only valid congruence relations (**Sec.** 5.3.4).

In the context of addressing our hypothesis that the oscillatory mechanism would generalize better than the attractive mechanism, our results were inconclusive (**Sec.** 5.3.5). We did not observe that that oscillatory mechanism exhibited greater generalization in all sub-experiments; however, there might be a trade-off between the mechanisms' abilities to generalize that are dependent on the difficulty of generalization. The attractive mechanism appeared to generalize more frequently in training regimes with many examples and the oscillatory mechanism appeared to generalize more frequently in training regimes with few examples.

[1] To our knowledge, while previous studies have trained multiple RNNs [12], no previous study has trained a quantity of RNNs of this magnitude.

5.3.1 Generalization literature

Before describing our generalization experiments and associated results, we review various definitions of the term "grok" and a previous example of generalization in the 'computation-through-dynamics' framework.

As mentioned in **Section** 3.2.2, the term "grok" refers to a specific phenomenon in the field of deep learning [25], [26]; however, "grok" has a variety of definitions:

1. To understand at a deep, intuitive level [65].
2. To spontaneously achieve near perfect accuracy on a validation set many epochs later than achieving near perfect accuracy on a training set [25], [26].
3. To identify the underlying pattern from a few examples.

Definition 1 refers to the original meaning of "grok" from Robert A. Heinlein's 1961 book *Stranger in a Strange Land* [65] and definition 2 refers to the deep learning phenomenon depicted in **Figure** 3.4. For this experiment, we use definition 3 to refer to CT-RNNs *grokking* by achieving $>95\%$ classification accuracy on the congruence relations excluded from the training set. In a sense, the meaning of definition 3 lies somewhere between the meanings of definition 1 and definition 2[2].

Along with this experiment's investigation of generalization in CT-RNNs, an example of generalization in CT-RNNs can be found in [14], which was reviewed in **Chapter** 2. In **Section** 2.4, we noted how CT-RNNs trained on the transitive inference task in [14] were trained only on adjacently-ordered symbol pairs and generalized to accurately compute transitive inference for all symbol pairs. Thus, using definition 3, the CT-RNNs in [14] can be described as *grokking* transitive inference. In this experiment, our CT-RNNs can be described as *grokking* modular arithmetic.

5.3.2 Experiment excluding all congruence relations

For this sub-experiment, we trained CT-RNNs with varying percentages of **all congruence relations excluded**[3]. This sub-experiment was conducted according to the following procedure:

1. Out of 27 possible congruence relations, both valid and invalid, p percent of congruence relations were uniformly selected to be excluded from the **Training Set**.
2. A **Training Set** of non-excluded congruence relations was created.
    - Each congruence relation had at least 50 associated trials.
    - Additional trials were created to balance the number of valid and invalid congruence relation trials.

[2]Refer to **Appendix** C for a brief overview of our work describing CT-RNNs *grokking* as defined in definition 1.

[3]We emphasize the phrase "all congruence relations excluded" because the other sub-experiments either excluded invalid congruence relations (**Sec.** 5.3.3) or valid congruence relations (**Sec.** 5.3.4).

3. A **Testing Set** of excluded congruence relations was created.
    - Each congruence relation had exactly 50 associated trials.
4. 192 CT-RNNs were trained with $\tau = 100$ms to induce the attractive mechanism (**Sec.** 4.2).
    - For each CT-RNN trained, new embedding vectors for each input value were created.
    - Each CT-RNN was trained for 5000 epochs with mini-batches of 256 trials.
5. The 192 trained CT-RNNs, with $\tau = 100$ms, were tested.
    - A CT-RNN *grokked* if it achieved above a 95% classification accuracy on the **Testing Set**.
    - The number of CT-RNNs that *grokked* was divided by 192. This percentage, denoted as a, indicated the probability for CT-RNNs implementing the attractive mechanism to successfully *grok* modular arithmetic.
6. 192 CT-RNNs were trained with $\tau = 10$ms to induce the oscillatory mechanism (**Sec.** 4.2).
    - The same training conditions as step 4 were used.
7. The 192 trained CT-RNNs, with $\tau = 10$ms, were tested using the **Testing Set**.
    - The same *grokking* criterion as step 5 was used.
    - The number of CT-RNNs that *grokked* was divided by 192. This percentage, denoted as o, indicated the probability for CT-RNNs implementing the oscillatory mechanism to successfully *grok* modular arithmetic.
8. Steps 1-7 were repeated for new percentages p.
    - The experiment was concluded when all percentages p had been evaluated.

Through comparing the probability for attractive mechanisms to *grok*, a, with the probability for oscillatory mechanisms to *grok*, o, for a variety of percentages of all congruence relations excluded, p, we can evaluate which dynamical mechanism generalized better in this sub-experiment. In essence, 'grokking curves' plot a and o against p.

Figure 5.3 depicts the results of this sub-experiment. While our hypothesis predicted for the oscillatory mechanism's grokking curve to be higher than the attractive mechanism's grokking curve, on average, the grokking curve for the attractive mechanism was higher. Until 40% of all congruence relations excluded, the attractive mechanism *groks* with higher probability than the oscillatory mechanism. After 40%, both mechanisms *grok* with similar probabilities.

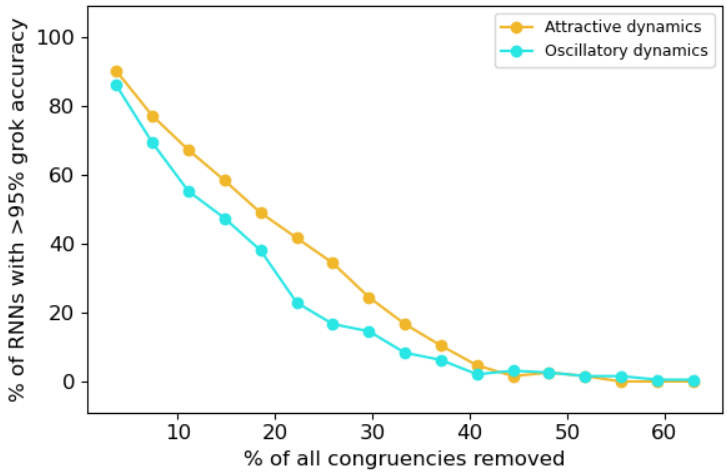


Figure 5.3: Grokking curves when excluding all congruence relations

5.3.3 Experiment excluding invalid congruence relations

For this sub-experiment, we trained CT-RNNs with varying percentages of **invalid congruence relations excluded**. The experimental procedure was identical to the procedure used in **Section** 5.3.2, but with step 1 changed to the following:

1. Out of 18 possible invalid congruence relations, p percent were uniformly selected to be excluded from the **Training Set**.

Unlike **Section** 5.3.2 where the attractive mechanism exhibited, on average, a higher probability of *grokking*, a more balanced trade-off between the two mechanisms was exhibited in this sub-experiment (**Fig.** 5.4). Until 60% of invalid congruence relations excluded, the attractive mechanism *groks* with higher probability than the oscillatory mechanism. After 60%, the oscillatory mechanism *groks* with higher probability. The oscillatory mechanism's higher probability of *grokking* in 'low-data training regimes', training datasets with few congruence relation examples, might be indicative of the oscillatory mechanism's aptitude for *grokking*, and thus, might support our hypothesis.

5.3.4 Experiment excluding valid congruence relations

For this sub-experiment, we trained CT-RNNs with varying percentages of **valid congruence relations excluded**. The experimental procedure was identical to the procedure used in **Section** 5.3.2, but with step 1 changed to the following:

1. Out of 9 possible valid congruence relations, p percent were uniformly selected to be excluded from the **Training Set**.

In this sub-experiment, we observed that the oscillatory mechanism's grokking curve was consistently higher than the attractive mechanism's grokking curve (**Fig.** 5.5). While we expected the oscillatory mechanism to exhibit a higher probability of *grokking* across all experimental conditions, it is reassuring how the oscillatory mechanism outperforms the

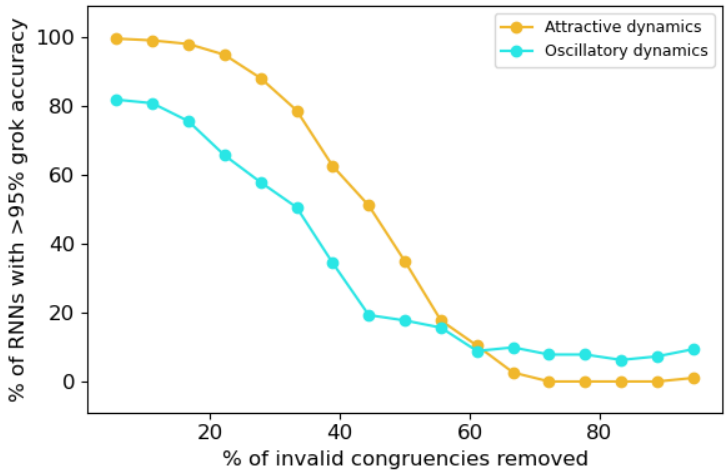


Figure 5.4: Grokking curves when excluding invalid congruence relations

attractive mechanism in this sub-experiment given how there are fewer valid congruence relations than invalid congruence relations. From the perspective of FSMs, the oscillatory mechanism, in principle, only needs to observe at least one valid congruence relation to acquire its valid congruence state, q_0, while the attractive mechanism needs to observe at least 4 valid congruence relations to acquire its valid states, q_0, q_9, q_{13}, q_{17} (**Sec.** 4.10a).

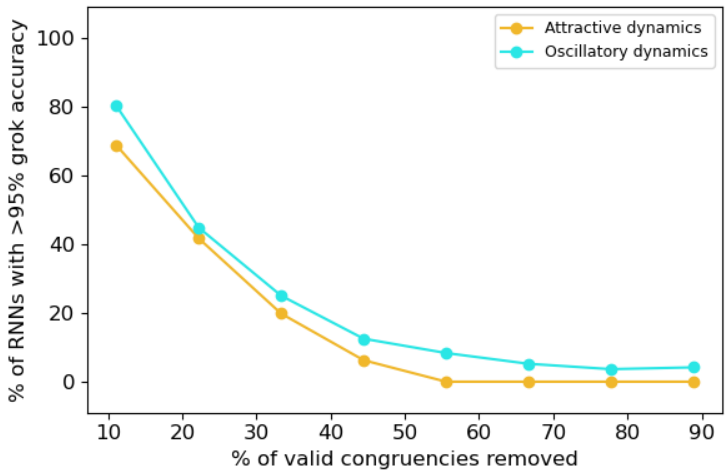


Figure 5.5: Grokking curves when excluding valid congruence relations

5.3.5 Grokking as a benchmark

Our results from the experiments in **Sections** 5.3.2, 5.3.3, and 5.3.4 can be summarized as follows:

1. When excluding both valid and invalid congruence relations, the attractive mechanism generalizes more frequently (**Sec.** 5.3.2).

2. When excluding only invalid congruence relations, the attractive mechanism generalizes more frequently with many training examples and the oscillatory mechanism generalizes more frequently with few training examples (**Sec.** 5.3.3).

3. When excluding only valid congruence relations, the oscillatory mechanism generalizes more frequently (**Sec.** 5.3.4).

To confirm our hypothesis that the oscillatory mechanism generalizes better than the attractive mechanism, we should have seen results 1 through 3 describe the oscillatory mechanism generalizing more frequently than the attractive mechanism; however, this was not the case. Yet, these results do not imply that our hypothesis is incorrect. Notice how result 1 involved excluding up to 27 congruence relations, result 2 involved excluding up to 18 congruence relations, and result 3 involved excluding up to 9 congruence relations. Therefore, as the pool of congruence relations becomes smaller, and thus the task of generalizing to unseen congruence relations becomes more difficult, the oscillatory mechanism generalizes better than the attractive mechanism. It might be the case that the trade-off observed in **Section** 5.3.3 applies to all generalization experiments; the attractive mechanism might generalize more frequently when generalization is easy and the oscillatory mechanism might generalize more frequently when generalization is difficult.

Given that our hypothesis from **Section** 4.3.2 was not conclusively accepted or rejected, one potential area for improvement in our experiments is to test different weight initializations in our trained CT-RNNs. For these experiments, as described in **Section** 3.4.1, we used a Glorot uniform initializer with the scale set to 1.0 [31]. Yet, previous work has shown how specific weight initializers are more conducive for the emergence of oscillatory dynamics in RNNs [46]. Therefore, it stands to reason that there are weight initializers that might result in higher *grokking* probabilities for the oscillatory mechanism, and thus, might better support our hypothesis. Even furthermore, grokking curves could be used as a means to robustly benchmark the performance of different weight initializers, allowing for a more nuanced comparison of RNN architectures than simply comparing testing accuracies.

In summary, these experiments produced a variety of grokking curves that could be used to identify one of our dynamical mechanisms in a neural system. For example, if a neural system is able to frequently *grok* modular arithmetic with few examples of valid congruence relations, then it is plausible, given the results of **Section** 5.3.4, that the neural system has implemented the oscillatory mechanism. While this example is rudimentary, it shows how grokking curves are behavioral phenomena and outlines how they may be used by experimental neuroscientists to address question 6 (**Ch.** 1).

5.4 Addressing question 7

As described in **Section** 5.2 and **Section** 5.3, behavioral phenomena could be used to experimentally identify dynamical mechanisms implemented in neural systems computing modular arithmetic; but moreover, we speculate that behavioral phenomena could be further used to explain why a particular dynamical mechanism was implemented. Such an explanation would address question 7 from **Chapter** 1:

7. What criteria determine the implementation of one dynamical system versus another?

For example, **Figure** 5.2 shows how the attractive mechanism ($\tau = 100$ms) consistently exhibits a higher FPR than the oscillatory mechanism ($\tau = 10$ms) when computing modular arithmetic for ambiguous stimuli. In the context of an ecological setting, there might be an advantage for making false positive classification errors more frequently than false negative errors. For example, if a biological agent were classifying mushrooms as poisonous or not poisonous, it follows that the agent would increase their odds of survival by making false positive errors in classifying mushrooms as poisonous given how the consequences of a misclassification are severe.

Figure 5.4 is particularly indicative of how behavioral phenomena can explain the implementation of dynamical mechanisms. **Figure** 5.4 indicated the existence of a trade-off in *grokking* probabilities between the dynamical mechanisms: the attractive mechanism is more likely to *grok* with high numbers of examples, while the oscillatory mechanism is more likely to *grok* with low numbers of examples. In an ecological context, a biological agent attempting to discern a pattern might change its dynamical mechanism responsible for pattern identification depending on the prevalence of examples in its environment. For example, a bee attempting to identify flowers with pollen might implement a particular mechanism in environments with many example flowers and a different mechanism in environments with few example flowers.

While the examples given in the previous two paragraphs are speculative, we argue that they suggest how behavioral phenomena can provide explanations as to why particular dynamical mechanisms are implemented, thus addressing question 7. Future experimental work expanding on this argument might compare the dynamical mechanisms implemented by a biological agent in a variety of parametrically-varied environments. If a particular environmental variable reliably influences the dynamical mechanisms implemented, then it could be argued that the particular variable acts as a *criterion* in determining the dynamical implementation. Furthermore, future experimental work might compare the dynamical mechanisms implemented by a variety of different species for a particular task. Just as there are *evolutionary principles* guiding neuronal biophysics [66], there might be *evolutionary criteria* guiding the implementation of dynamical mechanisms.

Chapter 6

Conclusion

Throughout this thesis, we have established how CT-RNNs can learn two distinct dynamical mechanisms on our modular arithmetic task and how behavioral phenomena can be used to experimentally identify these mechanisms. These efforts sought to address questions 6 and 7 aimed at the efficacy of the 'computation-through-dynamics' framework raised in **Chapter** 1. In this final chapter, we summarize our work (**Sec.** 6.1), discuss our findings in the context of our 'levels of neural computation' (**Sec.** 6.2), and identify avenues for future work (**Sec.** 6.3).

6.1 Summary

In **Chapter** 1, we began our work by explaining the 'computation-through-dynamics' framework and identifying a variety of challenges for the framework. We framed these challenges in the form of seven questions and specified how this thesis addresses questions 6 and 7. Notably, these questions challenged the notion that a formal mapping could be defined from computations to dynamical systems.

In **Chapter** 2, we explained questions 1-5 from **Chapter** 1 in the context of previous 'computation-through-dynamics' works [11]–[14]. These works also provided insight as to the types of dynamical systems learned by neural systems to perform behaviorally-relevant computations.

In **Chapter** 3, we formulated our task by explaining the card game SET and its connections to modular arithmetic. Our modular arithmetic task was characterized by presenting stimuli to an agent throughout the trial at times drawn from a uniform distribution and tasking the agent to produce a classification signal at the end of the trial. We also specified the parameters of the CT-RNNs trained on the task.

In **Chapter** 4, we articulated the two dynamical mechanisms, the attractive and oscillatory mechanisms, that CT-RNNs can learn to compute the task. The attractive mechanism was characterized by a grid of fixed-point attractors where input values would cause the network's activity to transition among the attractors. We described an abstracted model of the attractive mechanism, termed the "vector abstraction", that described the computational operations of the mechanism as vector addition with a feature map. The oscillatory mechanism was characterized by a limit cycle where input values would cause a phase-shift in the limit cycle. We described an abstracted model of the oscillatory mechanism, termed

the "sinusoidal abstraction", where the mechanism's limit cycle was modeled as a sinusoidal oscillation with input values being integrated in the phase-shift term of the sinusoid. At the end of the chapter, we compared the two mechanisms through the framework of finite-state machines (FSMs) and showed that the oscillatory mechanism contained fewer states.

In **Chapter** 5, we performed behavioral experiments on our dynamical mechanisms to demonstrate how behavioral phenomena can address questions 6 and 7 for the 'computation-through-dynamics' framework. Our first experiment showed how the dynamical mechanisms produce distinct psychometric curves, termed "psychometric signatures", when classifying ambiguous stimuli. Our second experiment showed how the dynamical mechanisms generalize at different rates to unseen stimuli when trained on datasets of limited modular arithmetic examples. We characterized the rate at which dynamical mechanisms generalize through behavioral phenomena termed "grokking curves". These two experiments and their corresponding behavioral phenomena exemplify how behavior can be used to experimentally identify distinct dynamical systems performing the same computation, thus addressing question 6. To address question 7, we speculate how these behavioral phenomena could be framed in an ecological context to serve as criteria that might be used to determine the implementation of one dynamical mechanism versus the other.

6.2 Levels of abstraction

A missing component of the 'computation-through-dynamics' framework is a formal method for reasoning about how dynamical mechanisms implement computational goals and how computational processes can be regarded as abstractions of dynamical mechanisms. In the field of computer science, levels of abstraction are crucial in specifying how computers execute programs; yet, such levels do not exist in the 'computation-through-dynamics' framework. Given its own levels of abstraction and a formal method of mapping between these levels, the 'computation-through-dynamics' framework might better address questions 1-7 posed in **Chapter** 1 and allow for more fruitful investigations into how cognition arises from neurological processes.

In this section, we begin by reviewing and critiquing *Marr's levels of analysis* [9], a common analysis framework in cognitive science and neuroscience (**Sec.** 6.2.1). Next, we review a formulation of levels of abstraction for digital computers (**Sec.** 6.2.2). Then, we formulate a levels of abstraction for the 'computation-through-dynamics' framework, termed "levels of neural computation", that seek to specify how computational goals and dynamical systems can be formally related (**Sec.** 6.2.3). We exemplify the utility of our 'levels of neural computation' through describing our work in this thesis. Finally, we reframe the seven questions for the 'computation-through-dynamics' framework in terms of our levels of abstraction to show how these questions become more precise and productive (**Sec.** 6.2.4).

6.2.1 Marr's levels of analysis

Proposed by cognitive scientist David Marr in 1982, Marr's levels of analysis seek to explain complex systems on three distinct levels:

1. Computational theory: What is the goal of the system?

2. Representation and algorithm: What is the algorithm that performs the computation? What representations does the algorithm manipulate?

3. Hardware implementation: How is the algorithm physically instantiated?

By describing a complex system at all three levels, a more comprehensive understanding is achieved than by focusing on just one level alone.

To illustrate the effectiveness of Marr's levels, let us describe the visual cortex [9]. At the 'computational theory' level, the goal of the visual cortex can be thought of as identifying objects in a mammal's environment. At the 'representation and algorithm' level, visual stimuli are hierarchically represented with primitive representations being basic properties of shapes and the underlying algorithm hierarchically combines these representations to eventually create and identify objects. At the 'hardware implementation' level, the visual cortex implements these hierarchical representations of objects through the hierarchical organization of its five subcortical areas: V1 through V4 and the inferior temporal cortex. If one explains the visual cortex at only one level, it is evident that crucial details are missing in order to fully comprehending its purpose or function. However, all three levels work together to create a complete description where the visual cortex's purpose and function are transparent.

For the 'computation-through-dynamics' framework, it may seem tempting to frame the connection between computations and dynamical systems with Marr's levels. Specifically, dynamical systems appear to be situated at the 'representation and algorithm' level. However, we find this interpretation problematic because in the context of our dynamical mechanisms, where would the vector abstraction (**Sec.** 4.1.1) and sinusoidal abstraction (**Sec.** 4.2.2) be categorized? It is unclear if the algorithm implemented is the dynamical mechanism or the abstraction. This issue could also apparent with the context-dependent computation task (**Sec.** 2.2.1) and the dynamical systems, (5.1) and (5.2), described in **Section** 5.1. For these reasons, we were motivated to formulate more complete levels of abstraction for the 'computation-through-dynamics' framework where dynamical systems serve as implementations of geometric abstractions.

6.2.2 Abstractions in digital computation

Perhaps the most important property of digital computation is its use of abstractions, which enable programmers to develop software while engineers concurrently optimize hardware, creating a continuous cycle of use and improvement. To guide our formulation of new levels of abstraction for the 'computation-through-dynamics' framework, we describe levels of abstraction for digital computers as proposed in [10]. These levels are described as the following:

1. Intention: What task does the user intend for the computer to perform?

2. Specification: What are the expected types of inputs and outputs of the program? Which inputs map to which outputs? How much time should the program take to complete and how much memory should be allocated?

3. Algorithm: What procedure executes the task?

4. High-level program: How do the instructions written in a high-level programming language implement the algorithm?

5. Assembly code: How do low-level manipulation of bits implement the high-level program?

6. Execution: How do the physical processes of the computer execute the assembly code?

In the case of our programmer and engineer example, a programmer operates on levels 1 through 4 while an engineer operates on levels 4 through 6.

It is worth noting that these levels of abstraction for digital computation are the result of simplifications made in [10]. For example, at the 'algorithm' level, algorithms themselves can be described on many distinct levels of abstraction. The purpose of these levels is to describe the salient features of digital computation.

6.2.3 Abstractions in neural computation

With the issues of Marr's levels of analysis identified (**Sec.** 6.2.1) and digital computation's levels of abstraction described (**Sec.** 6.2.2), we can now formulate our levels of abstraction for neural computation. These levels are as follows:

1. Task: What task is the neural system performing?

2. Mathematical specification: How are task inputs and outputs mathematically related?

3. Geometric representation and manipulation: How are task-relevant variables, as described in the 'mathematical specification' level, geometrically represented and manipulated?

4. Dynamical system: How are the geometric representations and manipulations implemented in a dynamical system?

5. Neuronal circuit: How does the neural substrate of the system instantiate the dynamical system?

These levels of abstraction, termed "levels of neural computation", are essentially adapted from the levels of abstraction for digital computation [10] where the 'intention' level is equivalent to the 'task' level, 'specification' is equivalent to 'mathematical specification', 'algorithm' is equivalent to 'geometric representation and manipulation', 'high-level program' is equivalent to 'dynamical system', and the 'assembly code' and 'execution' levels are equivalent to the 'neuronal circuit' level.

We argue that the most novel level from our 'levels of neural computation' is the 'geometric representation and manipulation' level. The introduction of this level resolves our issue with Marr's levels, identified in **Section** 6.2.1, by moving dynamical systems, such as the attractive mechanism, to an implementation level, and placing geometric representations, like the vector abstraction, at the algorithmic level. As discussed in [67], the field of neuroscience is often nonspecific about what constitutes a neural algorithm. [67] even goes as far as to ask, "Are we even positioned to understand what a neural algorithm even is?" With

the 'levels of neural computation', we argue that models of geometric representation and manipulation, like the vector abstraction and sinusoidal abstraction, are neural algorithms.

For an example of the 'levels of neural computation', let us describe our dynamical mechanisms. At the 'task level', our dynamical mechanisms perform our modular arithmetic task. Note that while we refer to the task as the "modular arithmetic task", the task is more representative of the *Rule of SET* than modular arithmetic (**Sec.** 3.2.1). At the 'mathematical specification' level, the dynamical mechanisms are specified with finite-state machines (FSMs); these FSM specifications formally relate the inputs of the task to the desired classifications. If we were have to defined our task where vector embeddings were resampled for each trial in a task, instead of being fixed across all trails in a task, then it might have been the case that a Bayesian inference mathematical specification could have been selected, as in [48]. At the 'geometric representation and manipulation' level, the attractive mechanism is described with the vector abstraction and the oscillatory mechanism is described with the sinusoidal abstraction. This level is crucial because it provides constraints as to what FSMs could have been realized; in particular, stimuli being represented as fixed-length vectors drastically reduce the possible FSMs that could have been implemented. At the 'dynamical system' level, the dynamical mechanisms are directly represented and implement the geometrical abstractions from the previous level. And finally, at the 'neuronal circuit' level, the dynamical mechanisms are implemented in the weights of the CT-RNN with the attractive mechanism having a time constant of $\tau = 100$ms and the oscillatory mechanism having a time constant $\tau = 10$ms.

6.2.4 Reframing questions from Chapter 1

In this section, we reframe questions for the 'computation-through-dynamics' framework in terms of our 'levels of neural computation'.

1. Are neural dynamics low-dimensional?

While question 1 refers to the dimensionality of the dynamical system implemented by a neural system, through our levels of analysis, the question is better posed as "What is the dimensionality of the geometric representations in the 'geometric representation and manipulation' level?" For example, in **Section** 4.1.2, our extension of the vector abstraction to the multi-attribute task involved four-dimensional representations. By understanding the dimensionality of the geometric abstractions, we can understand how we might reduce the dimensionality for the computational process through optimizing the geometric representations.

2. How do neural substrates implement dynamical systems?

Question 2 can be easily reframed as "What is the formal relationship between level 4 and level 5?" In the context of our work, is there a specific mapping from the connectivity of the weight matrices B and J to the dynamical mechanisms? Moreover, this question can be asked about any two adjacent layers. Perhaps there are a cascade of mappings that traverse

through all levels of abstraction. In the case of the attractive mechanism, the mapping could be as follows with the attractive mechanism denoted as $\dot{x} = f(x)$:

$$B, J \mapsto f(x) \mapsto W \mapsto \delta \tag{6.1}$$

In the case of the oscillatory mechanism, the mapping could be as follows with the oscillatory mechanism denoted as $\dot{x} = g(x)$:

$$B, J \mapsto g(x) \mapsto \phi(t) \mapsto \delta \tag{6.2}$$

One issue with these mappings may be that not all relationships are one-to-one, thus, giving rise to question 6.

3. Do dynamical systems explain causal relationships among task variables in neural systems?

Reframed in the 'levels of neural computation', question 3 becomes similar to question 2 in that they both ask about the formal relationship between different levels. Question 3 can specifically be reframed as "What is the formal relationship between level 3 and level 4?" However, the semantics of question 3 imply that the levels of abstraction are traversed from level 1 to level 5, instead of level 5 to level 1 as described in (6.1) and (6.2). In the context of the attractive mechanism, question 3 asks about how to specify the following relationships:

$$\delta \mapsto W \mapsto f(x) \mapsto B, J \tag{6.3}$$

In the context of the oscillatory mechanism, question 3 asks about how to specify the following relationship:

$$\delta \mapsto \phi(t) \mapsto g(x) \mapsto B, J \tag{6.4}$$

Question 3 and 4 taken together effectively ask if there is a bidirectional mapping between each stage; yet, question 6 still posses an issue for the existence of this mapping as one δ can correspond to two geometric abstractions.

4. How could separate dynamical systems be combined to produce complex behaviors?

While questions 1-3 are significantly changed via reframing, question 4 largely remains the same. In our levels of abstraction, question 4 asks about how geometric primitives, like vector encodings and feature maps, might be implemented in level 4 and 5 in such a way to allow for easy reuse. In the context of our task, if we trained CT-RNNs with modular arithmetic of (mod 3) and of (mod 4), how might geometrically similar features across the two modular bases be implemented in level 4 and 5 in such a way that switching between the two bases is efficient?

5. How could dynamical systems perform symbolic operations?

Question 5 is similar to question 3 in that it asks how levels can be traversed top-down from levels 2 to 5. Specifically, symbolic operations constitute a 'mathematical specification'. Thus, the question is reframed as "What are the corresponding 'geometric representations and manipulations', 'dynamical systems', and 'neuronal circuits' for a 'mathematical specification' of symbolic operations?"

6. How can computationally similar, yet dynamically distinct systems be experimentally identified?

In **Chapter** 5, we argued for the use of behavioral phenomena to identify dynamical mechanisms and address question 6. Essentially, behavioral phenomena are properties at level 5 that give clues about higher-level abstractions. While reframed questions 2 and 3 posited that weight matrices B, J are properties of level 5, so too are behavioral phenomena and neural activity. Thus, question 6 is also about specifying formal relationships between levels, specifically level 5 to level 4. Perhaps there is a method of combining all properties of level 5 to exactly specify the higher-level abstractions in levels 3 and 4.

7. What criteria determine the implementation of one dynamical system versus another?

Just as question 6 is about fully specifying the properties of level 5 in order to map to level 4, question 7 can be reframed as fully specifying the properties of level 1 in order to map to level 4. In **Section** 5.4, we argue how there are ecological criteria that determine implementation details and address question 7. These ecological criteria are properties of the 'task' level. By fully specifying the task in an ecological context, there might be a sufficient number of constraints to allow for a mapping from level 1 through to level 4. Taking question 6 and 7 together, by fully specifying the properties of level 1 and level 5, there might exist a bidirectional mapping between all levels in the 'levels of neural computation'.

6.3 Future directions

We envision four possible directions to extend our work in this thesis. The first direction seeks to establish formal mappings between the different 'levels of neural computation' (**Sec.** 6.3.1). The second direction focuses on collecting behavioral data from biological agents performing our modular arithmetic task in order to validate the association between our dynamical mechanisms and identified behavioral phenomena (**Sec.** 6.3.2). The third direction is to reframe previous works from the 'computation-through-dynamics' framework to our 'levels of neural computation' (**Sec.** 6.3.3). Finally, the fourth direction seeks to use the 'levels of neural computation' to describe how humans implement and execute visual search algorithms to play the card game SET (**Sec.** 6.3.4).

6.3.1 Mapping between levels of abstraction

The most immediate area of future work lies in addressing reframed question 2 in **Section** 6.2.4. Specifically, what is the formal mapping between levels 5, 4, and 3 in the context of our dynamical mechanisms and modular arithmetic task. With level 5 being a CT-RNN and level 4 the dynamical mechanisms, the latent circuit inference method, as described in [12], might provide a sufficient method for mapping between levels 5 and 4. However, the mapping between levels 4 and 3 has not been previously established. Perhaps a starting point for establishing the mapping between dynamical systems and geometric abstractions is to formally relate fixed-point attractors of the attractive mechanism to state-space vectors of the vector abstraction.

A simple approach to creating a formal mapping between the attractive mechanism and the vector abstraction is to create a linear map from the matrix of fixed-point attractors and the matrix of state-space vectors in the vector abstraction. Consider the following matrix of dynamical activity vectors $\mathbf{x}$ from the attractive mechanism $\dot{\mathbf{x}} = f(\mathbf{x})$:

$$X = \begin{bmatrix} | & | & | & \\ \mathbf{x}_0 & \mathbf{x}_1 & \mathbf{x}_2 & \cdots \\ | & | & | & \end{bmatrix}$$

Here, each vector $\mathbf{x}_i$ in X corresponds to a fixed-point attractor $\dot{\mathbf{x}}_i = f(\mathbf{x}_i) = \vec{0}$. Consider the following matrix of feature-mapped, state-space vectors $\Phi(\mathbf{y})$ from the vector abstraction (**Sec.** 4.1.1):

$$Y = \begin{bmatrix} | & | & | & \\ \phi(\mathbf{y}_0) & \phi(\mathbf{y}_1) & \phi(\mathbf{y}_2) & \cdots \\ | & | & | & \end{bmatrix}$$

Here, each vector $\Phi(\mathbf{y}_i)$ in Y corresponds to a unique feature-mapped, state-space vector. Thus, the mapping M between X and Y could simply be the following:

$$MX = Y \tag{6.5}$$
$$M = YX^{-1} \tag{6.6}$$

This map $M : \mathbb{R}^m \to \mathbb{R}^n$ is a matrix transformation from the dynamical activity space $\mathbb{R}^m$ of the attractive mechanism to the feature space $\mathbb{R}^n$ of the vector abstraction where $m \geq n$.

However, a formal mapping between levels 4 and 3 is not as simple as (6.6) because X is not invertible. Recall that trials presenting all different stimuli into the attractive mechanism result in the activity of the mechanism returning to the origin (**Figure** 4.1). Thus, the vectors $\mathbf{x}_g, \mathbf{x}_p, \mathbf{x}_r$, representing fixed-point attractors for trials of all uniform stimuli, should add to the zero vector: $\mathbf{x}_g + \mathbf{x}_p + \mathbf{x}_r = \vec{0}$. Therefore, the null-space of X is not empty and X is not invertible [68]. While we could use the Moore–Penrose inverse to approximate X^{-1}, it is important to preserve the null space of X as most congruence relations containing all different stimuli are computed by mapping to the null space.

Given the complications for establishing a linear map between dynamical mechanisms and geometric abstractions, we leave establishing this mapping as an area for future work. We speculate that the correct mapping will be nonlinear so as to properly preserve the null space of X. Furthermore, given how we effectively hand constructed Y in **Section** 4.1.1, future work could also specify a method for extracting Y from X.

6.3.2 Conducting behavioral experiments

In **Chapter** 5, we outlined a variety of behavioral experiments with the purpose of experimentally identifying our dynamical mechanisms. Performed on CT-RNNs, these experiments elicited a variety of behavioral phenomena distinctly associated with each of our dynamical mechanisms. The next logical step is to perform these experiments on biological agents with associated neurological recordings to assess if behavioral phenomena are actually associated with distinct dynamical systems. For example, by training mice to perform our

modular arithmetic task, experimental neuroscientists could perform the *psychometric signatures* experiment, from **Section** 5.2, while simultaneous recording neural data with implanted tetrodes to assess if our dynamical mechanisms elicit distinct psychometric curves. A potential difficulty might lie in identifying the brain area in mice responsible for modular arithmetic.

Furthermore, future work could establish more behavioral experiments that could be used in identifying dynamical systems and geometric abstractions. Existing psychophysics literature might serve as a fruitful resource in formulating future behavioral experiments and eliciting novel behavioral phenomena [48], [62], [63].

6.3.3 Reframe previous works into our levels of abstraction

Our 'levels of neural computation' were especially useful for framing our work in this thesis; however, these levels could be further applied to other works in the 'computation-through-dynamics' framework. For example, the subtractive dynamical solution that computes transitive inference in [14] could be framed in our 'levels of neural computation'. At the 'task' level, the task is their transitive inference task involving encoded symbols presented at disparate time intervals. At the 'mathematical specification' level, the inputs and outputs of the task are related with the symbolic operation of 'greater than', $>$, which serves to address question 5. At the 'geometrical representation and manipulation' level, inputs are represented with collinear vector encodings, manipulated with a 90^o rotation matrix, and classified through a dot product with a decision boundary. Here is a corresponding 'vector abstraction' for the mechanism in [14]:

$$z = \text{sgn}(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \cdot (\begin{bmatrix} \cos 90^o & -\sin 90^o \\ \sin 90^o & \cos 90^o \end{bmatrix} \vec{x}_1 + \vec{x}_2)) \tag{6.7}$$

In (6.7), $\vec{x}_1$ corresponds to the first symbol presented during a trial, $\vec{x}_2$ corresponds to the second symbol presented, and z corresponds to the classification. Lastly, the 'dynamical system' level corresponds to their identified dynamical mechanism and the 'neuronal circuit' levels corresponds to their trained CT-RNN.

As a starting point, future work could reframe existing work like we did for [14] in the previous paragraph. However, [11]–[13] did not specify the 'mathematical specification' level as specifically as [14] had; therefore, future work should take great care in choosing the 'mathematical specification' for previously established works.

6.3.4 Towards an understanding of neural search algorithms

As noted at the beginning of **Chapter** 3, our original intent for this thesis was to investigate the search algorithms humans implement to play the card game SET and their corresponding neural implementations. However, such an investigation was prohibitively difficult and we pivoted to study the modular arithmetic properties at the core of SET. Now, in light of our theories about behavioral phenomena and our 'levels of neural computation', we believe that studying neural search algorithms for SET is more feasible.

In the context of our 'levels of neural computation', an investigation into the neural search algorithms underlying SET could be carried out in the following process. First, investigators could formulate a variety of algorithms, inspired by existing search algorithms, that successfully play SET. This initial step corresponds to specifying level 1, where the 'task' is SET, and level 2, where the 'mathematical specification' consists of a variety of search algorithms. Next, investigators could simultaneously collect behavioral data from algorithms, humans, and trained CT-RNNs playing SET. Through comparing behavioral phenomena from algorithms, humans, and CT-RNNs, investigators could speculate as to which algorithms humans might implement and what CT-RNN parameters best correspond to the neuronal circuits humans might implement. This step corresponds to specifying properties of level 5. Finally, CT-RNNs that are behaviorally similar to humans and algorithms could be reverse-engineered to formulate the corresponding dynamical systems and geometric abstractions. This final step corresponds to traversing from levels 5 to 2 in a bottom-up fashion.

As a final note, such an investigation would resemble previous investigations into the game of tic-tac-toe [69].

Appendix A

Energy Abstraction

While we articulated how the attractive mechanism computes modular arithmetic through the vector abstraction in **Section** 4.1, the vector abstraction is not the only abstraction that we could have used. In this appendix, we outline the *energy abstraction* inspired by interpretations of the Hopfield network as computing through minimizing an energy function for some given input [70], [71]. In this abstraction, the attractive mechanism can be thought of as minimizing an energy function where the CT-RNN's activity attempts to reach an energy minima; however, the exact minima that the CT-RNN's activity falls into depends on the sequence of inputs it receives. **Figure** A.1 is a depiction of the energy landscape corresponding to the attractive mechanism with colored arrows showing how the CT-RNN's activity would traverse the lattice of basins.

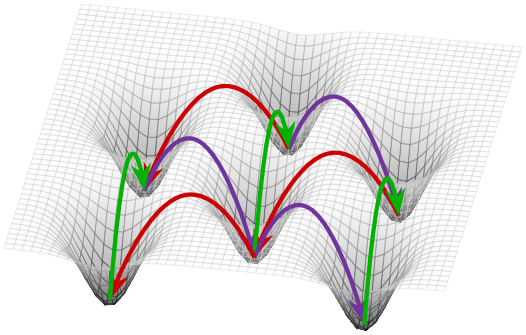
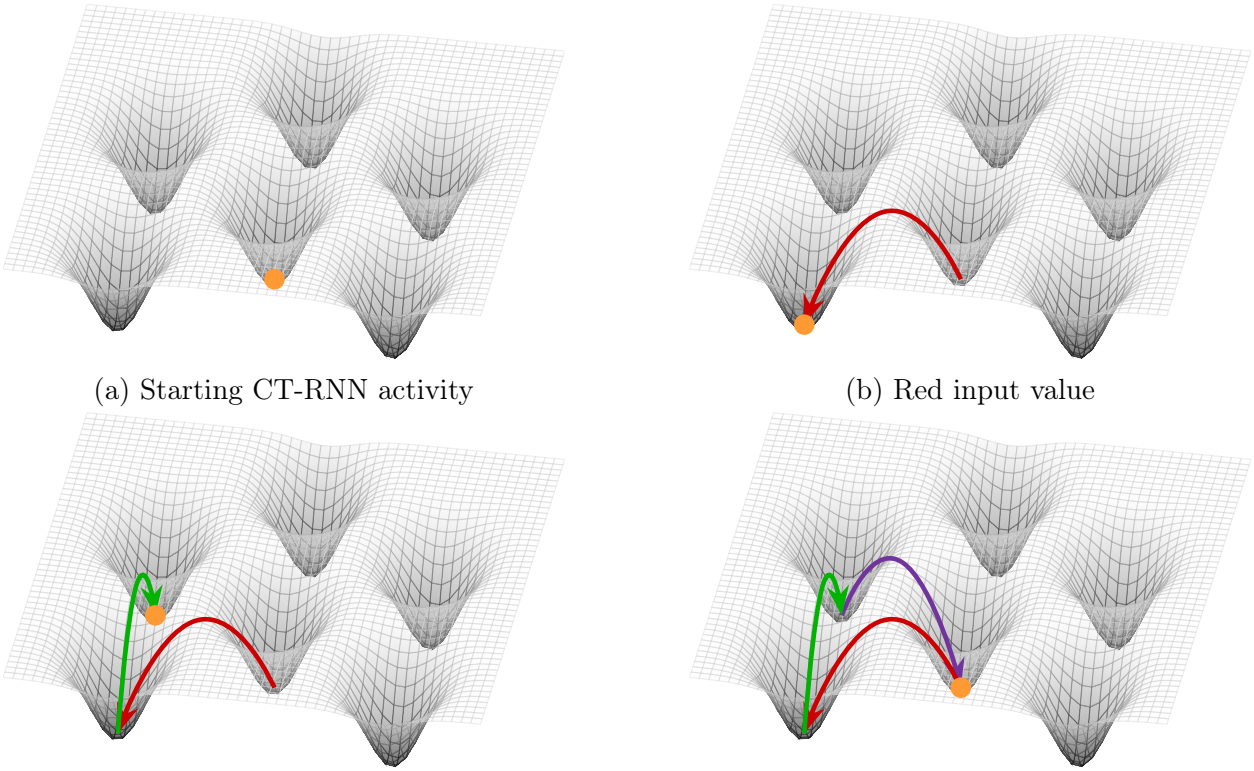


Figure A.1: The energy abstraction of the attractive mechanism

This interpretation is particularly apt for accounting for the distinct time intervals at which input values were presented to the CT-RNN. **Figure** A.2 is an example of how the activity of a CT-RNN would traverse the landscape over the course of a trial. The activity (the orange point) begins in the center of the lattice (**Fig.** A.2a). At some time t_0, the red input value is presented to the CT-RNN and pushes the activity into the lower left basin (**Fig.** A.2b). At some later time t_1, the green input value is presented and pushes the activity into the upper left basin (**Fig.** A.2c). At a final time t_2, the purple input value pushes the activity into the center basin, the starting basin (**Fig.** A.2d). By receiving all different values, the activity of the CT-RNN returns to the starting basin. Conversely, by receiving all the same values, the activity is pushed to a far basin.



(a) Starting CT-RNN activity

(b) Red input value

(c) Green input value

(d) Purple input value

Figure A.2: Example inputs in the attractive mechanism energy landscape

Given that valid and invalid congruence relations fall into distinct basins in the energy landscape, classification becomes simple. **Figure** A.3 shows how the depth of basins corresponding to valid relations could be shallower than the depth of invalid basins; thus, allowing for a hyperplane, the teal plane (**Fig.** A.3), to separate valid and invalid congruence relations. One potential flaw with this interpretation is that it implies that invalid basins are at a lower energy state than valid basins, and, consequently, may be 'more stable'. Besides allowing for a more interpretable classification, there is no a priori reason for invalid basins to be 'more stable' than valid basins.

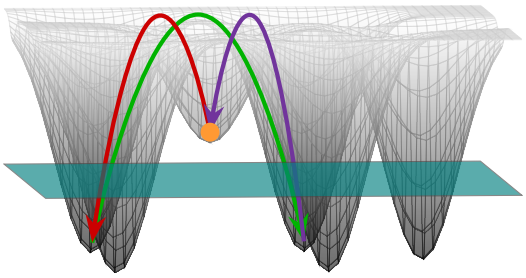


Figure A.3: A classification plane in the energy landscape

While the energy abstraction is intuitive, we chose to focus on the vector abstraction in this thesis because it was more amenable to a formal, mathematical description.

Appendix B

Emergence of a Ring Attractor

In **Section** 4.2.2, we briefly discussed how the *oscillatory mechanism* does not, in principle, need to oscillate to accurately classify congruence relations. Consider the sinusoidal abstraction (4.7) without a frequency:

$$f(t) = \cos\left(\int_0^t \phi(t')dt'\right) \tag{B.1}$$

This abstraction still computes our modular arithmetic task; however, the activity of the model would not vary through time. Following this observation, a reasonable question would be, "Would these dynamics still constitute an oscillatory mechanism?" We argue that a dynamical mechanism emulating (B.1) is still an oscillatory mechanism because the encoding of the stimuli are the most salient feature of the mechanism: the dynamical implementation is incidental.

Of relevant interest, we found one CT-RNN emulating (B.1). **Figure** B.1 depicts example outputs of the CT-RNN and its dynamics in PCA space. The time constant τ was 100ms, consistent with the emergence of other oscillatory mechanisms (**Sec.** 4.2). We are unsure whether the dynamical implementation is a circular line attractor or a series of fixed-point attractors arranged in a circle. In any case, it displays attractor dynamics consistent with ring attractors described in previous works [49], [50].

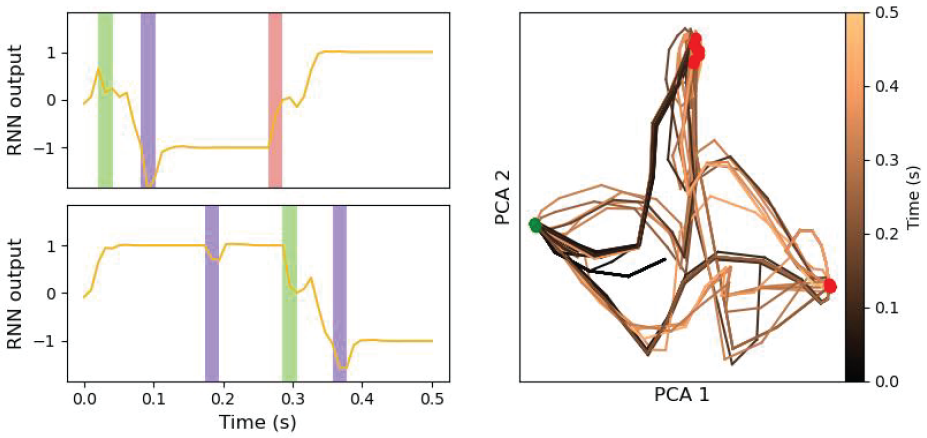


Figure B.1: Example outputs and dynamics of the ring attractor

Appendix C

Can RNNs grok like transformers?

In **Section** 5.3, we investigated how our dynamical mechanisms can *grok* modular arithmetic; however, we used the term "grok" in a manner inconsistent with previous deep learning literature [25], [26]. We did not discuss whether CT-RNNs trained on our modular arithmetic task exhibited a long delay between the model fitting to the training data and generalizing to the testing data, as depicted in **Figure** 3.4. This 'long training delay' has been observed in MLPs and transformers trained on modular arithmetic, causing a variety of theories attempting to explain why this phenomenon occurs [26], [27]. A reasonable question for our work would be, "Do CT-RNNs exhibit a long training delay on our modular arithmetic task?"

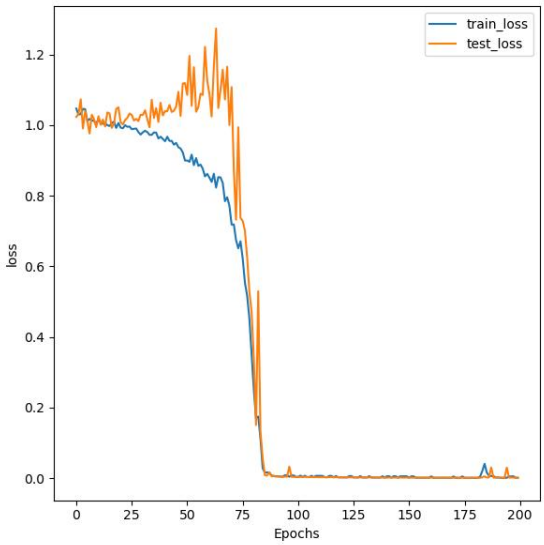
The short is answer is no. A longer answer is that we did not find any evidence of a 'long training delay' when training ∼16,500 CT-RNNs for **Chapter** 5; however, we did find evidence of the 'double-descent' loss phenomenon. This phenomenon is described as a neural network's testing loss traversing through three phases [27]:

1. Initial descent: The neural network beings to learn features relevant for performance, causing the testing loss to decrease.

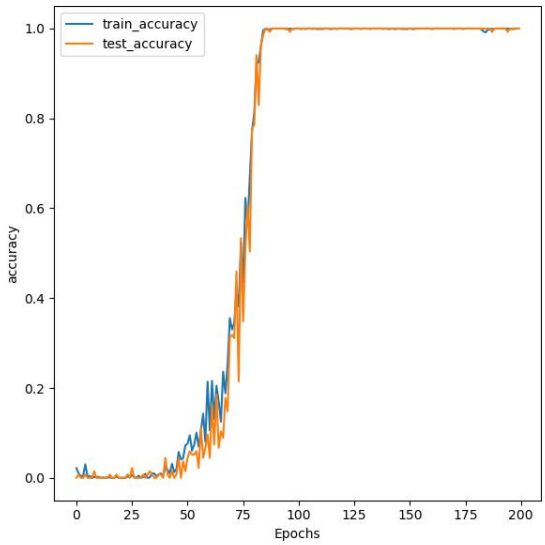
2. Overfitting to training data: The neural network learns features specific only to performance on the training data, causing the testing loss to increase.

3. Generalization: The neural network unlearns training-specific features and prioritizes features general for overall performance, causing the testing loss to rapidly decrease.

Figure C.1a depicts an example of how a CT-RNN trained on our modular arithmetic task will can exhibit this 'double-descent' loss phenomenon; however, it is not associated with a 'long training delay' in testing accuracy (**Figure** C.1b).



(a) Pronounce difference in loss curves



(b) No difference in accuracy curves

Figure C.1: Example of the 'double-descent' loss phenomenon

References

[1] S. Vyas, M. D. Golub, D. Sussillo, and K. V. Shenoy, "Computation through neural population dynamics," *Annual Review of Neuroscience*, vol. 43, pp. 249–275, 2020.

[2] S. H. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, And Engineering*. Boulder, CO: Westview Press, 1994.

[3] T. Van Gelder, "What might cognition be, if not computation?" *The Journal of Philosophy*, vol. 92, no. 7, pp. 345–381, 1995.

[4] T. Van Gelder, "The dynamical hypothesis in cognitive science," *Behavioral and Brain Sciences*, vol. 21, no. 5, pp. 615–628, 1998.

[5] E. Jonas and K. P. Kording, "Could a neuroscientist understand a microprocessor?" *PLoS Computational Biology*, vol. 13, no. 1, e1005268, 2017.

[6] L. McMahon, G. Gordon, H. Gordon, and R. Gordon, *The Joy of SET: The Many Mathematical Dimensions of a Seemingly Simple Card Game*. Princeton, NJ: Princeton University Press, 2017.

[7] A. R. Galgali, M. Sahani, and V. Mante, "Residual dynamics resolves recurrent contributions to neural computation," *Nature Neuroscience*, vol. 26, no. 2, pp. 326–338, 2023.

[8] M. Ostrow, A. Eisen, L. Kozachkov, and I. Fiete, "Beyond geometry: Comparing the temporal structure of computation in neural circuits with dynamical similarity analysis," *arXiv preprint arXiv:2306.10168*, 2023.

[9] D. Marr, *Vision: A computational investigation into the human representation and processing of visual information*. Cambridge, MA: MIT Press, 1982.

[10] G. Primiero, "Information in the philosophy of computer science," in *The Routledge Handbook of Philosophy of Information*, L. Floridi, Ed., Routledge, 2016.

[11] D. Sussillo and O. Barak, "Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks," *Neural Computation*, vol. 25, no. 3, pp. 626–649, 2013.

[12] C. Langdon and T. A. Engel, "Latent circuit inference from heterogeneous neural responses during cognitive tasks," *bioRxiv*, 2022.

[13] L. Driscoll, K. Shenoy, and D. Sussillo, "Flexible multitask computation in recurrent networks utilizes shared dynamical motifs," *bioRxiv*, 2022.

[14] K. Kay, X.-X. Wei, R. Khajeh, M. Beiran, C. J. Cueva, G. Jensen, V. P. Ferrera, and L. Abbott, "Neural dynamics and geometry for transitive inference," *bioRxiv*, 2022.

[15] K. T. Murray, "Recurrent networks recognize patterns with low-dimensional oscillations," *arXiv preprint arXiv:2310.07908*, 2023.

[16] E. Marder and D. Bucher, "Central pattern generators and the control of rhythmic movements," *Current Biology*, vol. 11, no. 23, R986–R996, 2001.

[17] V. Mante, D. Sussillo, K. V. Shenoy, and W. T. Newsome, "Context-dependent computation by recurrent dynamics in prefrontal cortex," *Nature*, vol. 503, no. 7474, pp. 78–84, 2013.

[18] M. Shinn, "Phantom oscillations in principal component analysis," *Proceedings of the National Academy of Sciences*, vol. 120, no. 48, e2311420120, 2023.

[19] D. P. Munoz and R. H. Wurtz, "Role of the rostral superior colliculus in active visual fixation and execution of express saccades," *Journal of Neurophysiology*, vol. 67, no. 4, pp. 1000–1002, 1992.

[20] P. Smolensky, "On the proper treatment of connectionism," *Behavioral and Brain Sciences*, vol. 11, no. 1, pp. 1–23, 1988.

[21] G. Marcus, "Deep learning: A critical appraisal," *arXiv preprint arXiv:1801.00631*, 2018.

[22] H. Jaeger, "From continuous dynamics to symbols," in *Dynamics, synergetics, autonomous agents: Nonlinear systems approaches to cognitive psychology and cognitive science*, World Scientific, 1999, pp. 29–48.

[23] H. Jaeger, "Towards a generalized theory comprising digital, neuromorphic and unconventional computing," *Neuromorphic Computing and Engineering*, vol. 1, no. 1, p. 012 002, 2021.

[24] J. M. Wolfe, "Visual search: How do we find what we are looking for?" *Annual Review of Vision Science*, vol. 6, pp. 539–562, 2020.

[25] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra, "Grokking: Generalization beyond overfitting on small algorithmic datasets," *arXiv preprint arXiv:2201.02177*, 2022.

[26] N. Nanda, L. Chan, T. Lieberum, J. Smith, and J. Steinhardt, "Progress measures for grokking via mechanistic interpretability," *arXiv preprint arXiv:2301.05217*, 2023.

[27] X. Davies, L. Langosco, and D. Krueger, "Unifying grokking and double descent," *arXiv preprint arXiv:2303.06173*, 2023.

[28] D. Sussillo, M. M. Churchland, M. T. Kaufman, and K. V. Shenoy, "A neural network that finds a naturalistic solution for the production of muscle activity," *Nature Neuroscience*, vol. 18, no. 7, pp. 1025–1033, 2015.

[29] C. J. Cueva and X.-X. Wei, "Emergence of grid-like representations by training recurrent neural networks to perform spatial localization," *arXiv preprint arXiv:1803.07770*, 2018.

[30] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[31] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[32] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[33] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman, "Universal differential equations for scientific machine learning," *arXiv preprint arXiv:2001.04385*, 2020.

[34] A. Pal, *Lux: Explicit Parameterization of Deep Neural Networks in Julia*, version v0.5.0, 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7808904.

[35] G. Van Rossum, F. L. Drake, *et al.*, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995, vol. 111.

[36] J. Bradbury, R. Frostig, P. Hawkins, *et al.*, *JAX: Composable transformations of Python+NumPy programs*, version 0.3.13, 2018. [Online]. Available: http://github.com/google/jax.

[37] J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner, and M. van Zee, *Flax: A neural network library and ecosystem for JAX*, version 0.7.4, 2023. [Online]. Available: http://github.com/google/flax.

[38] T. D. Kim, T. Can, and K. Krishnamurthy, "Trainability, expressivity and interpretability in gated neural odes," *arXiv preprint arXiv:2307.06398*, 2023.

[39] A. Reuther, J. Kepner, C. Byun, *et al.*, "Interactive supercomputing on 40,000 cores for machine learning and data analysis," in *2018 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, 2018, pp. 1–6.

[40] G. Piccinini, *Physical Computation: A Mechanistic Account*. Oxford, GB: Oxford University Press UK, 2015.

[41] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY: Springer, 2006.

[42] A. Meurer, C. P. Smith, M. Paprocki, O. Čertik, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, *et al.*, "Sympy: Symbolic computing in python," *PeerJ Computer Science*, vol. 3, e103, 2017.

[43] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[44] M. Khona and I. R. Fiete, "Attractor and integrator networks in the brain," *Nature Reviews Neuroscience*, vol. 23, no. 12, pp. 744–766, 2022.

[45] M. Pals, J. H. Macke, and O. Barak, "Trained recurrent neural networks develop phase-locked limit cycles in a working memory task," *bioRxiv*, pp. 2023–04, 2023.

[46] I. M. Park, Á. Ságodi, and P. A. Sokół, "Persistent learning signals and working memory without continuous attractors," *arXiv preprint arXiv:2308.12585*, 2023.

[47] N. Burgess, C. Barry, and J. O'keefe, "An oscillatory interference model of grid cell firing," *Hippocampus*, vol. 17, no. 9, pp. 801–812, 2007.

[48] R. Schaeffer, M. Khona, L. Meshulam, and H. R. Fiete, "Reverse-engineering recurrent neural network solutions to a hierarchical inference task for mice," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020, pp. 4584–4596.

[49] K. Zhang, "Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: A theory," *Journal of Neuroscience*, vol. 16, no. 6, pp. 2112–2126, 1996.

[50] S. S. Kim, H. Rouault, S. Druckmann, and V. Jayaraman, "Ring attractor dynamics in the drosophila central brain," *Science*, vol. 356, no. 6340, pp. 849–853, 2017.

[51] T. K. Rusch and S. Mishra, "Coupled oscillatory recurrent neural network (cornn): An accurate and (gradient) stable architecture for learning long time dependencies," *arXiv preprint arXiv:2010.00951*, 2020.

[52] C. E. Shannon and J. McCarthy, Eds., *Automata Studies. (AM-34), Volume 34*. Princeton, NJ: Princeton University Press, 1956.

[53] N. Chomsky, "Three models for the description of language," *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 113–124, 1956.

[54] M. Sipser, *Introduction to the Theory of Computation*. Boston, MA: Thomson Course Technology, 1996.

[55] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[56] S. C. Kleene, "Representation of events in nerve nets and finite automata," RAND Corporation, Santa Monica, CA, Tech. Rep. RM-704, 1951.

[57] F. Rosenblatt, "The perceptron: A perceiving and recognizing automaton," Cornell Aeronautical Laboratory, Ithaca, NY, Tech. Rep. 85-460-1, 1957.

[58] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, "Finite state automata and simple recurrent networks," *Neural Computation*, vol. 1, no. 3, pp. 372–381, 1989.

[59] W. Chaisangmongkon, S. K. Swaminathan, D. J. Freedman, and X.-J. Wang, "Computing by robust transience: How the fronto-parietal network performs sequential, category-based decisions," *Neuron*, vol. 93, no. 6, pp. 1504–1517, 2017.

[60] H. Sohn, D. Narain, N. Meirhaeghe, and M. Jazayeri, "Bayesian computation through cortical latent dynamics," *Neuron*, vol. 103, no. 5, pp. 934–947, 2019.

[61] W. J. Ma and B. Peters, "A neural network walks into a lab: Towards using deep nets as models for human behavior," *arXiv preprint arXiv:2005.02181*, 2020.

[62] G. T. Fechner, *Elemente der psychophysik*. Leipzig, DE: Breitkopf u. Härtel, 1860, vol. 2.

[63] S. J. Gershman, "Deconstructing the human algorithms for exploration," *Cognition*, vol. 173, pp. 34–42, 2018.

[64] J. D. Murray, A. Bernacchia, D. J. Freedman, R. Romo, J. D. Wallis, X. Cai, C. Padoa-Schioppa, T. Pasternak, H. Seo, D. Lee, *et al.*, "A hierarchy of intrinsic timescales across primate cortex," *Nature Neuroscience*, vol. 17, no. 12, pp. 1661–1663, 2014.

[65] R. A. Heinlein, *Stranger in a Strange Land*. New York, NY: Putnam Publishing Group, 1961.

[66] L. Beaulieu-Laroche, N. J. Brown, M. Hansen, E. H. Toloza, J. Sharma, Z. M. Williams, M. P. Frosch, G. R. Cosgrove, S. S. Cash, and M. T. Harnett, "Allometric rules for mammalian cortical layer 5 neuron biophysics," *Nature*, vol. 600, no. 7888, pp. 274–278, 2021.

[67] J. B. Aimone and O. Parekh, "The brain's unique take on algorithms," *Nature Communications*, vol. 14, no. 1, p. 4910, 2023.

[68] G. Strang, *Introduction to Linear Algebra*. Wellesley, MA: Wellesley-Cambridge Press, 2016.

[69] B. van Opheusden, I. Kuperwajs, G. Galbiati, Z. Bnaya, Y. Li, and W. J. Ma, "Expertise increases planning depth in human gameplay," *Nature*, pp. 1–6, 2023.

[70] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[71] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons.," *Proceedings of the National Academy of Sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.