# Variable Memory: Beyond the Fixed Memory Assumption in Memory Modeling

**Arjun Karuvally**
College of Information and Computer Sciences
University of Massachusetts, Amherst
Massachusetts, MA 01003
akaruvally@umass.edu

**Hava T. Siegelmann**
College of Information and Computer Sciences
University of Massachusetts, Amherst
Massachusetts, MA 01003
hava@umass.edu

## Abstract

Memory models play a pivotal role in elucidating the mechanisms through which biological and artificial neural networks store and retrieve information. Traditionally, these models assume that memories are pre-determined, fixed before inference, and stored within synaptic interactions. Yet, neural networks can also dynamically store memories available only during inference within their activity. This capacity to bind and manipulate variable information as variables enhances the generalization capabilities of neural networks. Our research introduces and explores the concept of "variable memories." This approach extends the conventional sequence memory models, enabling information binding directly in network activity. By adopting this novel memory perspective, we unveil the underlying computational processes in the learned weights of RNNs on simple algorithmic tasks – a fundamental question in the mechanistic understanding of neural networks. Our results underscore the imperative to evolve memory models beyond the fixed memory assumption towards more dynamic and flexible memory systems to further our understanding of neural information processing.

## 1 Introduction

Associative memory models play a critical role in understanding modern neural network architectures by unraveling their complex computational mechanisms. For instance, they have shed light on how different activation functions affect the behavior of neural networks (Krotov and Hopfield, 2016) and elucidated the powerful influence of the high-capacity attention mechanisms in transformer architectures (Ramsauer et al., 2020). At the heart of these memory models is a fundamental restrictive assumption: the memories, or the information the network uses to make predictions, are fixed prior to inference, stored in synaptic interactions, and any retrieval process can only access these pre-determined memories. This holds true in both single memory retrieval models (Hopfield, 1982; Krotov and Hopfield, 2020) or the more recent extensions to sequence memory models (Karuvally et al., 2022; Chaudhry et al., 2023). However, this fixed-memory assumption limits the purview of memory models to address a limited set of questions pertaining to synaptic memory storage in neural architectures. In addition to synaptic memory storage, neural networks can store memories in the neural activity as a working memory. Although it might seem like the working memory does not

Associative Memory & Hopfield Networks in 2023. NeurIPS 2023 workshop.

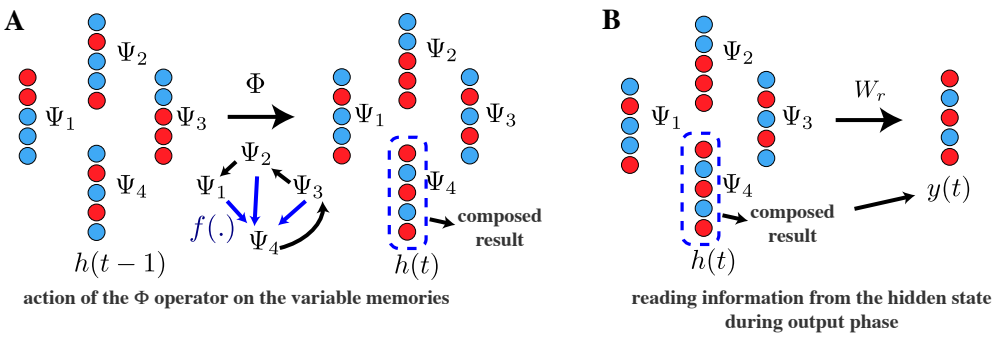



Figure 1: **Generalizable process on variable memories in an illustrative task of four variables, each with five dimensions**: **A**. The hidden state at time $t$ has subspaces capable of storing external information in their activities. The colors depict the vector components (or activity) of the hidden state in the variable memory basis. The linear operator $\Phi$ acts on the hidden state such that these activities are copied between variable memories except for $\Psi_4$, which implements the a linear function $f$. **B**. The $N^{\text{th}}$ variable contents are read during using appropriate linear operator $W_r = \Psi_N^*$.

rely on long term memories, they are essential in storing variable information for running abstract computational *processes* stored in the long term memory. For instance, consider a neural network learning to compute the sum of two numbers. With the fixed memory assumption, it would have to store all possible pairings of numbers and their corresponding sums. But neural networks have demonstrated the ability to learn and store the computational process of "addition" itself, instead of storing individual outcomes for every possible input number pairs. Essentially, neural networks treat the numbers as variables (stored in the neural activity), then a learned generalizable process (encoded in the synapses) works on these variables to produce the necessary computation.

In this study, we address the limitation of fixed memories by proposing "variable memories", which are memories stored in the activity of subspaces spanned by the fixed memories. We show how neural networks use the inter-memory interactions of the sequence memory models to store computational *processes* like variable addition which computes over the variables. This exploration not only offers a new perspective on the memory paradigm but also opens up possibilities for the application of memory models in the mechanistic interpretations of neural networks.

## 2 Related Works

**Mechanistic Interpretability**: Mechanistic interpretability seeks to reverse-engineer neural networks to expose the underlying mechanisms enabling them to learn and adapt to previously unencountered conditions. The prevailing strategy involves examining the networks' internal "circuits" (Conmy et al., 2023; Wang et al., 2022; Cammarata et al., 2020). Despite advancements, current approaches remain confined to networks without a recurrent state like MLPs and transformers. Recurrent architectures, which maintain and recursively update a hidden state for working memory storage, present a unique challenge - information needs to be stored and processed over time (Cruse, 1996; Hochreiter and Schmidhuber, 1997).

## 3 Variable Memories

We begin our discussion of variable memories by considering a discrete version of a recently introduced sequence memory model with a pseudoinverse learning rule (Chaudhry et al., 2023) and a readout layer. Specifically, we consider a discrete version of the Dense Sequence Memory Model (Karuvally et al., 2022) under the following conditions: $\mathcal{T}_f = 1, \mathcal{T}_d = 0$, and both the activation functions $\sigma_f$, $\sigma_h$ are identity functions. The activation $\sigma_h$ is identity implies that the model is not dense, and $\sigma_f$ is identity means that the system is fully linear. It is essential to note that although we formulate variable memories on a fully linear model, the mechanisms of information storage are general enough to be applicable even in non-linear cases. Formally, we introduce variable memories

in a linearized system defined by the following equations.

$$\begin{cases} h(t) = \Xi\, \Phi\, \Xi^{\dagger}\, h(t-1)\,, \\ y(t) = W_r\, h(t)\,. \end{cases} \tag{1}$$

Here, $\Xi, \Phi, W_r$ are linear operators, $h(t)$ is the hidden state, and $y(t)$ is the output. We use a simplifying assumption that $\Xi$ has sufficient dimensions to represent all the variables required for variable binding tasks. We further assume that $h(0)$ is the zero vector. We define some terms that are typical in the memory modeling literature. **stored memories**: The columns of the matrix $\Xi$ are called the stored memories. In memory modeling, these vectors represent the pre-determined memories that are stored and retrieved from the system. **inter-memory interactions**: The $\Phi$ matrix in sequence memory modeling is an adjacency matrix containing information about how the stored memories interact over time. It is typically asymmetric which lends temporal behavior for the sequence memory models.

To store variable information, we propose the stored memories as defining a vector space whose activity is the necessary information to store. We make the assumption here that the stored memories are linearly independent and has sufficient rank to encode the necessary information. To deal with this new perspective on memories, we introduce the Dirac and Einstein notations from abstract algebra (Appendix A.1). This new notation has two benefits - (1) the ability formalize variable binding mechanisms independent of the basis, and (2) enables clear and concise representations. Formally, we write any vector $v$ as $|v\rangle = \langle \epsilon^i | v \rangle\, |\epsilon_j\rangle = v^i\, |\epsilon_j\rangle$. $|\epsilon_j\rangle$ is the basis in which the vector has vector components $v^i = \langle \epsilon^i | v \rangle$. $\langle \epsilon^i |$ are the basis *covectors* defined such that $\langle \epsilon^i | \epsilon_j \rangle = \delta_{ij}$, and $\delta_{ij}$ is the Dirac delta function.

In the new notation, the model in Equation 1 evolving autonomously (without input) is reformulated as follows.

$$|h(t)\rangle = \left(\xi^i_\mu\, \Phi^\mu_\nu\, (\xi^\dagger)^\nu_j\right)\, |e_i\rangle\, \langle e^j |\, |h(t-1)\rangle \tag{2}$$

Here $|e_i\rangle$ is the standard basis vector which is typically used in simulations. To simplify this system further, consider a new basis $|\psi_\mu\rangle = \xi^i_\mu\, |e_i\rangle$.

$$|h(t+1)\rangle = (\Phi^\mu_\nu\, |\psi_\mu\rangle\, \langle \psi^\nu |)\, |h(t)\rangle \tag{3}$$

In the new basis, the hidden state vector is $|h(t)\rangle = h^{\psi_\mu}\, |\psi_\mu\rangle$. The new basis allows us to treat the model as applying a *single* linear operator as opposed to three in the original formulation. Further, the basis allows to view the stored memories as forming a vector space capable of storing variable information in the hidden state activity.

A single fixed memory can store a real valued number as activity in its space. If we are required to store a vector of real numbers, we need a collection of stored memories. This is a different notion of memory storage where memories are stored in activity and not synapses meaning they can adapt to different user inputs without change to any of the synapses. To highlight this difference in memory storage, we call it the variable memory. The **variable memory** is a collection of stored memories $\{\Psi_i\} = \{|\psi_\mu\rangle : \mu \in \{(i-1)d, \ldots, id\}\}$ that forms a vector space storing memories as activity in the space. Note that in the variable memory, the inter-memory interactions is encoding a *process* which governs how the variables change over time rather than strict transition rules between fixed memory items.

**Example: Generalized Fibonacci Series.** We consider a generalization of the Fibonacci sequence where each element $F_n \in \mathbb{R}^d$ is a vector defined recursively as,

$$F_n = \begin{cases} u^1 & n = 1 \\ u^2 & n = 2 \\ \vdots & \\ u^s & n = s \\ \sum_{t=n-s}^{n-1} F_t & n > s \end{cases} \tag{4}$$

For any $u^i \in \mathbb{R}^d$. In order to store this *process* of generating sequences of $F_n$, the vectors $u_1, u_2, \ldots u_s$ needs to be stored as variables and recursively added to produce new outputs. In our framework, this can be accomplished by initializing the hidden state such that

| Task | hidden size: 64 | | | hidden size: 128 | | |
|---|---|---|---|---|---|---|
| | L2: 0.0 | L2: 0.001 | L2: 0.1 | L2: 0.0 | L2: 0.001 | L2: 0.1 |
| $\mathcal{T}_1$ | — (0.97) | — (0.88) | — (0.50) | 0.0005 (1.00) | — (0.93) | — (0.50) |
| $\mathcal{T}_2$ | 0.0075 (1.00) | — (0.85) | — (0.50) | 0.0055 (0.98) | 0.0031 (0.98) | — (0.50) |
| $\mathcal{T}_3$ | 0.0026 (1.00) | 0.0010 (0.97) | — (0.50) | 0.0031 (0.98) | 0.0005 (1.00) | — (0.50) |
| $\mathcal{T}_4$ | 0.0112 (0.94) | 0.0011 (1.00) | — (0.50) | 0.0022 (1.00) | 0.0006 (1.00) | — (0.50) |

Table 1: **RNNs consistently converge to the theoretical** $\Phi$: The table shows the MAE in the complex argument between the eigenspectrum of the predicted $\Phi$ from the variable binding circuit and the empirically learned $W_{hh}$ in 4 tasks across 20 seeds under different RNN configurations. This average error is indeterminate if the rank of the theoretical $\Phi$ is different from the empirical $W_{hh}$. Values in the brackets show the average test accuracy of the trained model. For models that have high test accuracy ($> 0.94$), the error in the theoretically predicted spectrum is very low indicating consistent convergence. A notable exception of this behavior is $\mathcal{T}_1$ with hidden size$= 64$ and $L2 = 0$, where the restricted availability of dimensions forces the network to encode variables in bottleneck superposition resulting in a low-rank representation of the solution.

$|h(s)\rangle = \sum_i \sum_{\psi_\mu \in \{\Psi_i\}} u_i^\mu \, |\psi_\mu\rangle$, that is each $u^\mu$ is stored as activity of distinct subspaces of the hidden state. To encode the Fibonacci process in $\Phi$, we propose the following form for the inter-memory interactions.

$$\Phi = \sum_{\mu=1}^{(s-1)d} |\psi_\mu\rangle \langle\psi^{\mu+d}| + \underbrace{\left( \sum_{\mu=1}^{d} |\psi_{(s-1)d+\mu}\rangle \left( \sum_{\nu=0}^{s-1} \langle\psi^{\nu d+\mu}| \right) \right)}_{\sum_{t=n-s}^{n-1} F_t} . \tag{5}$$

This form of $\Phi$ has two parts. The first part implements a variable shift operation. The second part implements the summation function of Fibonacci. Since the hidden state is initialized with all the starting variables $u^\mu$, application of the $\Phi$ operator repeatedly, produces the next element in the sequence. As of now, the hidden state contains all the elements in the sequence. The abstract algebra notation allows proposing $W_r$ which will extract only the required output. Formally,

$$W_r = \Psi_s^* = \sum_{\mu=(s-1)d+1}^{sd} |e_{\mu-(s-1)d}\rangle \langle\psi^\mu| . \tag{6}$$

It is the dual space operator which extracts the contents of the $N^{\text{th}}$ variable memory in the standard basis. Note that the process works irrespective of the actual values of $u^\mu$. To summarize, we now have a memory model encoding a generalizable process of fibonacci sequence generation (Figure 1).

## 4 Mechanistic Interpretation of RNNs

The concept of variable memories is intimately connected to the concept of feature representations typically studied in the mechanistic interpretation of neural networks. A predominant hypothesis in the mechanistic interpretability literature is the *linear superposition hypothesis* (Elhage et al., 2022). The hypothesis states that neural networks represent features required for computation in linear subspaces, and the neural activity exists as a superposition of activity in these subspaces. The concept of variable memories provides a principled framework to rigorously deal with these types of feature representations. To validate that the form of $\Phi$ we proposed in Equation 5 is general and learned by practically trained RNNs, we trained Elman RNNs by backpropagation on four general sequence generation tasks ($\mathcal{T}_i \, , i \in \{1, \ldots 4\}$) similar to the generalized Fibonacci (Equation 4) but with more general linear functions than vector addition (see Appendix B.3 for details). We then compared the eigenspectrum of the Jacobian of the RNN after linearization with the spectrum of the theoretical $\Phi$. The spectrum of the Jacobian helps in determining the long term behavior of dynamical systems (Sussillo and Barak, 2013; Strogatz, 1994). Our results summarized in Table 1 reveal that trained RNNs consistently converge to the theoretical mechanisms.

## 5 Conclusion

We presented the novel concept of variable memories - a formal approach to dealing with memories that are stored in the activation of neurons. We showed how the fixed memory assumption is lifted by treating the stored memories of memory models as defining a basis for the storage of variable information. By demonstrating its applicability in trained RNNs, we point to its applicability in practical scenarios. We envision that the theory will provide a new path forward for memory modeling and increase its purview to the fundamental challenge of the mechanistic interpretation of learned neural networks.

## References


Cammarata, N., Carter, S., Goh, G., Olah, C., Petrov, M., Schubert, L., Voss, C., Egan, B., and Lim, S. K. (2020). Thread: Circuits. *Distill*. https://distill.pub/2020/circuits.

Chaudhry, H. T., Zavatone-Veth, J. A., Krotov, D., and Pehlevan, C. (2023). Long sequence hopfield memory. *ArXiv*, abs/2306.04532.

Conmy, A., Mavor-Parker, A. N., Lynch, A., Heimersheim, S., and Garriga-Alonso, A. (2023). Towards automated circuit discovery for mechanistic interpretability. *ArXiv*, abs/2304.14997.

Cruse, H. (1996). Neural networks as cybernetic systems.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T. J., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., Grosse, R. B., McCandlish, S., Kaplan, J., Amodei, D., Wattenberg, M., and Olah, C. (2022). Toy models of superposition. *ArXiv*, abs/2209.10652.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9:1735–1780.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79 8:2554–8.

Karuvally, A., Sejnowski, T. J., and Siegelmann, H. T. (2022). Energy-based general sequential episodic memory networks at the adiabatic limit. *ArXiv*, abs/2212.05563.

Krotov, D. and Hopfield, J. J. (2016). Dense associative memory for pattern recognition. *ArXiv*, abs/1606.01164.

Krotov, D. and Hopfield, J. J. (2020). Large associative memory problem in neurobiology and machine learning. *ArXiv*, abs/2008.06996.

Ramsauer, H., Schafl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Pavlovi'c, M., Sandve, G. K., Greiff, V., Kreil, D. P., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. (2020). Hopfield networks is all you need. *ArXiv*, abs/2008.02217.

Strogatz, S. H. (1994). Nonlinear dynamics and chaos: With applications to physics, biology, chemistry and engineering. *Physics Today*, 48:93–94.

Sussillo, D. and Barak, O. (2013). Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Computation*, 25:626–649.

Wang, K., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. (2022). Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *ArXiv*, abs/2211.00593.

# A Theoretical Models of Variable Binding

## A.1 Mathematical Preliminaries

The core concept of the episodic memory theory is basis change, the appropriate setting of the stored memories. Current notations lack the ability to adequately capture the nuances of basis change. Hence, we introduce abstract algebra notations typically used in theoretical physics literature to formally explain the variable binding mechanisms. We treat a *vector* as an abstract mathematical object invariant to basis changes. Vectors have *vector components* that are associated with the respective basis under consideration. We use Dirac notations to represent vector $v$ as - $|v\rangle = \sum_i v^i |e_i\rangle$. Here, the linearly independent collection of vectors $|e_i\rangle$ is the *basis* with respect to which the vector $|v\rangle$ has component $v^i \in \mathbb{R}$. Linear algebra states that a collection of basis vectors $|e_i\rangle$ has an associated collection of *basis covectors* $\langle e^i|$ defined such that $\langle e^i|e_j\rangle = \delta_{ij}$, where $\delta_{ij}$ is the Kronecker delta. This allows us to reformulate the vector components in terms of the vector itself as $|v\rangle = \sum_i \langle e^i|v\rangle |e_i\rangle$. We use the Einstein summation convention to omit the summation symbols wherever the summation is clear. Therefore, vector $|v\rangle$ written in basis $|e_i\rangle$ is

$$\begin{aligned} |v\rangle &= v^i |e_i\rangle \\ &= \langle e^i|v\rangle |e_i\rangle . \end{aligned} \tag{7}$$

The set of all possible vectors $|v\rangle$ is a *vector space* spanned by the basis vectors $|e_i\rangle$. A *subspace* of this space is a vector space that is spanned by a subset of basis vectors $\{|e'_j\rangle : |e'_j\rangle \subseteq \{|e_i\rangle\}\}$.

The RNN dynamics presented in Equation **??** represented in the new notation is reformulated as:

$$\begin{aligned} |h(t)\rangle &= \sigma_f \left( \left( \xi^i_\mu \, \Phi^\mu_\nu \, (\xi^\dagger)^\nu_j \; |e_i\rangle \langle e^j| \right) |h(t-1)\rangle \right) \\ &= \sigma_f \left( W_{hh} \, \vec{h}(t-1) \right) . \end{aligned} \tag{8}$$

The greek indices iterate over memory space dimensions $\{1, 2, \ldots, N_h\}$, alpha numeric indices iterate over feature dimension indices $\{1, 2, \ldots, N_f\}$. Typically, we use the standard basis in our simulations. For the rest of the paper, the standard basis will be represented by the collection of vectors $|e_i\rangle$ and the covectors $\langle e^i|$. The hidden state at time $t$ in the standard basis is denoted as $|h(t)\rangle = \langle e^j|h(t)\rangle |e_i\rangle$. $\langle e^j|h(t)\rangle$ are the *vector components* of $|h(t)\rangle$ we obtain from simulations.

## A.2 Example: Repeat Copy ($\mathcal{T}_1$)

Repeat Copy is a task typically used to evaluate the memory storage characteristics of RNNs since the task has a deterministic evolution represented by a simple algorithm that stores all input vectors in memory for later retrieval. Although elementary, repeat copy provides a simple framework to work out the variable binding circuit we theorized in action. For the repeat copy task, the linear operators of the RNN has the following equations.

$$\begin{cases} \Phi = \sum_{\mu=1}^{(s-1)\kappa} |\psi_\mu\rangle \langle \psi^{\mu+\kappa}| + \sum_{\mu=(s-1)\kappa+1}^{s\kappa} |\psi_\mu\rangle \langle \psi^{\mu-(s-1)\kappa}| \\ W_{uh} = \Psi_s \\ W_r = \Psi_s^* \end{cases} \tag{9}$$

This $\phi$ can be imagined as copying the contents of the subspaces in a cyclic fashion. That is, the content of the $i^{th}$ subspace goes to $(i-1)^{\text{th}}$ subspace with the first subspace being copied to the $N^{\text{th}}$ subspace. The dynamical evolution of the RNN is represented at the time step 1 as,

$$|h(1)\rangle = |\psi_{(s-1)\kappa+j}\rangle \langle e^j| \, u^i(1) \, |e_i\rangle \tag{10}$$

$$|h(1)\rangle = u^i(1) \, |\psi_{(s-1)\kappa+j}\rangle \langle e^j|e_i\rangle \tag{11}$$

$$|h(1)\rangle = u^i(1) \, |\psi_{(s-1)\kappa+j}\rangle \, \delta_{ij} \tag{12}$$

Kronecker delta index cancellation

$$|h(1)\rangle = u^i(1) \left|\psi_{(s-1)\kappa+i}\right\rangle \tag{13}$$

At time step 2,

$$|h(2)\rangle = u^i(1)\, \Phi\, \left|\psi_{(s-1)\kappa+i}\right\rangle + u^i(2)\, \left|\psi_{(s-1)\kappa+i}\right\rangle \tag{14}$$

Expanding $\Phi$

$$\begin{aligned} |h(2)\rangle = u^i(1) & \left( \sum_{\mu=1}^{(s-1)\kappa} |\psi_\mu\rangle \left\langle \psi^{\mu+\kappa} \right| + \sum_{\mu=(s-1)\kappa+1}^{s\kappa} |\psi_\mu\rangle \left\langle \psi^{\mu-(s-1)\kappa} \right| \right) \left|\psi_{(s-1)\kappa+i}\right\rangle \\ & + u^i(2)\, \left|\psi_{(s-1)\kappa+i}\right\rangle \end{aligned} \tag{15}$$

$$|h(2)\rangle = u^i(1)\, \left|\psi_{(s-2)\kappa+i}\right\rangle + u^i(2)\, \left|\psi_{(s-1)\kappa+i}\right\rangle \tag{16}$$

At the final step of the input phase when $t = s$, $|h(s)\rangle$ is defined as:

$$|h(s)\rangle = \sum_{\mu=1}^{s} u^i(\mu)\, \left|\psi_{(\mu-1)\kappa+i}\right\rangle \tag{17}$$

For $t$ timesteps after $s$, the general equation for $|h(s+t)\rangle$ is:

$$|h(s+t)\rangle = \sum_{\mu=1}^{s} u^i(\mu)\, \left|\psi_{[((\mu-t-1 \mod s)+1)\kappa+i]}\right\rangle \tag{18}$$

From this equation for the hidden state vector, it can be easily seen that the $\mu^{\text{th}}$ variable is stored in the $[(\mu - t - 1 \mod s) + 1]^{\text{th}}$ subspace at time step $t$. The readout weights $W_r = \Psi_s^*$ reads out the contents of the $s^{\text{th}}$ subspace.

# B Experiments

## B.1 Variable Binding Tasks

Simple algorithmic tasks are very useful for mechanistic interpretability as they provide a controllable empirical setup compared to complex and noisy real-world scenarios. We formulate a class of tasks with input and output phases. At each timestep of the input phase, external information is provided *to* the RNN. During the output phase, the RNN needs to utilize this external information to synthesize novel outputs at each time step that are subsequently read *from* the network as output. This simple two-phase setup closely matches the behavior of NLP tasks like translation and conditional generative modeling where the noisy real-world features are abstracted out. Formally, the input phase consists of $s$ total timesteps where at each timestep $t$, a vector of $d$ dimensions $u(t) = \left(u^1(t), u^2(t), \ldots, u^d(t)\right)^\top$ is input to the model. We call the vector components $u^i(t)$ the external information that needs to be *stored* in the RNN hidden state. After the input phase is complete, the zero vector is continually passed as input to the model, so we say the RNN evolves autonomously (without any external input) during the output phase. The future states of the system during output phase evolve according to the following equation.

$$u(t) = f(u(t-1), u(t-2), \ldots u(t-s)), \quad t > s. \tag{19}$$

For analytical tractability, we add two restrictions to the variable binding tasks: (1) the composition function $f$ is a linear function of its inputs. (2) the codomain of $f$ and the domain of the inputs is binary $\in \{-1, 1\}$.

### B.2 Data

We train RNNs on the variable binding tasks described in the main paper with the following restrictions - the domain of $u$ at each timestep is binary $\in \{-1, 1\}$ and the function $f$ is a linear function of its inputs. We collect various trajectories of the system evolving accoding to $f$ by first sampling uniformly randomly, the input vectors. The system is then allowed to evolve with the recurrent function $f$ over the time horizon defined by the training algorithm.

### B.3 Training Details

**Architecture** We used single layer Elman-style RNNs for all the variable binding tasks. Given an input sequence $(u(1), u(2), ..., u(T))$ with each $u(t) \in \mathbb{R}^d$, an Elman RNN produces the output sequence $y = (y(1), ..., y(T))$ with $y(t) \in \mathbb{R}^{N_{out}}$ following the equations

$$h(t+1) = \tanh(W_{hh}h(t) + W_{uh}u(t)) \quad , \quad y(t) = W_r h(t) \tag{20}$$

Here $W_{uh} \in \mathbb{R}^{N_h \times N_{in}}$, $W_{hh} \in \mathbb{R}^{N_h \times N_h}$, and $W_r \in \mathbb{R}^{N_{out} \times N_h}$ are the input, hidden, and readout weights, respectively, and $N_h$ is the dimension of the hidden state space.

The initial hidden state $h(0)$ for each model was *not* a trained parameter; instead, these vectors were simply generated for each model and fixed throughout training. We used the zero vector for all the models.

**Task Dimensions** Our results presented in the main paper for the repeat copy ($\mathcal{T}_1$) and compose copy ($\mathcal{T}_2$) used vectors of dimension $d = 8$ and sequences of $s = 8$ vectors to be input to the model.

**Training Parameters** We used PyTorch's implementation of these RNN models and trained them using Adam optimization with MSE loss. We performed a hyperparameter search for the best parameters for each task — see table 2 for a list of our parameters for each task.

| | Repeat Copy ($\mathcal{T}_1$) | Compose Copy ($\mathcal{T}_2$) |
|---|---|---|
| Input & output dimensions | 8 | 8 |
| Input phase (# of timesteps) | 8 | 8 |
| training horizon | 100 | 100 |
| Hidden dimension $N_h$ | 128 | 128 |
| # of training iterations | 45000 | 45000 |
| (Mini)batch size | 64 | 64 |
| Learning rate | $10^{-3}$ | $10^{-3}$ |
| applied at iteration | 36000 | |
| Weight decay ($L^2$ regularization) | none | none |
| Gradient clipping threshold | 1.0 | 1.0 |

Table 2: Architecture, Task, & Training Parameters

**Curriculum Time Horizon** When training a network, we adaptively adjusted the number of timesteps after the input phase during which the RNN's output was evaluated. We refer to this window as the *training horizon* for the model.

Specifically, during training we kept a rolling average of the model's *loss by timestep* $L(t)$, i.e. the accuracy of the model's predictions on the $t$-th timestep after the input phase. This metric was computed from the network's loss on each batch, so tracking $L(t)$ required minimal extra computation.

The network was initially trained at time horizon $H_0$ and we adapted this horizon on each training iteration based on the model's loss by timestep. Letting $H_n$ denote the time horizon used on training step $n$, the horizon was increased by a factor of $\gamma = 1.2$ (e.g. $H_{n+1} \leftarrow \gamma H_n$) whenever the model's accuracy $L(t)$ for $t \leq H_{\text{min}}$ decreased below a threshold $\epsilon = 3 \cdot 10^{-2}$. Similarly, the horizon was reduced by a factor of $\gamma$ is the model's loss was above the threshold ($H_{n+1} \leftarrow H_n/\gamma$). We also restricted $H_n$ to be within a minimum training horizon $H_0$ and maximum horizon $H_{\text{max}}$. These where set to 10/100 for the repeat copy task and 10/100 for the compose copy task.

We found this algorithm did not affect the results presented in this paper, but it did improve training efficiency, allowing us to run the experiments for more seeds.

### B.4 Repeat Copy: Further Examples of Hidden Weights Decomposition

This section includes additional examples of the hidden weights decomposition applied to networks trained on the repeat copy task.

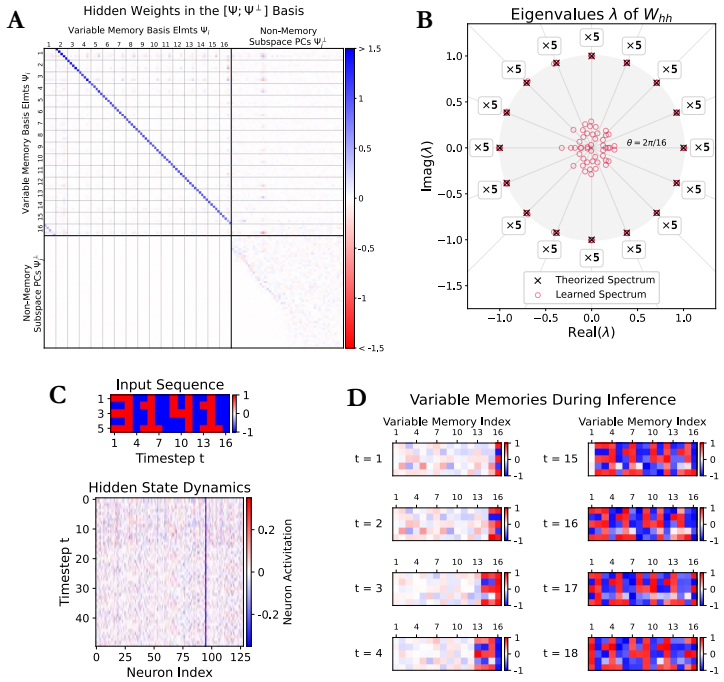



Figure 2: **Additional Experimental Results of Repeat Copy Task with 16 vectors, each of 5 dimensions**: **A**. $W_{hh}$ visualized in the variable memory basis reveals the variable memories and their interactions. **B**. After training, the eigenspectrum of $W_{hh}$ with a magnitude $\geq 1$ overlaps with the theoretical $\Phi$. The boxes show the number of eigenvalues in each direction. **C**. During inference, "3141" is inserted into the network in the form of binary vectors. This input results in the hidden state evolving in the standard basis, as shown. How this hidden state evolution is related to the computations cannot be interpreted easily in this basis. **D**. When projected on the variable memories, the hidden state evolution reveals the contents of the variables over time. Note that in order to make these visualization clear, we needed to normalize the activity along each variable memory to have standard deviation 1 when assigning colors to the pixels. The standard deviation of the memory subspaces varies due to variance in the strength of some variable memory interactions. These differences in interaction strengths does not impede the model's performance, however, likely due to the nonlinearity of the activation function. Unlike the linear model, interaction strengths well above 1 cannot cause hidden state space to expand indefinitely because the tanh nonlinearity restricts the network's state to $[-1, 1]^{N_h}$. This property appears to allow the RNN to sustain stable periodic cycles for a range of interaction strengths above 1.

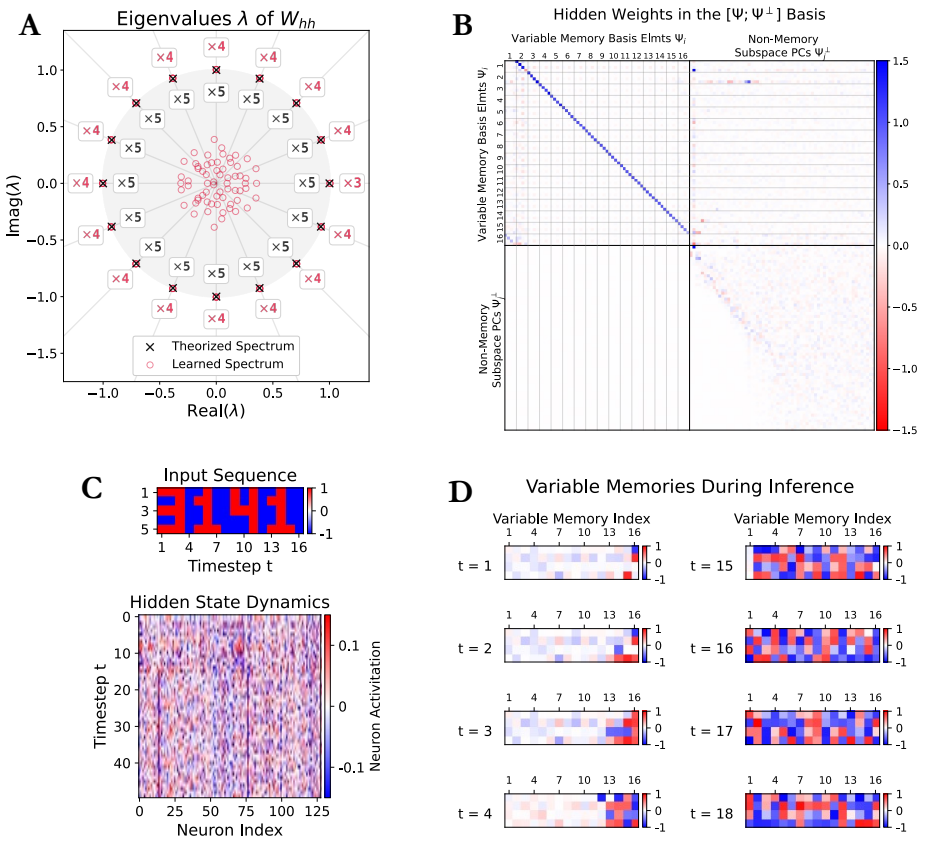



Figure 3: **Nonlinear Variable Memories Learned for the Repeat Copy Task with 16 vectors, each of 5 dimensions**. **A**. Eigenspectrum of $W_{hh}$ after training. The learned eigenvalues cluster into 16 groups equally spaced around the unit circle, but there are only 3-4 eigenvectors per cluster (indicated in red). Compare this to the theorized linear model, which has 5 eigenvalues per cluster (indicated in black). The task requires $16 \cdot = 5 = 80$ bits of information to be stored. Linearization about the origin predicts that the long-term behavior of the model is dictated by the eigenvectors with eigenvalue outside the unit circle because its activity along other dimensions will decay over time. The model has only $16 \cdot \mathbf{4} - 1 = 63$ eigenvectors with eigenvalue near the unit circle, so this results suggests the model has learned a non-linear encoding that compresses 80 bits of information into 63 dimensions. **B**: $W_{hh}$ visualized in the variable memory basis reveals the variable memories and their interactions. Here, the variable memories have only 4 dimensions because the network has learned only 63 eigenvectors with eigenvalue near the unit circle. The variable memory subspaces also have non-trivial interaction with a few of the the non-memory subspaces. **C**. During inference, "3141" is inserted into the network in the form of binary vectors. This input results in the hidden state evolving in the standard basis, as shown. How this hidden state evolution is related to the computations cannot be interpreted easily in this basis. **D**. When projected on the variable memories, the hidden state evolution is still not easily interpreted for this network, likely due to a nonlinear variable memories. As in the previous figure, we normalized the activity along each variable memory to have standard deviation 1 when assigning colors to the pixels.

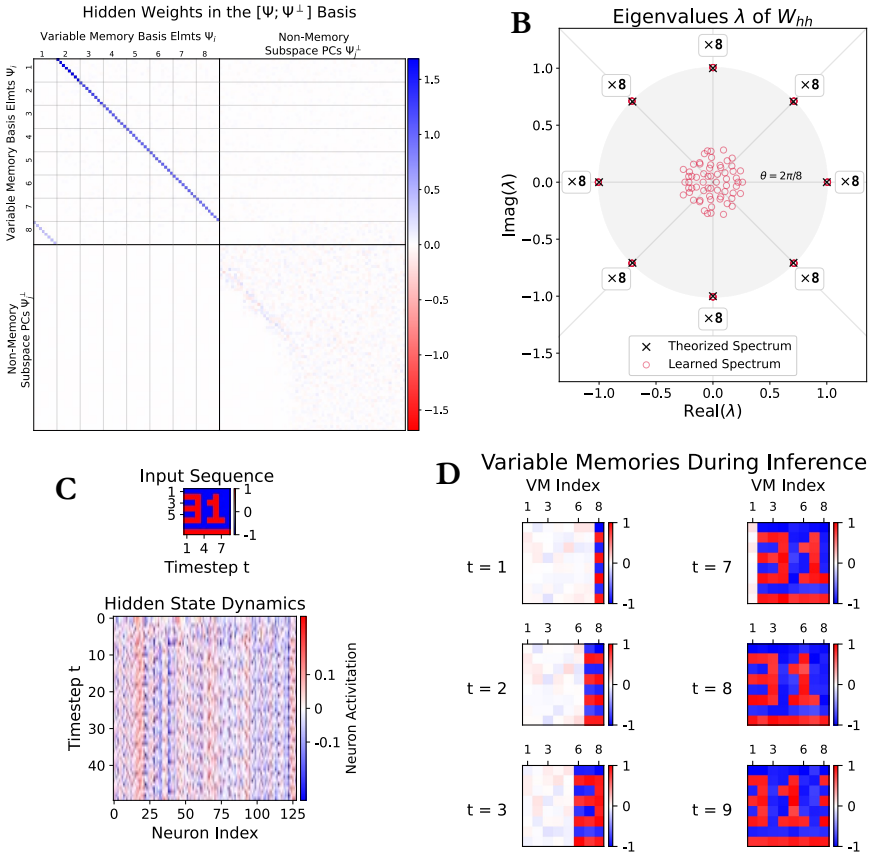
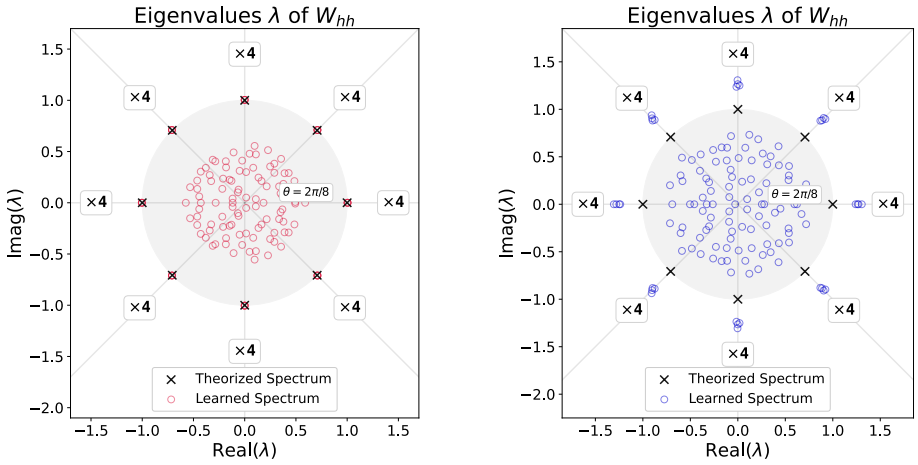



Figure 4: **Additional Experimental Results of Repeat Copy Task with 8 vectors, each of 8 dimensions**. This figure was included to show the decomposition applied to other values of $s$ and $d$ for the Repeat Copy task. **A**: $W_{hh}$ visualized in the variable memory basis reveals the variable memories and their interactions. **B**. After training, the eigenspectrum of $W_{hh}$ with a magnitude $\geq 1$ overlaps with the theoretical $\Phi$. The boxes show the number of eigenvalues in each direction. **C**. During inference, "3141" is inserted into the network in the form of binary vectors. This input results in the hidden state evolving in the standard basis, as shown. How this hidden state evolution is related to the computations cannot be interpreted easily in this basis. **D**. When projected on the variable memories, the hidden state evolution reveals the contents of the variables over time. As in the previous figures, we normalized the activity along each variable memory to have standard deviation 1 when assigning colors to the pixels.

### B.5 Uniform vs. Gaussian Parameter Initialization

We also tested a different initialization scheme for the parameters $W_{uh}, W_{hh}$, and $W_{hy}$ of the RNNs to observe the effect(s) this would have on the structure of the learned weights. The results presented in the main paper and in earlier sections of the Supplemental Material used PyTorch's default initialization scheme: each weight is drawn *uniformly* from $[-k, k]$ with $k = 1/\sqrt{N_h}$. Fig. 5 shows the resulting spectrum of a trained model when it's parameters where drawn from a Gaussian distribution with mean 0 and variance $1/N_h$. One can see that this model learned a spectrum similar to that presented in the main paper, but the largest eigenvalues are further away from the unit circle. This result was observed for most seeds for networks trained on the repeat copy task with $s = 8$ vectors of dimension $\kappa = 4$ and $\kappa = 8$, though it doesn't hold for every seed. We also find that the networks whose spectrum has larger eigenvalues usually generalize longer in time than the networks with eigenvalues closer to the unit circle.




(a) Uniform Initialization

(b) Gaussian Initialization

Figure 5: **An effect of parameter initialization for the Repeat Copy Task** with $s = 8$ vectors, each of dimension $d = 4$. **A**: Spectrum (in red) of the learned hidden weights $W_{hh}$ for a network whose parameters where initialized from a uniform distribution over $[-k, k]$ with $k = 1/\sqrt{N_h}$. This network has 32 eigenvalues that are nearly on the unit circle. These eigenvalues are clustered into groups of 4, each group being an angle of $\theta = 2\pi/s$ apart from each other. These eigenvalues coincide with the eigenvalues of the linear model for solving the repeat copy task. **B**: Spectrum (in blue) of the learned hidden weights $W_{hh}$ for a network whose parameters where initialized from a Gaussian distribution with mean 0 and variance $1/N_h$. This network has 32 eigenvalues outside the unit circle, but they are a larger radii than the model initialized using the uniform distribution. These eigenvalues still cluster into eight groups of four, and the average complex argument of each group aligns with the complex arguments of the eigenvalues for the linear model.